Query Games in Databases

Ester Livshits
Technion, Haifa, Israel
esterliv@cs.technion.ac.il

Leopoldo Bertossi
Univ. Adolfo Ibáñez and
Millennium Inst. Foundations
of Data (IMFD), Chile
leopoldo.bertossi@uai.cl

Benny Kimelfeld
Technion, Haifa, Israel
bennyk@cs.technion.ac.il

Moshe Sebag
Technion, Haifa, Israel
moshesebag@campus.technion.ac.il

ABSTRACT

Database tuples can be seen as players in the game of jointly realizing the answer to a query. Some tuples may contribute more than others to the outcome, which can be a binary value in the case of a Boolean query, a number for a numerical aggregate query, and so on. To quantify the contributions of tuples, we use the Shapley value that was introduced in cooperative game theory and has found applications in a plethora of domains. Specifically, the Shapley value of an individual tuple quantifies its contribution to the query. We investigate the applicability of the Shapley value in this setting, as well as the computational aspects of its calculation in terms of complexity, algorithms, and approximation.

1. INTRODUCTION

In data management, so as in artificial intelligence (AI), there is an increasing need for characterizing and computing explanations of the outcomes of algorithms. In AI, this is commonly attempted for classification algorithms. In relational databases, query answering may be the operation that requires explanations. As we will see later on, the two areas share concerns and approaches. On the databases side, we may want to explain *why* we obtained a particular answer to a query; or why we *did not* get some other answer we may have in mind. As expected, the explanations depend on the underlying data and the query itself. (We assume that the query-answering algorithm is correct.) In this work, we concentrate on the positive case, that is, on why an answer was obtained (as opposed to not obtained).

Explanations can be given in terms of individual tuples of the underlying database that contribute to the answer or in terms of the *provenance* or *lineage* of the query [10, 18, 35], that explicitly describe the logical connection between a generic query answer and the possible tuples of the database. In this sense, provenance implicitly describes *how* a query answer can be obtained. In this work, we focus mostly on the identification of individual tuples that contribute to a query answer, through, and accompanied by, their quantitative "degree of contribution."

Example 1. Consider the following relational database D, with the table *Store* representing official stores, and the table *Receives* for stores receiving goods from other stores:

Receives	receiver	sender
	s_2	s_1
	s_3	s_3
	s_4	s_3

Store	store
	s_2
	s_3
	s_4

For accounting purposes, a store could be its own supplier, as shown by the tuple $\mathit{Receives}(s_3, s_3)$. The query that asks whether there are pairs of official stores in a receiving relationship is expressed (in predicate logic) as

$$\mathcal{Q}\colon\ \exists x \exists y (\mathit{Store}(x) \wedge \mathit{Receives}(x, y) \wedge \mathit{Store}(y)) \qquad (1)$$

and we use $D \models \mathcal{Q}$ to denote that it holds true in the database D. This is a *conjunctive query* (CQ), as it is built as a conjunction of *atoms* (database predicates instantiated on constants or variables) preceded by a sequence of existential quantifiers.

The question is about the tuples (i.e., rows in tables) that cause this query to be true (or lead to the answer *yes*). In this case, among the tuples that contribute to the answer we find the tuples $\mathit{Store}(s_4)$, $\mathit{Receives}(s_4, s_3)$ and $\mathit{Store}(s_3)$ that, taken together, make $\mathcal{Q}$ evaluate to true. □

In Example 1 we informally used the notion of a *cause*. Actually, there is a precise notion of a tuple as an *actual cause* for a query answer [26, 27]. This notion was borrowed from a more general notion of actual causality [20] that defines causes in terms of counterfactual interventions (cf., e.g., [19]). These are hypothetical, exploratory updates on a variable of a (structural) model that are made for detecting whether the output changes. For monotone queries, such as (1) whose answer set can only grow when the database grows, the relevant interventions are tuple deletions and changes of attribute values. In this work, we stick to the former. Conjunctive queries are always monotone, and so are *unions (disjunctions) of conjunctive queries* (UCQs).

Actual causality can be extended by means of a measure of *causal responsibility* that quantifies the strength of an actual cause for the outcome at hand [12]. Responsibility can be applied to database tuples, to capture in quantitative terms the relevance of a tuple for the query result [9, 27]. We introduce and illustrate actual causality and responsibility on our running example.

Example 2. (ex. 1 cont.) The tuple *Store*(s_3) is a *counterfactual cause* for $\mathcal{Q}$ (to be true in D), because: (a) $D \models \mathcal{Q}$, and (b) $D \smallsetminus \{\mathit{Store}(s_3)\} \not\models \mathcal{Q}$; that is, $\mathcal{Q}$ is no longer true once *Store*(s_3) is removed.

Now, *Receives*(s_4, s_3) is an *actual cause*, because: (a) $D \models \mathcal{Q}$, and (b) there is $\Gamma \subseteq D$ with $\mathit{Receives}(s_4, s_3) \notin \Gamma$, such that $D \smallsetminus \Gamma \models \mathcal{Q}$, but $D \smallsetminus (\Gamma \cup \{\mathit{Receives}(s_4, s_3)\}) \not\models \mathcal{Q}$. In this case, Γ is a *contingency set* for *Receives*(s_4, s_3), and $\Gamma = \{\mathit{Receives}(s_3, s_3)\}$ is a minimum-size contingency set. It holds $D \smallsetminus \{\mathit{Receives}(s_4, s_3), \mathit{Receives}(s_3, s_3)\} \not\models \mathcal{Q}$.

We can see that the counterfactual cause *Store*(s_3) is also an actual cause, with the empty set as the minimum-size contingency set: it does not need any company to invalidate the query when removed from the database.

Causal responsibility is defined in terms of minimum-size contingency sets: If a tuple τ is an actual cause for $\mathcal{Q}$, its responsibility is $\rho(\tau) = \frac{1}{1+|\Gamma|}$, where Γ is a minimum-size contingency set for τ. The responsibility is set to 0 when a tuple is not an actual cause.

We can see then that *Store*(s_3) is an actual cause with responsibility 1; and the tuple *Receives*(s_4, s_3) is an actual cause with responsibility $\frac{1}{2}$. Similarly, *Receives*(s_3, s_3) and *Store*(s_4) are actual causes, each with responsibility $\frac{1}{2}$. □

As earlier said, the responsibility score can be assigned to variables, or values of variables, that participate in a model that contains output variables. In databases, tuples can be seen as binary variables taking values 1 or 0, indicating whether or not the tuple belongs to the database, respectively. At the same time, these variables are the input of a model that captures both the database and the query at hand. A probability-based generalization of the responsibility score has also been applied to assign (numerical) scores to values of features that characterize entities that are subject to classification by means of a trained classification model [7]. This can be done with either black-box or open models. This way, it is possible to identify the feature values that are most relevant to the outcome of the classification.

As we have seen, the contribution of tuples to query results has been based on causality-based approaches, and indirectly, on the concept of explanation. Actually, explanations have been treated in many disciplines, and, in particular, in AI, under *model-based diagnosis* [34], where *consistency-based* and *abductive* are the main approaches. Some connections between actual causality in databases and both forms of diagnosis have been established in [8, 9], respectively.

Back to databases, Salimi et al. [31] have illustrated that in some situations, responsibility may assign non-intuitive scores to tuples. As an alternative, they introduced the *causal-effect* score. This goes through creating a probabilistic database [35], seeing the query as a binary random variable, and computing the difference between the expected values of the query conditioned on the presence and the absence of the tuple under consideration. Their examples showed that the causal-effect score better captures the intuition in some cases. Yet, their effort brings up some basic questions. Is there any notion that directly considers and quantifies the intuitive notion of *contribution*? What makes the choice of a contribution score a good one? Fortunately, questions of this sort have been addressed in decades of research in the field of game theory.

Indeed, in this work, we treat the contribution from the viewpoint of game theory and tie it to the question of *how to properly distribute wealth (profit) among collaborating agents*. To this end, we appeal to widely applied and established concepts and techniques from *cooperative game theory*. There, we find as a natural candidate the popular *Shapley value*, introduced by Lloyd Shapley in 1953 [32] as a measure of the contribution of a player to the common wealth associated with a multiplayer coalition game.

In our setting, the database tuples can be seen as the players, and the query value as the numerical, joint wealth function. The query can be Boolean, taking the values 1 or 0, for true and false, respectively. This is the case for the query in (1). It could also be an aggregate numerical query that maps the database into a number.

Example 3. (ex. 1 cont.) We add a third attribute (column), amount, to the table *Receives*, indicating the amount of money received by the store in the first column from that in the second. We also add the last two tuples, obtaining a new database instance D'.

Receives$'$	receiver	sender	amount
	s_2	s_1	10
	s_3	s_3	25
	s_4	s_3	15
	s_2	s_4	18
	s_2	s_3	20

We can pose the query about the total amount received by store s_2 from official stores:

$$\mathcal{Q}_1: \ \mathit{sum}\{\!\{y \mid \exists z(\mathit{Receives}'(s_2, z, y) \land \mathit{Store}(z))\}\!\}\,. \quad (2)$$

Here, we use the bag notation $\{\!\{\cdot\}\!\}$ since we care about all of the numbers y and not just the distinct ones. Only the last two tuples of *Receives*$'$ contribute to the sum (not the first one, because s_1 is not an official store). So, the query answer is $18 + 20 = 38$. Similarly, we can pose the query about the maximum amount received by s_2:

$$\mathcal{Q}_2: \ \mathit{max}\{z \mid \exists y(\mathit{Receives}'(s_2, y, z) \land \mathit{Store}(y))\}, \quad (3)$$

with answer 20. Notice that these are aggregations over a conjunctive query.

In this work, we consider only *scalar aggregate queries*, i.e. without *group-by*. □

Using the Shapley value, we can quantify the contribution of tuples to query answers. For example, the contribution of the tuple *Store*(s_3) to the answer, 1, of the query (1), will be $\mathsf{Shapley}_{\mathcal{Q}}(D, \mathit{Store}(s_3))$, that is, the Shapley value of the tuple *Store*(s_3) of database D w.r.t. the query $\mathcal{Q}$. Similarly, we can quantify the contribution of the tuple *Receives*$'(s_2, s_4, 18)$ of D' to the query $\mathcal{Q}_1$, through $\mathsf{Shapley}_{\mathcal{Q}_1}(D', \mathit{Receives}'(s_2, s_4, 18))$.

As we will see in Section 2, the general definition of the Shapley value has a counterfactual flavor, in that interventions on the players are implicitly considered. As in actual causality in databases [27], our application of the Shapley value allows to partition the database into *endogenous* and *exogenous* tuples. Only for the former, we quantify the degree of contribution to query answering, whereas the latter are taken as given and fixed. They are beyond our control or scope of analysis. They could be, for example, tuples inherited from external sources or legacy data. The partition is application dependent. For the same reason, exogenous tuples are not subject to counterfactual interventions, in particular, to hypothetical deletions, and are always present.

Given the set of players and the "wealth function", the definition of the Shapley value of a player follows a general pattern (see Section 2). It is also well known that the Shapley value possesses properties that cast it as natural and intuitive. Actually, the Shapley value emerged as the only function that enjoys those desirable properties [33].

If we decide to adopt this approach, the main general question in our context is: *What is the computational cost of computing the Shapley value of a tuple, for a fixed, given query?* Since the query is fixed, this question is about *data complexity*, that is, the time complexity of computing the Shapley value for a tuple in terms of the size of the underlying database D. We should mention that, although we are computing the Shapley value for a single, particular tuple, all the other players (tuples) have to be taken into account.

Interestingly, the answer to the above question depends on the syntax of the query. We can provide a fairly complete picture of the complexity of the Shapley value computation when the query is conjunctive, or an aggregation over a conjunctive query. However, with a proviso: as long as the query does not contain self-joins. For example, query (1) contains a self-join since the predicate *Store* appears twice in the conjunction. It is important to notice that CQs with self-joins have turned out to be elusive when it comes to fully characterizing the complexity of dealing with them in several different data management tasks.

For CQs without self-joins, we can give a precise characterization of the complexity of computing the Shapley value. Even more, we can provide a *dichotomy result* that tells us that CQs of a certain syntactic form can be computed in polynomial time (in the size of the database D), and any other CQ is provably *hard* to compute, in a precise sense, as we will see in Section 4. This classification can be extended to some aggregations over self-join free CQs.

Example 4. (ex. 3 cont.) The purely conjunctive part of query (2), namely

$$\mathcal{Q}_1' : \exists y \exists z (\mathit{Receives}'(s_2, z, y) \wedge \mathit{Store}(z)). \qquad (4)$$

is self-join free. Just by looking at its syntactic shape, we can conclude that the Shapley value of any of the tuples in the database D' can be computed in polynomial time in the size of D'.

Now, let us add to the database a new table, *IntStore*, displaying international stores. We denote the resulting database by D''.

IntStore	istore
	s_2
	s_3
	s_5

Now, the query is about international stores in receiving relationship with official stores:

$$\mathcal{Q}_3 : \exists x \exists y \exists z (\mathit{IntStore}(x) \wedge \mathit{Receives}'(x, y, z) \wedge \mathit{Store}(y)). \quad (5)$$

This query is self-join free. Again, from the syntactic shape of this formula, we can conclude that computing the Shapley values of database tuples is computationally hard. □

Actually, the main result in this work is a dichotomy criterion that tells us that, for a self-join free CQ $\mathcal{Q}$, the following holds: If $\mathcal{Q}$ is *hierarchical* (as we formally define in Section 4), then $\mathsf{Shapley}_{\mathcal{Q}}(D, \tau)$, for $\tau \in D$, can be computed in polynomial time in the size, $|D|$, of D. Otherwise, the computation is #P-complete, which can be interpreted as an intractable complexity class. The hierarchy condition is purely syntactic and can be easily checked. This result can be extended to some of the aforementioned aggregations.

It is worth mentioning here that this dichotomy result follows the same pattern as that for answering conjunctive queries over *probabilistic databases* [15, 35]. In particular, the same hierarchy condition is used. However, the proofs for the probabilistic case cannot be used or easily adapted to our case. We take a fresh route, at least for the harder part of the proof (namely, the hardness part of the dichotomy).

Another interesting result we obtain states the existence of an efficient and good approximation algorithm for the hard cases of the computation of the Shapley value. In particular, every UCQ (and some aggregations over UCQs) has efficient approximation algorithms for arbitrary approximation ratios.

In this section, we discussed mainly Boolean CQs and unions thereof, where the answer to the query is 1 or 0, and scalar, numerical aggregations over CQs. In this way, the queries always return a number. The extension of the results presented here to *open queries* with variables, such as $\mathcal{Q}(z) : \exists y (\mathit{Receives}'(s_2, z, y) \wedge \mathit{Store}(z))$, asking now about official stores, is rather straightforward: one keeps fixed a particular answer, instantiates the query with it, obtaining a Boolean query, and then one proceeds as before. (More details are given in Section 4.)

Many more results than those presented here, and their technical details, can be found in the conference version of this article [23].

2. THE SHAPLEY VALUE OF TUPLES

Let us consider a set of players D, and a *wealth function* (or simply, *game function*) $\mathcal{G}$ that assigns real numbers to the subsets of the players, that is, $\mathcal{G} : \mathcal{P}(D) \longrightarrow \mathbb{R}$. Here, $\mathcal{P}(D)$ denotes the power set of D (i.e., the set of all subsets of D). The contribution of a particular player $p \in D$ to the common wealth of the game represented by $\mathcal{G}$ is its Shapley value, defined by:

$$\mathsf{Shapley}_{\mathcal{G}}(D, p) \overset{\text{def}}{=} \sum_{S \subseteq D \setminus \{p\}} \frac{|S|!(|D|-|S|-1)!}{|D|!} (\mathcal{G}(S \cup \{p\}) - \mathcal{G}(S)). \qquad (6)$$

An intuitive explanation of this formula is as follows. Consider the situation when we form the coalition D by selecting random players from D, one by one, randomly and uniformly (without replacement). The Shapley value of p is the expected (or average) contribution of p when it is added to the set of players selected up to that point. Put differently, when ordering the players of D in a random permutation, how does $\mathcal{G}$ differ between the prefix that precedes p and the one that ends with p? Accordingly, $|S|!(|D| - |S| - 1)!$ corresponds to the number of permutations of D with all players in S coming first, then p, and then all the others.

In our case where the players are the tuples in the database D, we view D as consisting of two parts, D_{x} and D_{n}; the former comprises the *exogenous* tuples and the latter the *endogenous* tuples. A player p of the general setting becomes an endogenous tuple $\tau \in D_{\mathsf{n}}$, and, in Formula (6), S and $S \cup \{\tau\}$ become subinstances of D_{n}. The wealth function

$\mathcal{G}(S)$ becomes $\mathcal{Q}[D_\mathsf{x} \cup S] - \mathcal{Q}[D_\mathsf{x}]$. Note that $\mathcal{Q}[D_\mathsf{x}]$ is the result of the query on the subinstance that contains all exogenous and none of the endogenous tuples. The subtraction of $\mathcal{Q}[D_\mathsf{x}]$ is due to the formal requirement of the wealth function underlying the Shapley value to be zero on the empty set of players. The Shapley value of a tuple then becomes:

$$\mathsf{Shapley}_{\mathcal{Q}}(D,\tau) \overset{\text{def}}{=\!=} \tag{7}$$
$$\sum_{S \subseteq D_\mathsf{n} \setminus \{\tau\}} \frac{|S|!(|D_\mathsf{n}| - |S| - 1)!}{|D_\mathsf{n}|!} (\mathcal{Q}[D_\mathsf{x} \cup S \cup \{\tau\}] - \mathcal{Q}[D_\mathsf{x} \cup S]) .$$

Example 5. Consider a very simple database instance D represented by the following table.

Players	name	amount
	john	5
	joe	2
τ	**sue**	**4**
	mary	5
	peggy	14

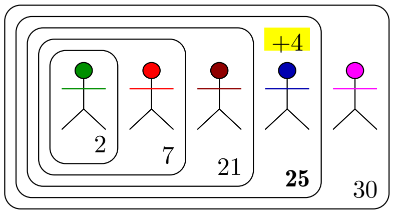
Here we assume that all tuples are endogenous, that is, $D_\mathsf{n} = D$ and $D_\mathsf{x} = \emptyset$. The query

$$\mathcal{Q}\colon\ sum\{\!\{y \mid \exists x Players(x,y)\}\!\}$$

represents the sum of numbers in the last attribute. We wish to quantify, via the Shapley value, the contribution of the selected tuple τ, namely $Players(sue, 4)$, to the answer.

The table above does not encode any particular order among the tuples. One particular subset S of $D \setminus \{\tau\}$ contains the tuples indicated by an arrow in the table below. The second to last tuple is not chosen. One possible permutation of D, and also of S, is given by the auxiliary tuple numbers on the left-hand side. It is also shown in the figure. The numbers at the lower-right corners show the total wealth of the players in the game so far. With this particular permutation, the contribution of τ, namely $(\mathcal{Q}[S \cup \{\tau\}] - \mathcal{Q}[S])$ in the sum in (7), is 4.

Players	name	amount	
#2	john	5	←
#1	joe	2	←
#4	**sue**	**4**	τ
#5	mary	5	×
#3	peggy	14	←



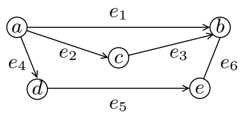
This contribution of 4 will appear in the sum as many times as the number of permutations of S, that is, 6 times. If we had not only tuple #5, but also a sixth one, say #6, outside S, we would multiply 4×6 by 2 (for the two possible permutations of the tuples left outside S). □

It is well-known that, in various instantiations of coalition games, the computation of the Shapley value for a player can be hard to compute, actually #P-hard. This means it is at least as difficult as any problem in the class #P that contains the computational problems of counting the solutions of problems in the class NP, that is, decision problems that can be solved in polynomial time by means of a non-deterministic Turing machine [3]. Among the best-known and hard problems in the class #P, we find #SAT, i.e., the problem of computing the number of satisfying truth valuations for a propositional formula. It should be clear that this problem is at least as hard as SAT, since a formula is satisfiable if and only if the number of solutions is nonzero.

3. COMPARING THE MEASURES

We now illustrate the differences among the contribution measures we mentioned in the previous sections, namely *responsibility*, *causal-effect*, and *Shapley value*, on several examples. We begin with the following example, taken from Salimi et al. [31], that was used as a motivation to introduce an alternative to the notion of causal responsibility, that of causal-effect.

Example 6. Consider a database D defined via a table *Edge* with two attributes. Each tuple in the table represents an edge of the following graph.



Here, we assume that all edges e_i are endogenous tuples. Let $\mathcal{Q}_{ab}$ be the Boolean query (definable in, e.g., Datalog, or as a UCQ) that determines whether there is a path from a to b. Let us compute the contribution of different edges e_i to the query result. Intuitively, we expect e_1 to have the highest value as it provides a direct path from a to b, while e_2 contributes to a path only in the presence of e_3, and e_4 enables a path only in the presence of both e_5 and e_6.

All tuples in D are actual causes since every tuple appears in a path from a to b. It is easy to verify that all the tuples in D have the same causal responsibility, $\frac{1}{3}$, which may be considered as counter-intuitive. In that sense, the causal-effect and Shapley value return more intuitive results, as we have that [31]:

$$\mathsf{CE}_{\mathcal{Q}_{ab}}(D,e_1) = 0.65625$$
$$\mathsf{CE}_{\mathcal{Q}_{ab}}(D,e_i) = 0.21875,\ \text{for } i \in \{2,3\}$$
$$\mathsf{CE}_{\mathcal{Q}_{ab}}(D,e_i) = 0.09375,\ \text{for } i \in \{4,5,6\}$$

where we use the notation $\mathsf{CE}_{\mathcal{Q}_{ab}}(D,e_i)$ for the causal effect of $e_i \in D$ w.r.t. $\mathcal{Q}_{ab}$, and:

$$\mathsf{Shapley}_{\mathcal{Q}_{ab}}(D,e_1) = 0.5833$$
$$\mathsf{Shapley}_{\mathcal{Q}_{ab}}(D,e_i) = 0.1333,\ \text{for } i \in \{2,3\}$$
$$\mathsf{Shapley}_{\mathcal{Q}_{ab}}(D,e_i) = 0.05,\ \text{for } i \in \{4,5,6\}$$

(The detailed computations can be found in [23].) □

Note that the responsibility measure is fundamentally designed for non-numerical queries, and it is not at all clear whether it can incorporate the numerical contribution of a tuple (e.g., recognizing that some tuples contribute more than others due to high numerical attributes). Therefore, in the following example, we only consider the causal-effect measure and the Shapley value.

Example 7. Consider again the query $\mathcal{Q}_3$ in (3), and assume that all the tuples in the table *Receives'* in Example 3 are exogenous, while the tuples in the table *Store* are endogenous. It is rather straightforward to see that:

$$\mathsf{CE}_{\mathcal{Q}_3}(D, \mathit{Store}(s_2)) = \mathsf{Shapley}_{\mathcal{Q}_3}(D, \mathit{Store}(s_2)) = 0$$

as this tuple has not impact on the query result (that is, the addition of this tuple to any subset of the endogenous tuples does not change the query result).

For the other two tuples of *Store*, a simple computation shows that:

$$\mathsf{CE}_{\mathcal{Q}_3}(D, \mathit{Store}(s_3)) = \mathsf{Shapley}_{\mathcal{Q}_3}(D, \mathit{Store}(s_3)) = 11$$
$$\mathsf{CE}_{\mathcal{Q}_3}(D, \mathit{Store}(s_4)) = \mathsf{Shapley}_{\mathcal{Q}_3}(D, \mathit{Store}(s_4)) = 9. \qquad \square$$

These two examples show that the numbers for the causal-effect measure and the Shapley value may coincide in some cases, and be different in other cases. In both examples, however, the two measures rank the tuples similarly according to their contribution to the query result.

We showed in [23] that the causal-effect score coincides with another popular score used in cooperative game theory and related areas, namely the *Banzhaf Power Index* [16]. Its definition is similar to that of the Shapley value, but instead of considering permutations of subsets of players, only subsets of players are considered, which has advantages in some applications. In general, its computation is also intractable [17]. It is known that the Shapley value and Banzhaf power index may produce different rankings in general [30].

While the justification to measuring tuple contribution using one measure over the other is yet to be established, we believe that the suitability of the Shapley value is backed by the aforementioned theoretical justification as well as its massive adoption in a plethora of fields.

4. COMPUTATIONAL COMPLEXITY

We now discuss the computational complexity of calculating the Shapley value of a database tuple with respect to a database query, either exactly or approximately.

4.1 Conjunctive Queries

We first discuss the case of *Boolean* CQs, that is, CQs such as $\mathcal{Q}$ of Equation (1) where all variables are existentially quantified. For the fragment of CQs without self-joins (i.e., no relation name is mentioned more than once), we can classify all the queries into tractable (polynomial-time) and intractable (#P-hard) ones. Interestingly, the tractability criterion is the same as that of query answering over tuple-independent probabilistic databases [13]: being *hierarchical*. (We recall the precise definition later on.)

Theorem 4.1. *Let $\mathcal{Q}$ be a Boolean CQ without self-joins. If $\mathcal{Q}$ is hierarchical, then $\mathsf{Shapley}_{\mathcal{Q}}(D, \tau)$ can be computed in polynomial time, given D and τ. Otherwise, the problem is #P-hard.*

Recall that #P is the complexity class of problems that can be defined as that of counting the witnesses of a problem in NP. A Boolean CQ $\mathcal{Q}$ is *hierarchical* if for every two variables x and y it holds that the set of atoms (conjuncts) that contain x either contains, is contained in, or is disjoint from the set of atoms that contain y. For example, every CQ with at most two atoms (e.g., the query $\mathcal{Q}'_1$ in (4)) is hierarchical. The query $\mathcal{Q}_3$ of Example 4, on the other hand, is not hierarchical, since the sets of atoms that contain the variables x and y are $\{\mathit{IntStore}(x), \mathit{Receives}'(x, y, z)\}$ and $\{\mathit{Receives}'(x, y, z), \mathit{Store}(y)\}$, respectively, which have neither disjointness nor containment between them.

For illustration, we conclude from Theorem 4.1 that the Shapley value is tractable for $\mathcal{Q}'_1$ of Equation (4), but intractable for $\mathcal{Q}_3$ of Equation (5). On the other hand, the theorem does not say anything about the complexity of $\mathcal{Q}$ in Equation (1), since $\mathcal{Q}$ has a self-join (as it contains two occurrences of the *Store* relation). Nevertheless, an easy reduction from the case of $\mathcal{Q}_3$ shows that the Shapley value is #P-hard to compute for $\mathcal{Q}$ as well (see, e.g., [29]).

The algorithm for hierarchical Boolean CQs $\mathcal{Q}$ without self-joins is a reduction to the following counting problem: given a database D and a number k, how many sets of k tuples from D satisfy $\mathcal{Q}$? (The problem of counting the subsets of D that satisfy $\mathcal{Q}$, regardless of their size, has been the subject of recent studies [1, 22].) In [23] we show that when $\mathcal{Q}$ is a hierarchical CQ without self-joins, this problem is solvable in polynomial time.

Similarly to Dalvi and Suciu [14], our proof of hardness in [23] consists of two steps. First, we prove hardness for the simplest non-hierarchical query

$$\mathcal{Q}_{\mathsf{RST}}\colon \exists x \exists y (R(x) \land S(x, y) \land T(y))\,.$$

Then, we reduce the computation of $\mathsf{Shapley}_{\mathcal{Q}_{\mathsf{RST}}}(D, \tau)$ to that of $\mathsf{Shapley}_{\mathcal{Q}}(D, \tau)$ for any non-hierarchical CQ $\mathcal{Q}$ without self-joins. The second step is the same as that of Dalvi and Suciu [14]. The proof of the first step is by a reduction from the problem of computing the number of independent sets of a bipartite graph. Our reduction adopts a technique that Aziz and Keijzer [4] used for proving the hardness of computing the Shapley value for a *matching game* on unweighted graphs: solve several instances of the problem in order to construct a full-rank set of linear equations.

To generalize our complexity results to non-Boolean CQs, we apply the aforementioned standard approach of converting the CQ into a Boolean CQ by referring to each output variable as a constant. The idea is that we view every answer as a separate Boolean CQ. (Recall that we are using data complexity.) Hence, for a CQ $\mathcal{Q}$ we consider the Boolean version $\mathcal{Q}_b$ where each free (output) variable becomes a constant. When $\mathcal{Q}_b$ is a Boolean CQ without self-joins, the complexity of calculating the Shapley value of a tuple τ for any answer to $\mathcal{Q}$ is the same as that of calculating $\mathsf{Shapley}_{\mathcal{Q}_b}(D, \tau)$. In particular, we can compute it in polynomial time if $\mathcal{Q}_b$ is hierarchical, and it is #P-hard otherwise. For example, in the following variation of $\mathcal{Q}_3$ in (5), we can efficiently compute the Shapley value of every tuple to every query answer.

$$\mathcal{Q}'(x)\colon\; \exists y \exists z (\mathit{IntStore}(x) \land \mathit{Receives}'(x, y, z) \land \mathit{Store}(y))$$

This is because the Boolean CQ

$$\mathcal{Q}'_b\colon\; \exists y \exists z (\mathit{IntStore}(s_2) \land \mathit{Receives}'(s_2, y, z) \land \mathit{Store}(y))\,.$$

is a hierarchical CQ.

4.2 Aggregate Queries

Next, we consider aggregate queries. For simplicity of presentation, the aggregate queries that we consider here have the form $\mathcal{Q}(D) = \alpha\{\!\{x \mid \mathcal{Q}'(D)\}\!\}$ when applied to a database D. Here, $\mathcal{Q}'$ is a CQ, x is one of the free variables of $\mathcal{Q}'$ that we view as a numerical attribute, and α is an *aggregate* *operator* that maps a bag of numbers into a single number. Recall that $\{\!\{\cdot\}\!\}$ is used here for *bag* notation. Examples of α include the functions *sum*, *min*, *max*, *average* and so on. We call $\mathcal{Q}'$ the *underlying CQ* of $\mathcal{Q}$.

Remark 1. *In the conference version of this article [23], we consider a more general class of queries where, instead of the variable x, we allow for any* feature function *that transforms a given tuple into a number (e.g., the product of two attributes). The results we state here hold for this generalized model as well.*

As expected, the complexity of an aggregate query depends on the complexity of its underlying CQ. The following theorem generalizes the hardness side of Theorem 4.1 and states that it is #P-hard to compute $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)$ whenever the underlying CQ is a non-hierarchical CQ without self-joins. The only exception is the case where $\mathcal{Q}$ is a *constant* query, that is, $\mathcal{Q}(D)=\mathcal{Q}(D')$ for all databases D and D'—in that case, $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)=0$ always holds.

Theorem 4.2. *Let $\mathcal{Q}$ be a fixed aggregate query where the underlying CQ is a non-hierarchical CQ without self-joins. If $\mathcal{Q}$ is not constant, then computing $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)$, given D and τ, is #P-hard.*

For instance, it follows from Theorem 4.2 that, computing $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)$ for the query

$$\mathcal{Q}_4\colon\ \alpha\{\!\{z \mid (\mathit{IntStore}(x)\wedge \mathit{Receives}'(x,y,z)\wedge \mathit{Store}(y))\}\!\} \quad (8)$$

is hard for all $\alpha\in\{\mathit{min},\mathit{max},\mathit{sum},\mathit{average}\}$, and, in fact, for any aggregate function α that is not a constant.

What about the other direction? Does the tractability of the Shapley value for the underlying CQ imply the tractability of the Shapley value for the whole aggregate query? We do not have any result that is as general as that of Theorem 4.2, but we can show that this is the case for the aggregate operator *sum* (and *count* as a special case). This is a simple corollary of Theorem 4.1 that we obtain by applying the linearity of expectation.

Corollary 4.3. *Let $\mathcal{Q}(D)=\mathit{sum}\{\!\{x \mid \mathcal{Q}'(D)\}\!\}$. If $\mathcal{Q}'$ is a hierarchical CQ without self-joins, then $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)$ can be computed in polynomial time, given D and τ.*

As an example, the Shapley value can be computed in polynomial time for the query $\mathcal{Q}_1$ of Example 3, because the underlying CQ has only two atoms, and is then hierarchical.

The complexity of computing the Shapley value for other aggregate queries remains an open problem for the general case where the underlying CQ is a hierarchical CQ without self-joins. We can, however, state a positive result for *max* and *min*, for the special case where the underlying CQ consists of a single atom (i.e., aggregation over a single relation). As an example, computing $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)$ can be done in polynomial time for:

$$\mathcal{Q}\colon\ \{\!\{\mathit{max}\{z \mid \exists y(\mathit{Receives}'(s_2,y,z))\}\!\}$$

However, we do not know whether this is also the case for the query $\mathcal{Q}_2$ of Example 3.

4.3 Approximation

The complexity results presented so far imply that computing the exact Shapley value is often intractable. Nevertheless, the picture is far more optimistic when allowing approximation with strong precision guarantees. A conventional feasibility notion of arbitrarily-tight approximations is via efficient approximation *schemes* such as the *Fully Polynomial-Time Approximation Scheme* (*FPRAS* for short). An FPRAS for a numeric function f is a randomized algorithm $A(x,\epsilon,\delta)$, where x is an input for f and $\epsilon,\delta\in(0,1)$, that returns an ϵ-approximation of $f(x)$ with probability $1-\delta$ (where the probability is over the randomness of A) in time polynomial in x, $1/\epsilon$ and $\log(1/\delta)$. To be more precise, we distinguish between an *additive* (or *absolute*) FPRAS:

$$\Pr\left[f(x)-\epsilon\le A(x,\epsilon,\delta)\le f(x)+\epsilon\right]\ \ge\ 1-\delta$$

and a *multiplicative* (or *relative*) FPRAS:

$$\Pr\left[\frac{f(x)}{1+\epsilon}\le A(x,\epsilon,\delta)\le(1+\epsilon)f(x)\right]\ \ge\ 1-\delta\,.$$

Using the Chernoff-Hoeffding bound, we easily get an additive FPRAS of $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)$ when $\mathcal{Q}$ is *any* monotone Boolean query computable in polynomial time, by simply taking the ratio of successes over $O(\log(1/\delta)/\epsilon^2)$ trials of the following experiment:

1. Select a random permutation $(t_1,\dots,t_n)$ over the set of endogenous tuples in the database,
2. Suppose that $\tau=t_i$, and let $D_{i-1}=D_{\mathsf{x}}\cup\{t_1,\dots,t_{i-1}\}$. If $\mathcal{Q}(D_{i-1})$ is false and $\mathcal{Q}(D_{i-1}\cup\{\tau\})$ is true, then report "success;" otherwise, "failure."

Moreover, the additive FPRAS extends to non-Boolean CQs in the same manner described in Section 4.1.

In general, the existence of an additive FPRAS for a function f does not guarantee the existence of a multiplicative one, since $f(x)$ can be very small. For example, we can get an additive FPRAS of the satisfaction of a propositional formula over Boolean i.i.d. variables by, again, sampling the averaging, but there is no multiplicative FPRAS for such formulas unless NP $\subseteq$ BPP. Nevertheless, the situation is different for $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)$ when $\mathcal{Q}$ is a CQ, and even a UCQ, since the Shapley value is *never* too small. In particular, we have the following "gap" property.

Proposition 4.4. *Let $\mathcal{Q}$ be a fixed Boolean UCQ. Then $\mathcal{Q}$ satisfies the* gap property*: there is a polynomial p such that for all databases D and tuples τ of D it is the case that $\mathsf{Shapley}_{\mathcal{Q}}(D,\tau)$ is either zero or at least $1/(p(|D|))$.*

It follows from Proposition 4.4 that a multiplicative FPRAS can be obtained using the above sampling algorithm, possibly with a different (yet still polynomial) number of samples. Hence, we have the following.

Corollary 4.5. *For every fixed UCQ, the Shapley value has both an additive and a multiplicative FPRAS.*

Observe that Corollary 4.5 does not make any assumption about self-joins—it allows for any UCQ. As we explain shortly, it ceases to hold once *negation* is allowed. Corollary 4.5 also generalizes to a multiplicative FPRAS for summation over CQs, in the case where all the values in the summation have the same sign (i.e., they are either all negative or all non-negative). In other words, there is a multiplicative FPRAS for the query $\mathcal{Q}(D)=\mathit{sum}\{\!\{x \mid \mathcal{Q}'(D)\}\!\}$ under the assumption that all of the values that x is assigned have the same sign.

What changes when we allow for *negation*? This problem has been studied by Reshef et al. [29]. On the positive side, the additive approximation is tractable even when UCQs are allowed to use (safe) negation.

Proposition 4.6. ([29]) *For every fixed UCQ with negation, the Shapley value has an additive FPRAS.*

Yet, the "gap" property can be violated when CQs are allowed to use negation. Reshef et al. [29] have shown that we lose this property when considering CQs with negated atoms. For example, when considering the query

$$\exists x \exists y (\mathit{Store}(x) \wedge \mathit{Receives}(x,y) \wedge \neg \mathit{Store}(y))$$

that differs from the query $\mathcal{Q}$ of Example 1 only by the negation of the last atom, the Shapley value of a tuple may be as small as $2^{-\Theta(|D|)}$. In fact, a CQ with negation almost always violates the gap property [29, Theorem 5.1].

While the gap property is a technique for extending additive approximations into multiplicative ones, its violation does not preclude the existence of *any* multiplicative approximation. Nevertheless, Reshef et al. [29] have shown that there are CQs with negation where a multiplicative approximation is infeasible, since it is already NP-hard to determine whether the Shapley value is nonzero. An example is the following CQ:

$$\exists x \exists y \exists z \exists w (T(z) \wedge \neg R(x) \wedge \neg R(y) \wedge R(z) \wedge R(w) \\ \wedge S(x,y,z,w)).$$

5. CONCLUSIONS

Explanations in AI, Machine Learning, in particular, and Data and Knowledge Management have become prominent and active areas of research. There are different notions about, and approaches to, explanations in those fields (and more generally and traditionally, in Science and Philosophy). There is still much debate about what is an explanation, and even more, about what is a *good explanation*.

Many approaches in the broad areas we just mentioned are based on some sort of causal analysis, and more specifically, on counterfactual analysis. We could say that the Shapley value has a rather implicit counterfactual component, through the average of the game outcome depending on the presence or not of a particular player, under different scenarios related to the other players.

Explanations are usually, but not always, expressed as numerical scores that represent the importance of a player, a database tuple, a feature value (c.f. below), etc., for the outcome of a function, a database, or a computational model. In this work, we have adopted this approach to the explanations for query answers from relational databases.

In Section 1, we mentioned the *causal-effect* [31] as an explanation score for database tuples and query answering, as an alternative to *responsibility*. All of our complexity results for the exact computation of the Shapley value are also applicable to the causal-effect measure (and Banzhaf power index). However, we can show that the "gap" property does not hold for this measure, and the question of whether there exists a multiplicative approximation remains open for future investigation.

The Shapley value has been used in a similar yet different manner for another fundamental task in data management: measuring the responsibility of a tuple to the level of *inconsistency* [28, 36]. For such a measure to be applicable, one needs first to adopt a measure for quantifying the amount of inconsistency of the database. More concretely, such a measure should indicate the extent to which the database's integrity constraints are violated. In a more recent work [24], we have studied the complexity of the Shapley value of tuples under various natural measures of inconsistency: the number of violations, the number of tuples that participate in violations, the number of repairs, and the minimal number of tuples that one needs to delete to retain consistency [5].

It is worth mentioning, for a broader view of things, that the Shapley value has been used lately in machine learning, to quantify the relevance of a feature value for the outcome of a model (e.g., the result of a classifier), which is important in *Explainable AI* (XAI). In this area, it is known as the *SHAP-score* [25]; and has been applied both with black-box and open-box models. Only the input/output relation is needed for its computation. However, the availability of the model may make the computation much more efficient [2,37], actually tractable in some cases. In [7], the SHAP-score was experimentally compared with other scores, such as, *RESP*, an adaptation of the responsibility score to the classification setting, and the Rudin-score for FICO data [11].

There are many directions in which our and related work could be extended. One that looks particularly promising and relevant has to do with the definition or computation of explanation scores in combination with domain knowledge or, *semantics*, in more general terms [6]. In particular, this additional knowledge could inform the scores about the explanations, such as database tuples, feature values, etc., that become useful, or more technically, *actionable* or *algorithmic recourses* [21], i.e. something we can do something with. For example, if a loan application is rejected due to blurry personal payment history, we might be in a position to clarify our records. However, if it is due to our low educational level, it might be too late (or impossible) to do anything about it.

Acknowledgments

The work of L. Bertossi was funded by ANID - Millennium Science Initiative Program- Code ICN17002. The work of Ester Livshits, Benny Kimelfeld, and Moshe Sebag was supported by the Israel Science Foundation (ISF), grants 1295/15 and 768/19, and the Deutsche Forschungsgemeinschaft (DFG) project 412400621 (DIP program). The work of Ester Livshits was also supported by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel Cyber Bureau.

6. REFERENCES

[1] A. Amarilli and B. Kimelfeld. Model counting for conjunctive queries without self-joins. *CoRR*, abs/1908.07093, 2019. To appear at ICDT 2021.

[2] M. Arenas, P. Barceló, L. Bertossi, and M. Monet. The tractability of SHAP-scores over deterministic and decomposable boolean circuits. In *Proceedings of AAAI*, 2021. CoRR abs/2007.14045.

[3] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[4] H. Aziz and B. de Keijzer. Shapley meets Shapley. In *STACS*, pages 99–111, 2014.

[5] L. Bertossi. Repair-based degrees of database inconsistency. In *LPNMR*, volume 11481 of *LMCS*, pages 195–209. Springer, 2019.

[6] L. Bertossi. Declarative approaches to counterfactual explanations for classification. *CoRR*, abs/2011.07423, 2020. Extended version of RuleML+RR'20 paper.

[7] L. Bertossi, J. Li, M. Schleich, D. Suciu, and Z. Vagena. Causality-based explanation of classification outcomes. In *DEEM@SIGMOD*, pages 6:1–6:10. ACM, 2020.

[8] L. Bertossi and B. Salimi. Causes for query answers from databases: Datalog abduction, view-updates, and integrity constraints. *Int. J. Approx. Reason.*, 90:226–252, 2017.

[9] L. Bertossi and B. Salimi. From causes for database queries to repairs and model-based diagnosis and back. *Theory Comput. Syst.*, 61(1):191–232, 2017.

[10] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2001.

[11] C. Chen, K. Lin, C. Rudin, Y. Shaposhnik, S. Wang, and T. Wang. An interpretable model with globally consistent explanations for credit risk. *CoRR*, abs/1811.12615, 2018.

[12] H. Chockler and J. Y. Halpern. Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res.*, 22:93–115, 2004.

[13] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[14] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875. Morgan Kaufmann, 2004.

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

[16] P. Dubey and L. S. Shapley. Mathematical properties of the Banzhaf power index. *Mathematics of Operations Research*, 4(2):99–131, 1979.

[17] G. Greco, F. Lupia, and F. Scarcello. Structural tractability of Shapley and Banzhaf values in allocation games. In *IJCAI*, pages 547–553, 2015.

[18] T. J. Green and V. Tannen. The semiring framework for database provenance. In *PODS*, pages 93–99. ACM, 2017.

[19] J. Y. Halpern. A modification of the Halpern-Pearl definition of causality. In *IJCAI*, pages 3022–3033. AAAI Press, 2015.

[20] J. Y. Halpern and J. Pearl. Causes and Explanations: A Structural-Model Approach. Part I: Causes. *The British Journal for the Philosophy of Science*, 56(4):843–887, 2005.

[21] A. Karimi, B. J. von Kügelgen, B. Schölkopf, and I. Valera. Algorithmic recourse under imperfect causal knowledge: a probabilistic approach. In *NeurIPS*, 2020.

[22] B. Kenig and D. Suciu. A dichotomy for the generalized model counting problem for unions of conjunctive queries. *CoRR*, abs/2008.00896, 2020.

[23] E. Livshits, L. Bertossi, B. Kimelfeld, and M. Sebag. The shapley value of tuples in query answering. In *ICDT*, volume 155 of *LIPIcs*, pages 20:1–20:19, 2020.

[24] E. Livshits and B. Kimelfeld. The shapley value of inconsistency measures for functional dependencies. *CoRR*, abs/2009.13819, 2020. To appear at ICDT 2021.

[25] S. M. Lundberg, G. Erion, H. Chen, A. D. Grave, J. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67, 2020.

[26] A. Meliou, W. Gatterbauer, J. Y. Halpern, C. Koch, K. F. Moore, and D. Suciu. Causality in databases. *IEEE Data Eng. Bull.*, 33(3):59–67, 2010.

[27] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *Proc. VLDB Endow.*, 4(1):34–45, 2010.

[28] K. Mu, W. Liu, and Z. Jin. Measuring the blame of each formula for inconsistent prioritized knowledge bases. *Journal of Logic and Computation*, 22(3):481–516, 02 2011.

[29] A. Reshef, B. Kimelfeld, and E. Livshits. The impact of negation on the complexity of the shapley value in conjunctive queries. In *PODS*, pages 285–297. ACM, 2020.

[30] D. G. Saari and K. K. Sieberg. Some surprising properties of power indices. *Games Econ. Behav.*, 36(2):241–263, 2001.

[31] B. Salimi, L. Bertossi, D. Suciu, and G. V. den Broeck. Quantifying causal effects on query answering in databases. In *TaPP*. USENIX Association, 2016.

[32] L. S. Shapley. *A Value for n-Person Games*. RAND Corporation, Santa Monica, CA, 1952.

[33] L. S. Shapley and A. E. Roth. *The Shapley value : essays in honor of Lloyd S. Shapley*. Cambridge, 1988.

[34] P. Struss. Model-based problem solving. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 395–465. Elsevier, 2008.

[35] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[36] M. Thimm. Measuring inconsistency in probabilistic knowledge bases. In *UAI*, pages 530–537, 2009.

[37] G. Van den Broeck, A. Lykov, M. Schleich, and D. Suciu. On the tractability of SHAP explanations. In *Proceedings of AAAI*, 2021. CoRR abs/2009.08634.