


The Complexity of the Shapley Value for Regular Path Queries

Majd Khalil ✉

Technion, Haifa, Israel

Benny Kimelfeld ✉ 

Technion, Haifa, Israel

Abstract

A path query extracts vertex tuples from a labeled graph, based on the words that are formed by the paths connecting the vertices. We study the computational complexity of measuring the contribution of edges and vertices to an answer to a path query, focusing on the class of conjunctive regular path queries. To measure this contribution, we adopt the traditional Shapley value from cooperative game theory. This value has been recently proposed and studied in the context of relational database queries and has uses in a plethora of other domains.

We first study the contribution of edges and show that the exact Shapley value is almost always hard to compute. Specifically, it is $\#P$ -hard to calculate the contribution of an edge whenever at least one (non-redundant) conjunct allows for a word of length three or more. In the case of regular path queries (i.e., no conjunction), the problem is tractable if the query has only words of length at most two; hence, this property fully characterizes the tractability of the problem. On the other hand, if we allow for an approximation error, then it is straightforward to obtain an efficient scheme (FPRAS) for an additive approximation. Yet, a multiplicative approximation is harder to obtain. We establish that in the case of conjunctive regular path queries, a multiplicative approximation of the Shapley value of an edge can be computed in polynomial time if and only if all query atoms are finite languages (assuming non-redundancy and conventional complexity limitations). We also study the analogous situation where we wish to determine the contribution of a vertex, rather than an edge, and establish complexity results of similar nature.

2012 ACM Subject Classification Theory of computation → Data provenance

Keywords and phrases Path queries, regular path queries, graph databases, Shapley value

Digital Object Identifier 10.4230/LIPIcs.ICDT.2023.11

Related Version *Full Version*: <https://arxiv.org/abs/1412.2221> [15]

Funding This work was supported by the Israel Science Foundation (ISF), Grant 768/19, and the German Research Foundation (DFG) Project 412400621 (DIP program).

1 Introduction

Graph databases arise in common applications where the underlying data is a network of entities, especially when connectivity and path structures are of importance. Such usage spans many fields, including the Semantic Web [2], social networks [10], biological networks [20, 38], data provenance [1], fraud detection [30], recommendation engines [37], and many more. In its simplest form, a graph database is a finite, directed, edge-labeled graph. Vertices represent entities and edges represent binary relationships of different types (labels) between entities. Query mechanisms for graph databases enable the retrieval of parts of the graph according to patterns of connections between vertices.

A canonical example of a graph query is the Regular Path Query (RPQ) [5, 7, 8, 36]. An RPQ qualifies paths using a regular expression over the edge labels. When evaluated on a graph, the answers are source-target pairs of vertices that are connected by a path that conforms to the regular expression. This allows users to inspect complex connections in



© Majd Khalil and Benny Kimelfeld;
licensed under Creative Commons License CC-BY 4.0
26th International Conference on Database Theory (ICDT 2023).

Editors: Floris Geerts and Brecht Vandevoort; Article No. 11; pp. 11:1–11:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

graphs by enabling them to form queries that match arbitrarily long paths. An important generalization of the class of RPQs is the class of *Conjunctive Regular Path Queries* (CRPQs) that extend regular path queries to conjunctions of atoms, each being an RPQ that should hold between two specified variables [7, 8].

Being simple and expressive, RPQs and CRPQs are an integral part of popular graph query languages for graphs, such as GraphLog, Cypher, XPath, and SPARQL. Therefore, they motivate and give rise to much research effort, including the study of some natural computational problems and variations thereof [11, 22, 23, 26]: What is the complexity of deciding whether an RPQ matches a path from a given vertex to another (what we refer to as *Boolean query evaluation*)? Can we efficiently count and enumerate these paths? Is a given CRPQ contained in another given CRPQ? The combined complexity of Boolean query evaluation is in polynomial time for RPQs and NP-complete for CRPQs [4]. Data complexity, however, is NLOGSPACE-complete for both [3]. The containment problem for RPQs is PSPACE-complete, and for CRPQs, it is EXPSPACE-hard [6, 11].

In this paper, we focus on the problem of *quantifying the responsibility and contribution* of different components in the graph, namely edges and vertices, to an answer to the CRPQ (and RPQ in particular). This problem has been studied in the context of queries on relational databases, and our motivation here is the same as in the relational context: we wish to provide the database user with an explanation of *why* (or what in the database led to that) we got a specific answer; when many combinations of data items can lead to an answer, and the lineage is too large or complex, we wish to quantify the contribution of individual items in order to distinguish the more important from the less important to the answer [9].

How does one quantify the contribution of a database item to a query answer? In the relational model, several definitions and frameworks have been proposed for measuring the contribution of a tuple. For example, Meliou et al. [25] defined the responsibility of a tuple t as, roughly, the inverse of the minimal number of tuples needed to be removed in order to make t counterfactual (i.e., the query answer is determined by the existence of t); this measure is an adaptation of earlier notions of formal causality by Halpern and Pearl [13]. *Causal effect* is another measure proposed by Salimi et al. [31]: if the database is probabilistic and each tuple has independently the probability 1/2 of existence, how does the probability of the answer change if we assume the existence or absence of t ? Lastly, and most relevant to our work, recent work has studied the adoption of the *Shapley value* as a responsibility measure [9, 16, 18, 29].

The Shapley value is a formula for wealth distribution in a cooperative game [32]. In databases, the conceptual application is straightforward: the tuples are the players who play the game of answering the Boolean (or numerical) query; hence, the wealth function is the result of the query [16]. The Shapley value has a plethora of applications, including profit sharing between ISPs [21], influence measurement in social network analysis [28], determining the most important genes for specific body functions [27], and identifying key players in terrorist networks [34], to name a few. Closer to databases is a recent application to model checking for measuring the influence of formula components [24]. As another example, in machine learning, the SHAP score [19] has been used for measuring the contribution of each feature to the prediction, and it is essentially the Shapley value with the features as players. This value was also used for quantifying the responsibility that every tuple has on the *inconsistency* of a knowledge base [14, 39] and a database [18]. The Shapley value is often intractable to calculate, and particularly, the execution cost might grow exponentially with the number of players. Hence, past research has been investigating islands of tractability and approximation algorithms.

Contribution. We study the complexity of computing the Shapley value of edges and vertices for CRPQs over graph databases. In the remainder of this section and throughout the paper, we focus on edges (and discuss vertices in Section 6). Computing the Shapley value of an edge e then boils down to answering the following question. If we eliminate all edges and add them back one by one in uniformly random order, what is the probability that e is a *counterfactual cause* (i.e., its inclusion is necessary and sufficient [25]) for the answer at hand? As done in previous work in the context of relational databases [18, 25], we view the graph as consisting of two types of edges: *endogenous* edges and *exogenous* edges. The endogenous edges are the ones that we consider for reasoning on responsibility, and they are the players of the game. The exogenous edges constitute external knowledge that we take for granted, and so, they are not players in the cooperative game (and not eliminated at the beginning of the probabilistic process).

To be more precise, an instance of our problem involves a query q (e.g., an RPQ or a CRPQ), an input graph G , an answer tuple \vec{t} of vertices of G , and an edge e whose contribution to \vec{t} we seek to measure. We adopt the yardstick of *data complexity* [35] where we consider the query q as fixed. Hence, each fixed query q is associated with a distinct computational problem that takes as input G , \vec{t} , and e .

We first show that the exact computation of the Shapley value is almost always hard. Specifically, it is sufficient for the CRPQ to have a non-redundant atom (i.e., a conjunct associated with a regular language) with a word of length three or more for the computation to be #P-hard (FP^{#P}-complete). In addition, for RPQs (i.e., single-atom CRPQs), we complete this hardness condition to a full dichotomy by showing that the Shapley value can be computed in polynomial time if the language contains only words of length at most two.

Next, we study the complexity of approximation. In our context, we adopt a standard yardstick of tractable approximation, namely FPRAS (Fully Polynomial-Time Approximation Scheme). An approximation of the Shapley value of an edge to a CRPQ can be computed via a straightforward Monte Carlo (average-over-samples) estimation of the probability that we previously defined. This estimation guarantees an additive (or absolute) approximation. However, we are also interested in a multiplicative (or relative) approximation.

We establish a dichotomy that classifies CRPQs into a class where there is a multiplicative FPRAS and the complementing class where there cannot be any such FPRAS under conventional complexity assumptions. Specifically, if the CRPQ contains an atom (non-redundant atom) with an infinite regular language, then (any) multiplicative approximation is intractable since it is already NP-complete to determine whether the Shapley value is nonzero. In every other case (assuming no redundant atoms), an additive FPRAS can also be used to obtain a multiplicative FPRAS, due to the *gap property*, previously established in the relational model [18, 29]: if the Shapley value is nonzero, then it is at least the reciprocal of a polynomial. Note that this is contrasting the situation with relational conjunctive queries, where there is always a multiplicative FPRAS [16]. Intuitively, this is true since, unlike the case of conjunctive queries, in the case of RPQs and CRPQs we do not necessarily have any fixed upper bound on the minimal number of tuples (edges) that need to be present for e to become a counterfactual cause.

Moving on from edges to vertices, the complexity situation remains quite similar. In particular, it is generally hard to compute the exact Shapley value of a vertex: it is sufficient for the CRPQ to have a non-redundant atom that contains a word of length four or more for the computation to be hard. For RPQs, we establish that the family of tractable queries for edges is also tractable for vertices. Yet, for vertices, a gap remains and we do not complete a full classification. For approximate evaluation, we establish the same dichotomy as for edges.

Organization. The rest of the paper is organized as follows. We introduce some basic terminology in Section 2. In Section 3, we formally define how the Shapley value is applied in our setting for edges in graph databases. We study the complexity of computing exact Shapley values for CRPQs in Section 4, and investigate approximations in Section 5. In Section 6, we present the complexity results for the case when measuring the contribution of vertices instead of edges. We conclude and discuss directions for future work in Section 7. For lack of space, some of the proofs are omitted and can be found in the full version of the paper [15].

2 Preliminaries

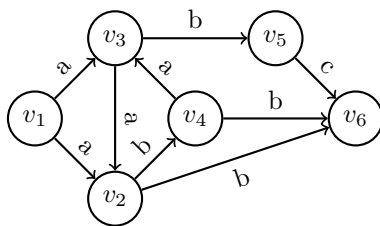
We begin by setting some terminology and notation that we use throughout the paper.

Graphs and Path Queries

We use Σ to denote a finite alphabet (i.e., set of symbols) that is used for labeling edges of graphs. A *word* is a finite sequence of symbols from Σ . As usual, Σ^* denotes the set of all words. A *language* L is a (finite or infinite) subset of Σ^* . By a slight abuse of notation, we may identify a language L with a representation of L such as a regular expression or a finite-state automaton. A *regular expression* is defined as follows: \emptyset , ϵ , and $\sigma \in \Sigma$ represent the empty language, the empty word, and the symbol σ , respectively; and if r and s are regular expressions, then $(r \mid s)$ and $(r \cdot s)$ and (r^*) are also regular expressions, denoting union, concatenation, and Kleene star, respectively. We sometimes omit parentheses and dots when there is no risk of ambiguity (so we may write rs instead of $(r \cdot s)$, for instance). The language $L(r)$ that r accepts (recognizes) is defined as usual. We abbreviate by Σ^* the regular expression that accepts every word. A *deterministic finite automaton* (DFA) A is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta: Q \times \Sigma \rightarrow Q$ is the transition function, q_0 is the initial state, F is the set of accepting states. By $\delta^*(w)$ we denote the state that the automaton reaches after reading w , starting from the initial state. The automaton *accepts* a word w if $\delta^*(w) \in F$. We again use $L(A)$ to denote the language that A recognizes. (Recall that the classes of regular expressions and DFAs coincide in their expressive power.)

By a *graph* we mean an edge-labeled directed graph $G = (V, E)$ where V is the finite set of vertices and $E \subseteq V \times V$ is the set of edges (v, u) , each with a label $lbl(v, u)$. We will consistently denote by n and m the number of vertices and edges, respectively; that is, $n = |V|$ and $m = |E|$. A path p from the vertex u to the vertex v in G is a sequence $p = (v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ of edges in G such that $u = v_0$ and $v = v_k$. By $|p|$ we denote the length k of p , and by $lbl(p)$ we denote the word $lbl(v_0, v_1) \cdots lbl(v_{k-1}, v_k)$. If $G = (V, E)$ is a graph and $E' \subseteq E$ is a set of edges, then we denote by $G[E']$ the subgraph $G' = (V, E')$ of G . In other words, $G[E']$ is obtained from G by removing every edge in $E \setminus E'$.

A *path query* q has the form (x, L, y) where x and y are variables and L is a language. When evaluated on a graph G , it returns the set $q(G)$ of all pairs (s, t) such that s and t are vertices of G and there exists a path p from s to t with $lbl(p) \in L$. An answer (s, t) is viewed as an assignment of s and t to x and y , respectively; this will become important later when we combine multiple path queries. For convenience, we may view q as a function that takes G , s and t as input, where $q[s, t](G) = 1$ if (s, t) is an answer and $q[s, t](G) = 0$ otherwise. As a special case, a *regular path query* (RPQ) is such that L is a regular language, defined via a regular expression r or an automaton A . We sometimes use the shorthand L for the query (x, L, y) , or r in the case of a regular expression.



■ **Figure 1** The graph of our running example. In the paper, we denote the edge (v_i, v_j) as e_{ij} .

► **Example 1.** Figure 1 depicts the graph G over $\Sigma = \{a, b, c\}$ for our running example. We show a few examples of RPQs on G .

- $q_1 = \Sigma^*$. This query tests whether there is a path from s to t in G . For example, we have that $q_1[v_1, v_2](G) = 1$, and $q_1[v_1, v_6](G) = 1$, since there are paths from v_1 to both v_2 and v_6 . In contrast, $q_1[v_3, v_1](G) = 0$ since there is no path from v_3 to v_1 .
- $q_2 = abc$. This query tests whether there is a path from s to t in G that matches the word abc . For example, we have that $q_2[v_1, v_6](G) = 1$, as there is a path $v_1 \rightarrow v_3 \rightarrow v_5 \rightarrow v_6$ that matches abc . But $q_2[v_3, v_5](G) = 0$, as the only path from v_3 to v_5 consists of a single edge labeled b .
- $q_3 = ab^*$. This query tests whether there is a path from s to t in G that matches regular expression ab^* . We have $q_3[v_1, v_6](G) = 1$ due to the path $v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_6$, or alternatively, $v_1 \rightarrow v_2 \rightarrow v_6$, that match ab^* . But we have $q_2[v_3, v_5](G) = 0$, as the only path from v_3 to v_5 consists of a single edge with label b , which is not a match for ab^* .

We will later use these queries to illustrate additional concepts in the paper. ─

Conjunctive Regular Path Queries

A *conjunctive regular path query*, CRPQ for short, is a conjunction of RPQs with possibly shared variables. More precisely, a CRPQ q has the form

$$q[x_1, \dots, x_k] = \bigwedge_{i=1}^m (y_i, r_i, z_i) \quad (1)$$

where each y_i and z_i is a variable from $\{x_1, \dots, x_k\}$ and each r_i is a regular expression. The RPQ (y_i, r_i, z_i) is also referred to as the i th atom of q and is denoted by q_i . As before, when evaluated on a graph G , we denote by $q(G)$ the set of all of assignments (u_1, \dots, u_k) to (x_1, \dots, x_k) , such that all atoms are satisfied. We also denote the assignment (u_1, \dots, u_k) as a function $\mu : \{x_1, \dots, x_k\} \rightarrow \{u_1, \dots, u_k\}$ such that $\mu(x_i) = u_i$ for $i = 1, \dots, k$. We use a numeric notation similarly to RPQs, that is, $q[u_1, \dots, u_k](G) = 1$ if (u_1, \dots, u_k) is an answer and $q[u_1, \dots, u_k](G) = 0$ otherwise.

► **Example 2.** Let us look at the query $q[x_1, x_2, x_3] = (x_1, a^*, x_2) \wedge (x_2, b^*, x_3)$. When evaluated on a graph, this query returns triplets (u_1, u_2, u_3) such that there is a path from u_1 to u_2 of edges labeled a , and from u_2 to u_3 of edges labeled b . In our running example (Figure 1), we have that $q[v_1, v_2, v_6](G) = 1$, as there is a path $v_1 \rightarrow v_2$ that matches a^* , and a path $v_2 \rightarrow v_4 \rightarrow v_6$ that matches b^* . Yet, $q[v_1, v_3, v_6](G) = 0$, since every match from v_3 to v_6 contains a label that is not b . ─

11:6 The Complexity of the Shapley Value for Regular Path Queries

An atom q_j is *redundant* if its removal from q results in a query that is equivalent to q . Formally, denote by $q^{\setminus j}$ the CRPQ that is obtained from q by removing the j th atom.

$$q^{\setminus j}[x_1, \dots, x_k] = \bigwedge_{i=1; i \neq j}^m (y_i, r_i, z_i)$$

Then the j th atom is *redundant* if $q \equiv q^{\setminus j}$, that is, $q(G) = q^{\setminus j}(G)$ for all graphs G .

► **Example 3.** Let us look at $q[x_1, x_2, x_3] = (x_1, a, x_2) \wedge (x_2, b, x_3) \wedge (x_1, a^*b^*, x_3)$. In this query, the third atom is redundant according to our definition, as removing it does not change the result set on any graph. Intuitively, if the first two queries return true then so does the third, thus the third atom does not add any restriction to the conjunction. ◻

We later refer to the following obvious (and standard) observation.

► **Observation 4.** Let q be a CRPQ. If the i th atom is non-redundant, then there exists a graph G and assignment μ to (x_1, \dots, x_k) such that $q_j[\mu(y_j), \mu(z_j)](G) = 1$ for $j \neq i$ and $q_i[\mu(y_i), \mu(z_i)](G) = 0$.

In the sequel, we say that q is *without redundancy* if every atom of q is non-redundant. Note that every CRPQ q can be made one without redundancy (while preserving equivalence) by repeatedly removing redundant atoms.

The Shapley Value

Let A be a finite set of players. A cooperative game is a function $v: P(A) \rightarrow \mathbb{R}$, where $P(A)$ is the power set of A (containing all subsets of A), such that $v(\emptyset) = 0$. For $S \subseteq A$, the value $v(S)$ represents a value, such as wealth, jointly obtained by S when the players of S cooperate. The Shapley value for the player a is defined to be:

$$\text{Shapley}(A, v, a) = \frac{1}{|A|!} \sum_{\pi \in \Pi_A} (v(\pi_a \cup \{a\}) - v(\pi_a)). \quad (2)$$

Here, Π_A is the set of all possible permutations over the players in A , and for each permutation π we denote by π_a the set of players that appear before a in the permutation. Alternatively, the Shapley value can be written as follows:

$$\text{Shapley}(A, v, a) = \sum_{B \subseteq A \setminus \{a\}} \frac{|B|!(|A| - |B| - 1)!}{|A|!} (v(B \cup \{a\}) - v(B)).$$

Intuitively, the Shapley value of a player a is the expected contribution of a to the value $v(B)$ where B is a set of players chosen by randomly (and uniformly) selecting players one by one without replacement. The Shapley value is known to be unique up to some rationality axioms that we omit here (c.f. [32]).

3 The Shapley Value of Edges

Throughout the paper, we focus on the Shapley value of edges of the input graph G . Later, in Section 6, we also discuss the extension of our results to the Shapley value of vertices.

Given a CRPQ q , our goal is to quantify the contribution of edges in the input graph G to an answer \vec{u} for q . We adopt the convention that, for the sake of measuring contribution, the database is viewed as consisting of two types of data items – we reason about the

contribution of the *endogenous* items while we take for granted the existence of the *exogenous* items (that serve as out-of-game background) [16, 25, 31]. Hence, in our setup, we view the graph as consisting of two types of edges: *endogenous edges* and *exogenous edges*. For a graph $G = (V, E)$, we denote by E_n and E_x the sets of endogenous and exogenous edges, respectively, and we assume that E is the disjoint union of E_n and E_x .

Our goal is to quantify the contribution of an edge $e \in E_n$ to an answer $\vec{u} = (u_1, \dots, u_k)$ of the query q , that is, to the fact that $q[\vec{u}](G) = 1$. To this end, we view the situation as a cooperative game where the players are the endogenous edges. The Shapley value of an edge $e \in E_n$ in this setting will be denoted by $\text{Shapley}\langle q \rangle(G, \vec{u}, e)$.

$$\text{Shapley}\langle q \rangle(G, \vec{u}, e) \stackrel{\text{def}}{=} \text{Shapley}(E_n, v_q, e)$$

where the function Shapley is as defined in Equation (2) and v_q is the numerical function that takes as input a subset of the endogenous edges and is defined as follows:

$$v_q(B) \stackrel{\text{def}}{=} q[\vec{u}](G[B \cup E_x]) - q[\vec{u}](G[E_x])$$

In particular, $v_q(\emptyset) = 0$. Put differently, we have the following.

$$\begin{aligned} \text{Shapley}\langle q \rangle(G, \vec{u}, e) = \\ \sum_{B \subseteq E_n \setminus \{e\}} \frac{|B|!(|E_n| - |B| - 1)!}{|E_n|!} \left(q[\vec{u}](G[B \cup E_x \cup \{e\}]) - q[\vec{u}](G[B \cup E_x]) \right) \quad (3) \end{aligned}$$

For a CRPQ q , the computational problem $\text{CRPQShapley}\langle q \rangle$ is that of computing the Shapley value of a given edge:

Problem $\text{CRPQShapley}\langle q \rangle$	
Parameter:	CRPQ q
Input:	Graph G , vertex vector $\vec{u} = (u_1, \dots, u_k)$, endogenous edge e
Goal:	Compute $\text{Shapley}\langle q \rangle(G, \vec{u}, e)$

When q has only one atom, and is in fact an RPQ (x, r, y) with r being a regular expression, we may replace q with r in the notation and write $\text{Shapley}\langle r \rangle(G, s, t, e)$ and $\text{RPQShapley}\langle r \rangle$ with the meaning of $\text{Shapley}\langle q \rangle(G, s, t, e)$ and $\text{RPQShapley}\langle q \rangle$, respectively.

► **Example 5.** Considering the running example of Figure 1, assume that all edges are endogenous. Let us first compute the contribution of the edges to the answer (v_2, v_6) to b^* .

- The edge e_{26} changes $q[v_2, v_6](G)$ from 0 to 1 if and only if it is selected first or second among $\{e_{24}, e_{26}, e_{46}\}$. This event happens with probability $2/3$, so

$$\text{Shapley}\langle b^* \rangle(G, v_2, v_6, e_{26}) = 2/3.$$

- For e_{24} to increase the value, it should be selected before e_{26} and after e_{46} , and this happens with probability $1/6$. Hence, $\text{Shapley}\langle b^* \rangle(G, v_2, v_6, e_{24}) = 1/6$.
- Similarly to e_{24} , $\text{Shapley}\langle b^* \rangle(G, v_2, v_6, e_{46}) = 1/6$.
- Every other edge is irrelevant to the answer (v_2, v_6) , and so, its Shapley value is zero.

Note that the sum of the Shapley values of all edges is 1, which is no coincidence, since in general the Shapley value over all players sums up to the overall wealth of the entire set of players [32]. Following are additional examples.

11:8 The Complexity of the Shapley Value for Regular Path Queries

- $\text{Shapley}\langle abc \rangle(G, v_1, v_6, e)$. Any edge that is not on the only path that matches abc , namely $p: v_1 \rightarrow v_3 \rightarrow v_5 \rightarrow v_6$, will have the Shapley value of zero. For edges on the path p , the computations are similar to each other and they all have the same Shapley value. For one of them to change the query result, it needs to appear after both other edges in the permutation of E_n . This happens in $\frac{9!}{3}$ of the overall $9!$ permutations. So we have:

$$\begin{aligned} \text{Shapley}\langle abc \rangle(G, v_1, v_6, e_{13}) &= \text{Shapley}\langle abc \rangle(G, v_1, v_6, e_{35}) \\ &= \text{Shapley}\langle abc \rangle(G, v_1, v_6, e_{56}) = \frac{1}{3}. \end{aligned}$$

If we assume that e_{13} is exogenous, then the other two edges will *split* the contribution evenly. Then we get:

$$\text{Shapley}\langle abc \rangle(G, v_1, v_6, e_{35}) = \text{Shapley}\langle abc \rangle(G, v_1, v_6, e_{56}) = \frac{1}{2}.$$

- $\text{Shapley}\langle ab^* \rangle(G, v_1, v_6, e)$. There are two paths that match the regular expression ab^* (as we have seen in Example 1). Again, any edge that is not on any of these paths has the Shapley value zero. But now, the contributions of the remaining edges is not equal since, for instance, e_{12} is on both paths so we expect it to have higher contribution than the others. For the edge e_{26} to change the query result, it needs to appear after edge e_{12} but before at least one of e_{24} and e_{46} . Permutations where this happens are either permutations where e_{26} appears after e_{12} and one of e_{24} and e_{46} but before the other one, and there are $2 \cdot \sum_{i=0}^5 (i+2)! \binom{5}{i} (8-i-2)! = \frac{2}{12} \cdot 9!$ such permutations. This is also possible in permutations where e_{26} appears after e_{12} but before both e_{24} and e_{46} , and there are $\sum_{i=0}^5 (i+1)! \binom{5}{i} (8-i-1)! = \frac{1}{12} \cdot 9!$ such permutations. There are an overall of $9!$ possible permutations, so,

$$\text{Shapley}\langle ab^* \rangle(G, v_1, v_6, e_{26}) = \frac{1}{4}.$$

For the two edges e_{24} and e_{46} , the computations are similar to each other. For one of them to change the query, it needs to appear after e_{12} and the other one, but before e_{26} . Similar to before, there are $\sum_{i=0}^5 (i+2)! \binom{5}{i} (8-i-2)! = \frac{1}{12} \cdot 9!$ permutations where this happens, so,

$$\text{Shapley}\langle ab^* \rangle(G, v_1, v_6, e_{24}) = \text{Shapley}\langle ab^* \rangle(G, v_1, v_6, e_{46}) = \frac{1}{12}.$$

For the last edge e_{12} , it needs to appear after e_{26} or after both e_{24} , e_{46} . Permutations where this happens are either permutations where e_{12} appears second after e_{26} , and these account for $\frac{1}{4} \cdot \frac{1}{3} \cdot 9! = \frac{1}{12} \cdot 9!$ of all permutations, or permutations where e_{12} appears third or fourth, and these account for $\frac{1}{2} \cdot 9!$ of all permutations. So overall we get that

$$\text{Shapley}\langle ab^* \rangle(G, v_1, v_6, e_{12}) = \frac{7}{12}.$$

Note that, again, $\sum_{e \in E_n} \text{Shapley}\langle ab^* \rangle(G, v_1, v_6, e) = 1$, as expected. \lrcorner

4 The Complexity of Exact Computation

In this section, we study the complexity of $\text{CRPQShapley}\langle q \rangle$, where the goal is to compute the exact Shapley value of an edge. Note that the query q is fixed in the analysis, hence, every q defines a separate computational problem $\text{CRPQShapley}\langle q \rangle$. The following theorem show that $\text{CRPQShapley}\langle q \rangle$ is computationally intractable for almost *every* CRPQ q , except for limited cases. We prove the theorem later, in Section 4.1.

► **Theorem 6** (Hardness). *Let q be a CRPQ. If q has a non-redundant atom with a language that contains a word of length three or more, then $\text{CRPQShapley}\langle q \rangle$ is $\text{FP}^{\#\text{P}}$ -complete.*

Recall that $\text{FP}^{\#\text{P}}$ is the class of functions computable in polynomial time with an oracle to a problem in $\#\text{P}$ (e.g., counting the number of satisfying assignments of a propositional formula). This class is considered intractable, and above the polynomial hierarchy (Toda's theorem [33]).

The question of whether the condition of Theorem 6 is necessary for hardness remains open. Yet, we can show that it is, indeed, necessary, in the case of a single atom (RPQ):

► **Theorem 7** (Tractability). *Let q be an RPQ with the regular expression r . If every word in $L(r)$ is of length at most two, then $\text{RPQShapley}\langle q \rangle$ is solvable in polynomial time.*

Proof. We give a polynomial-time algorithm for computing $\text{RPQShapley}\langle r \rangle$ where $L = L(r)$ consists of words of length at most two. We denote by $\mathcal{M}(G, k)$ the set of all subsets $E' \subseteq E_n$ of size k such that $G[E_x \cup E']$ contains a path of L from s to t . We have the following from Equation (3):

$$\begin{aligned} \text{RPQShapley}\langle r \rangle(G, s, t, e) &= \sum_{k=0}^{m'-1} \frac{k!(m'-k-1)!}{m'} |\mathcal{M}(G_e, k)| \\ &\quad - \sum_{k=0}^{m'-1} \frac{k!(m'-k-1)!}{m'} |\mathcal{M}(G \setminus e, k)|. \end{aligned}$$

Here, G_e is the same as G , except for e that is exogenous instead of endogenous, $G \setminus e$ is the graph G with the exclusion of e , and $m' = |E_n|$. This shows that the computation of $\text{RPQShapley}\langle r \rangle(G, s, t, e)$ reduces efficiently to computing $|\mathcal{M}(G, k)|$, that is, counting the subsets of E_n (of endogenous edges) of size k that, when added to E_x , connects s to t via a path that matches a word in $w \in L$.

We assume that L does not contain the empty word. This is without loss of generality, for the following reason. If L contains the empty word ϵ , then either $s = t$ and e has the Shapley value zero (since it is irrelevant), or $s \neq t$ and we can ignore the empty word of L .

We now show that $|\mathcal{M}(G, k)|$ can be computed in polynomial time when $L(r)$ consists of words of length at most two. First, let us observe that we can compute $|\mathcal{M}(G, k)|$ by computing the complement set $|\overline{\mathcal{M}(G, k)}|$ which is defined similarly but for subsets of length k where there is *no* path in L :

$$|\mathcal{M}(G, k)| = \binom{m'}{k} - |\overline{\mathcal{M}(G, k)}|.$$

So, it suffices to show how to compute $|\overline{\mathcal{M}(G, k)}|$.

For a subset of endogenous edges to be in $\overline{\mathcal{M}(G, k)}$, it should not connect, with E_x , any path from s to t matching $w \in L$ (i.e., matching one of w_0, \dots, w_l). In other words, it should not connect any path of length one matching some w_i with $|w_i| = 1$, or any path of length two matching some w_i with $|w_i| = 2$. This partitions the set of endogenous edges into three categories:

- **Permitted:** Edges that are not part of any path that matches L .
- **Forbidden:** Edges that connect s to t without needing any other endogenous edge, either because they have a label that constitutes a word w_i , or because they connect a path of length two together with an exogenous edge.
- **On2Path:** All other edges, that is, the edges that belong to pairs of endogenous edges that are needed together in order to connect s to t through a word in L .

11:10 The Complexity of the Shapley Value for Regular Path Queries

Observe that following. First, the three sets Permitted, Forbidden and On2Path are pairwise disjoint (by definition). Second, On2Path can be partitioned into $|\text{On2Path}|/2$ pairwise-disjoint pairs, each constitutes a path with a word in L . (Note that our data model does not allow for parallel edges.)

It follows that to construct a set of k edges in $\overline{\mathcal{M}(G, k)}$, we can select $i \leq k$ edges from Permitted, then $k - i$ pairs from the $|\text{On2Path}|/2$ pairs, and then one edge from each pair. Hence, we get:

$$|\overline{\mathcal{M}(G, k)}| = \sum_{i=0}^k \binom{|\text{Permitted}|}{i} \cdot \binom{\frac{|\text{On2Path}|}{2}}{k-i} \cdot 2^{k-i}. \quad (4)$$

Finally, observe that we can compute each of Permitted, Forbidden and On2Path in polynomial time, and we can then compute Equation (4) in polynomial time. This concludes the proof. ◀

Hence, we get a full classification for RPQs:

► **Corollary 8.** *Let q be an RPQ with the regular expression r . Assuming $P \neq NP$, the following are equivalent:*

1. $\text{RPQShapley}\langle q \rangle$ is solvable in polynomial time.
2. Every word in $L(r)$ is of length at most two.

In the remainder of this section, we prove the hardness side (Theorem 6).

4.1 Proof of Hardness

Membership in $\text{FP}^{\#P}$ is straightforward from the definition of the Shapley value in Equation (2). Indeed, $\text{Shapley}\langle q \rangle(G, \vec{u}, e)$ can be computed using an oracle to the problem of counting the permutations over the edge set such that e changes the evaluation from zero (false) to one (true). For the $\text{FP}^{\#P}$ -hardness, we prove it in a sequence of reductions. We begin with hardness for the special case where the language consists of a single three-letter word. For that, we will use a result by Livshits et al. [17] on the computation of Shapley values for facts (tuples) in relational databases. We use that to prove hardness for the general case of a language with one or more words of length at least three, even when restricted to simple graphs.

We first recall the result of Livshits et al. [17]. They considered relational databases D where some of the facts are endogenous and the rest exogenous. As in our notation, the corresponding subsets of D are denoted by D_n and D_x , respectively. For a Boolean query q that maps every database into $\{0, 1\}$, they defined the Shapley value of a fact similarly to the way we define the Shapley value of an edge: the endogenous facts are the players and the query is the wealth function:

$$\text{Shapley}\langle q \rangle(D, f) = \text{Shapley}(D_n, v_{ab}, f)$$

where $v_{ab}(E) = q(E \cup D_x) - q(D_x)$. They established a complete classification of the class of conjunctive queries without self-joins into tractable and intractable queries for the computation of the Shapley value. What is relevant to us is that the following conjunctive query is $\text{FP}^{\#P}$ -hard:

$$Q_{\text{RST}}(): \exists x, y [R(x) \wedge S(x, y) \wedge T(y)]$$

We define a special kind of graphs that will help us in some of the proofs. A graph $G = (V, E)$ is called a *leveled graph* if there exists a split of the vertex set into levels V_0, \dots, V_k , such that:

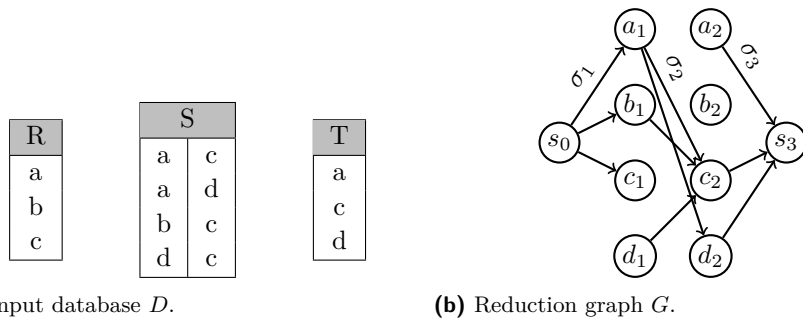


Figure 2 An example for the construction in the reduction of the proof of Lemma 9.

1. The set of vertices V is the disjoint union of V_0, \dots, V_k .
2. Every edge is from a vertex of some level V_i to a vertex of V_{i+1} .

From the hardness of the Shapley value for Q_{RST} , it is easy to prove the following.

► **Lemma 9.** *Let $\sigma_i \in \Sigma$ for $i = 1, 2, 3$. $\text{RPQShapley}\langle\sigma_1\sigma_2\sigma_3\rangle$ is $\text{FP}^{\#P}$ -hard, even when restricted to leveled graphs.*

The proof (given in the long version of the paper) is via the reduction illustrated in Figure 2. Next, we have the following generalization of Lemma 9.

► **Lemma 10.** *Let r be a regular expression. If there exists a word in $L(r)$ of length at least three, then $\text{RPQShapley}\langle r \rangle$ is $\text{FP}^{\#P}$ -hard, even when restricted to leveled graphs.*

The reduction from the problem of Lemma 9 to that of Lemma 10 is illustrated in Figure 3 (and explained in the full version). With Lemma 10, we can prove Theorem 6.

Proof of Theorem 6. We know that q has a non-redundant atom q_i such that $L(r_i)$ contains a word of length at least three. We reduce $\text{RPQShapley}\langle r_i \rangle$ on leveled graphs (Lemma 10) to $\text{CRPQShapley}\langle q \rangle$. Given an input leveled graph G , source vertex s , target vertex t and edge e for $\text{RPQShapley}\langle r_i \rangle$, we construct an input instance G^* for $\text{CRPQShapley}\langle q \rangle$.

Since the i th atom is non-redundant, we can use Observation 4 and conclude that there exists a graph G_i and assignment \vec{v} to \vec{x} such that all RPQ atoms return true except for the i th atom; that is, we have that $q_j[s_j, t_j](G_i) = 1$ for every $j \neq i$ and $q_i[s_j, t_j](G_i) = 0$. Here, s_j and t_j are the vertices assigned to the variables y_j and z_j , respectively, from Equation (1).

We assume that G and G_i are disjoint. We construct a new graph G^* by adding G to G_i , and merging s and t into s_i and t_i , respectively. Hence, in G^* , the vertex s_i has all edges that it has in G_i , in addition to the outgoing edges that s has in G . Similarly, in G^* , the

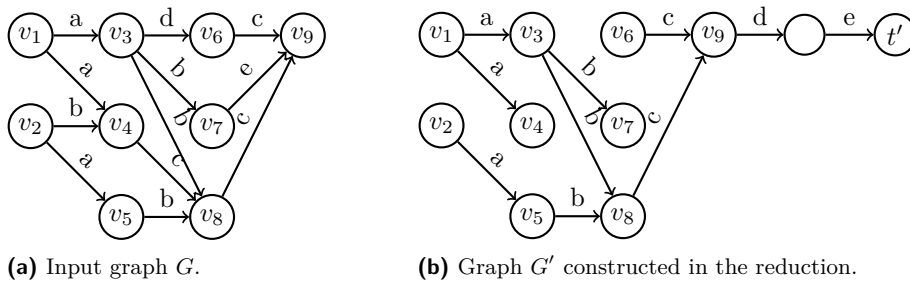


Figure 3 An example for the reduction in Lemma 10, for a regular expression that accepts the word $abcde$, source vertex $s = v_1$, target vertex $t = v_9$.

11:12 The Complexity of the Shapley Value for Regular Path Queries

vertex t_i has all of the edges that it has in G_i , in addition to the incoming edges that t has in G . In G^* , we set all of the edges of G_i to be exogenous ones. Moreover, G^* and G have the same set of endogenous edges.

To complete the proof, observe that $q[\vec{v}](G^*)$ is equal to $q_i[s, t](G)$. To see that, observe that G^* has a matching path for the j th atom for all $j \neq i$ (since G_i does). Recall also that there are no paths from s_i to t_i matching r_i in G_i . Hence, from our construction of G^* (and in particular given that s_i and t_i are not part of any cycle), we get that every path from s_i to t_i that matches r_i should be fully contained in G . Hence, G^* has a q_i path from s_i to t_i if and only if G has a q_i path from s to t .

Let E_x and E_n be the sets of exogenous and endogenous edges of G , respectively, and let E_x^* be the set of exogenous edges of G^* . We can extend the above argument to conclude that

$$q[\vec{v}](G^*[E_x^* \cup E']) = q_i[s, t](G[E_x \cup E'])$$

for all subsets E' of E_n , since every edge of G_i is exogenous in G^* . From that we can now conclude that $\text{Shapley}\langle r_i \rangle(G, s, t, e) = \text{Shapley}\langle q \rangle(G^*, \vec{v}, e)$, as claimed. \blacktriangleleft

This completes the proof of the hardness side of Theorem 6.

5 Complexity of Approximation

We now study the complexity of approximating $\text{CRPQShapley}\langle q \rangle$. We aim for a *fully polynomial randomized approximation scheme*, or FPRAS for short. Formally, an FPRAS for a numeric function f is a randomized algorithm $A(x, \epsilon, \delta)$, where x is an input for f and $\epsilon, \delta \in (0, 1)$, such that $A(x, \epsilon, \delta)$ 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 $\log(1/\delta)$. We distinguish between an *additive* FPRAS:

$$\Pr [f(x) - \epsilon \leq A(x, \epsilon, \delta) \leq f(x) + \epsilon] \geq 1 - \delta$$

and a *multiplicative* FPRAS:

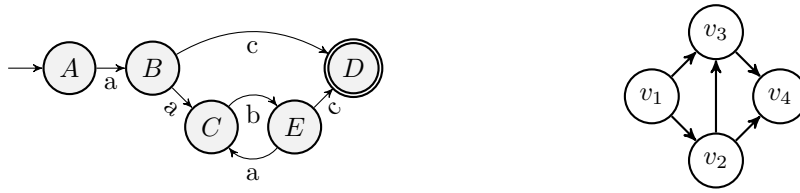
$$\Pr \left[\frac{f(x)}{1 + \epsilon} \leq A(x, \epsilon, \delta) \leq (1 + \epsilon)f(x) \right] \geq 1 - \delta.$$

5.1 Results

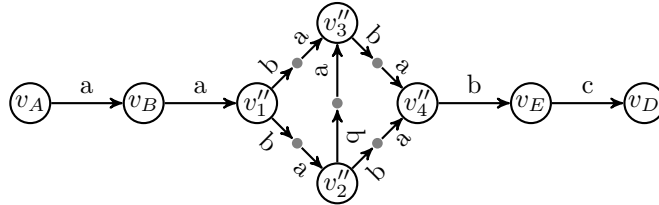
There is a simple Monte-Carlo algorithm that guarantees an additive approximation for the Shapley value on any CRPQ, and we show that it also serves as a multiplicative FPRAS in some cases. In this section, we establish a dichotomy in the complexity of multiplicative approximation for the class of CRPQs. We note that here and later on, we sometimes give results for general CRPQs, yet without redundancy. These results generalize to CRPQs with redundant atoms by application to any CRPQ obtained by repeatedly eliminating redundancy (as mentioned in Section 2).

► Theorem 11. *Let q be a CRPQ without redundancy. If $L(r_i)$ is finite for every atom r_i of q , then $\text{CRPQShapley}\langle q \rangle$ has a multiplicative FPRAS. Otherwise, $\text{CRPQShapley}\langle q \rangle$ has no multiplicative approximation (of any ratio) or else $NP \subseteq BPP$.*

In the remainder of this section, we prove Theorem 11, starting with the hardness side (Section 5.2) and moving on to the FPRAS algorithm (Section 5.3).



(a) The DFA for regular expression $a(a + b)^*c$. (b) Input graph G .



(c) The graph G' of the reduction for input instance (G, v_1, v_4, e) .

■ **Figure 4** An example for the construction in the reduction of the proof of Lemma 15.

5.2 Proof of Hardness

For the hardness, we use a direct consequence of the characterization of Fortune, Hopcroft and Wyllie [12] of the *subgraph homeomorphism problem*:

► **Proposition 12.** *It is NP-complete to determine, given a graph G , vertices s and t , and edge e , whether e lies on any simple path from s to t .*

Equipped with Proposition 12, we can now show the hardness for Σ^* using the following characterization of when the Shapley value of an edge is nonzero.

► **Lemma 13.** *Let G be a graph where all edges are endogenous. Let s and t be two vertices of G , and e an edge of G . $\text{Shapley}\langle \Sigma^* \rangle(G, s, t, e) > 0$ if and only if e belongs to a simple path from s to t .*

Proof. Denote by q the RPQ (x, Σ^*, y) . We handle separately each direction of the claim. If $\text{Shapley}\langle \Sigma^* \rangle(G, s, t, e) > 0$, then it follows from the definition of the Shapley value that there exists a subset E' of the edges such that $q[s, t](G[E']) = 0$ and $q[s, t](G[E' \cup \{e\}]) = 1$. Let $G' = G[E' \cup \{e\}]$. Then G' contains a simple path from s to t . If this simple path does not contain e , then it is a simple path in $G[E']$, which contradicts the fact that $q[s, t](G[E']) = 0$. Conversely, suppose that e lies on a simple path P from s to t in G . If the random selection of Shapley selects precisely all edges of P except for e , then the addition of e would change the result from 0 to 1. Hence, the Shapley value is nonzero. ◀

Hence, from Proposition 12 and Lemma 13 we conclude the following corollary, which proves the hardness part of Theorem 11 for the language Σ^* .

► **Corollary 14.** *It is NP-complete to determine whether $\text{Shapley}\langle \Sigma^* \rangle(G, s, t, e) > 0$, given G , s , t and e , even if all edges of G are assumed to be endogenous.*

Next, we generalize Corollary 14 from Σ^* to any arbitrary infinite regular language r .

► **Lemma 15.** *Let r be a regular expression. If $L(r)$ is infinite, then it is NP-complete to determine whether $\text{Shapley}\langle r \rangle(G, s, t, e) > 0$.*

Proof sketch. It is straightforward to show that the problem is in NP, as any subset of endogenous edges that adding e to it connects a matching path serves as a witness and can be verified in polynomial time. We will prove NP-hardness by showing a reduction from the problem of determining whether $\text{Shapley}\langle\Sigma^*\rangle(G, s, t, e) > 0$ where all edges are endogenous, and then apply Corollary 14. Given an input instance (G, s, t, e) , we will show how to construct an instance (G', s', t', e') for our problem so that $\text{Shapley}\langle\Sigma^*\rangle(G, s, t, e) > 0$ if and only if $\text{Shapley}\langle r \rangle(G', s', t', e') > 0$.

Since $L(r)$ is infinite, we know that its corresponding DFA A has at least one cycle. We find a path from an initial state to an accepting state that passes through a state s of A that participates in a cycle. We will denote the path by: $l : s_0 \rightarrow \dots \rightarrow s_i \rightarrow \dots \rightarrow s_k$ where $s_i = s$.

We assumed s is a part of a cycle. Let us denote the labels along the cycle starting, from s , by $w_o = \sigma_0 \dots \sigma_c$. The graph G' is constructed from G so that every path from s' to t' matches r in the following way. The graph G' consist of three subgraphs, as illustrated in Figure 4.

- A copy of the graph G where every edge is replaced with a fresh path of c edges with labels matching w_o . We denote the correspondent of each vertex v of G as v'' . Hence, s and t become s'' and t'' , respectively. All of the edges in this part are endogenous.
- A copy of the path $s_0 \rightarrow \dots \rightarrow s_i$, with the same labels as in the DFA A , where we identify s_i with s'' . The edges of this part are all exogenous.
- The path $s_i \rightarrow \dots \rightarrow s_k$ with the same labels as in the DFA A , there we now identify s_i with t'' . The edges of this part are all exogenous.

We now define s' to be the copy of s_0 , we define t' to be the copy of s_k , and we choose as e' any edge along the path that replaces e .

From here it is easy to prove that e belongs to a simple path of G from s to t if and only if e' belongs to a simple path of G' from s' to t' , and from there we conclude similarly to Lemma 13 that $\text{Shapley}\langle\Sigma^*\rangle(G, s, t, e) > 0$ if and only if $\text{Shapley}\langle r \rangle(G', s', t', e') > 0$. ◀

Finally, we extend Lemma 15 from RPQs to CRPQs similarly to the way we proved Theorem 6.

► **Lemma 16.** *Let q be a CRPQ without redundancy. If $L(r_i)$ is infinite for some atom r_i of q , then determining whether $\text{Shapley}\langle q \rangle(G, \vec{u}, e) > 0$ is NP-complete.*

This completes the proof of the hardness side of Theorem 11. Next, we will prove the positive side.

5.3 Proof of Tractability

We now show that for every CRPQ where the hardness condition of Theorem 11 does not hold, a multiplicative FPRAS exists. We start by showing that in this case, the *gap property* (as defined by Livshits et al. [17]) holds: if the Shapley value is nonzero, then it must be at least the reciprocal of a polynomial.

► **Lemma 17.** *Let q be a fixed CRPQ without redundancy. If $L(r_i)$ is finite for every atom r_i of q , then there exists a polynomial p such that $\text{Shapley}\langle q \rangle(G, \vec{u}, e)$ is either zero or at least $1/p(|G|)$.*

Proof. If there is no subset E' of E_n such that adding e to $E' \cup E_x$ changes the value of query q from false to true, then $\text{Shapley}\langle q \rangle(G, \vec{u}, e) = 0$. Otherwise, let E' be a minimal such set. Then $|E'| \leq k_1 + \dots + k_m$, where each k_i is the length of the longest word in $L(r_i)$; the

language for the i -th atom in q , as at worst case, the paths match the longest word for each RPQ. Since each $L(r_i)$ is finite, every k_i is a finite constant, and so, $k = k_1 + \dots + k_m$ is a constant.

Returning to the definition of the Shapley value (Equation (2)), the probability of selecting a permutation π such that π_e is exactly $E' \setminus \{e\}$ is

$$\frac{(|E| - 1)!(m' - |E|)!}{m'!} \geq \frac{(m' - k)!}{m'!}$$

where $m' = |E_n|$. Hence, we have that

$$\text{Shapley}\langle q \rangle(G, \vec{u}, e) \geq \frac{(m' - k)!}{m'!} = \frac{1}{(m' - k + 1) \cdot \dots \cdot m'}.$$

This completes the proof. ◀

Similarly to Livshits et al. [17], we can use the gap property to show that an additive FPRAS can be turned into a multiplicative FPRAS.

► **Lemma 18.** *Let q be a CRPQ without redundancy. If $L(r_i)$ is finite for every atom r_i of q , then $\text{CRPQShapley}\langle q \rangle$ has both an additive and a multiplicative FPRAS.*

Proof. Using the Chernoff-Hoeffding bound, we can get an additive FPRAS of the value $\text{Shapley}\langle q \rangle(G, \vec{u}, e)$, by simply taking the ratio of successes over $O(\log(1/\delta)/\epsilon^2)$ trials of the following experiment:

- Select a random permutation (e_1, \dots, e_m) over the set of endogenous edges E_n .
- Suppose that $e = e_i$, and let $E_{i-1} = \{e_1, \dots, e_{i-1}\}$. If $q[\vec{u}](G[E_{i-1} \cup E_x \cup \{e\}]) = 1$ and $q[\vec{u}](G[E_{i-1} \cup E_x]) = 0$, then report “success,” otherwise, report “failure.”

From Lemma 17 (the gap property), we conclude that in order to get a multiplicative ϵ -approximation, it suffices to apply an additive ϵ' -approximation where $1/\epsilon'$ is polynomial in the size of G and in $1/\epsilon$. ◀

5.4 Open Problem: Directed Acyclic Graphs

It is worth noting that the proof of hardness fails when the graph is acyclic, as it relies on Proposition 12. The lemma states that it is NP-complete to determine whether a given graph G has a simple path from a given source vertex s to a given target vertex t through a given edge e . While this is true in the case of a general graph G , the problem is easily solvable in polynomial time when the graph is acyclic, since every path is simple. This leaves open the question of whether we can have a multiplicative FPRAS for the Shapley value even for RPQs with an infinite language. We leave this question for future investigation. Note, however, that for the exact computation there is no change even when restricted to DAGs, since the reductions constructed DAGs.

6 Shapley Value of Vertices

In this section, we discuss the complexity of the Shapley value for *vertices*, rather than *edges*, of the graph. Similarly to the case for edges, our goal is to quantify the contribution of vertices in the input graph to an answer of a path query. So, now, the graph consists of two types of vertices: *endogenous vertices* and *exogenous vertices*.

11:16 The Complexity of the Shapley Value for Regular Path Queries

For a graph $G = (V, E)$, we denote by V_n and V_x the sets of endogenous and exogenous vertices, respectively, and we assume that V is the disjoint union of V_n and V_x . If U is a set of vertices, then $G[U]$ denotes the subgraph of G that is induced by U ; hence, the vertex set of $G[U]$ is U and the edge set of $G[U]$ consists of every edge of G with both endpoints in U . We denote by $\text{Shapley}\langle q \rangle(G, \vec{u}, w)$ the Shapley value of a vertex $w \in V_n$, that is:

$$\text{Shapley}\langle q \rangle(G, \vec{u}, w) \stackrel{\text{def}}{=} \text{Shapley}(V_n, v_q^v, w)$$

where the function Shapley is as defined in Equation (2) and v_q^v is defined as follows:

$$v_q^v(B) \stackrel{\text{def}}{=} q[\vec{u}](G[B \cup V_x]) - q[\vec{u}](G[V_x])$$

We denote by $\text{CRPQShapley}^v\langle q \rangle$ and $\text{RPQShapley}^v\langle r \rangle$ the computational problems that correspond to the ones defined earlier for the Shapley values of edges. We now state the results that we establish with some notes on the changes that should be made in the proofs.

6.1 Complexity of Exact Computation

We can show the following regarding the exact computation of the Shapley value of a graph vertex.

► **Theorem 19.** *The following hold for a CRPQ q .*

1. *If q has a non-redundant atom with a language that contains a word of length four or more, then $\text{CRPQShapley}^v\langle q \rangle$ is $\text{FP}^{\#P}$ -complete.*
2. *If q is an RPQ with the regular expression r , and every word in $L(r)$ is of length at most two, then $\text{RPQShapley}^v\langle q \rangle$ is solvable in polynomial time.*

Note that we leave a gap in the classification of RPQs. Theorem 19 states that if there exists a word of length four or more, then the problem is hard, and if all words are of length at most two, then the problem is solvable in polynomial time. The case where there are words of length three but not longer remains an open problem (as opposed to the case of edges where we had a full dichotomy on RPQs due to Corollary 8).

The proof of the hardness part is almost the same as the proof of Theorem 6 for the case of edges. We begin with hardness for the special case where the regular language (or any language) consists of a single four-letter word (rather than three in the case of edges). For that, we use the same result by Livshits et al. [16] on the computation of Shapley values for facts in relational databases. We then continue with the same sequence of reductions as done for edges to get the hardness for a general CRPQ. The proof of the tractability part is also similar to the proof of Theorem 7.

6.2 Complexity of Approximation

For calculating the Shapley value approximately, we get the exact same dichotomy for vertices as Theorem 11 for edges.

► **Theorem 20.** *Let q be a CRPQ without redundancy. If $L(r_i)$ is finite for every atom r_i of q , then $\text{CRPQShapley}^v\langle q \rangle$ has a multiplicative FPRAS. Otherwise, $\text{CRPQShapley}^v\langle q \rangle$ has no multiplicative approximation (of any ratio) or else $\text{NP} \subseteq \text{BPP}$.*

The proof is also similar to that of Theorem 11. We establish an FPRAS through a straightforward additive approximation and the gap property. For hardness, we know from Fortune, Hopcroft and Wyllie [12] that it is NP-complete to determine whether a vertex

v lies on a simple path from s to t in a given graph G , and from that we conclude that deciding whether $\text{Shapley}(\Sigma^*)(G, s, t, v) > 0$ is also NP-complete. From there we continue with a sequence of reductions that is similar to what we have for the case of edges.

6.3 Summary

We conclude that the complexity for both exact computation and approximation of the Shapley value of vertices is very similar to the case of edges. It is generally hard to compute exact values; it is sufficient for the CRPQ to have an atom that is non-redundant and contains a word of length four or more for the computation to be hard, while for RPQs we identify that the tractable family of queries for edges is also tractable for vertices. For approximation, we have an identical dichotomy for the existence of a multiplicative FPRAS.

7 Concluding Remarks

This work continues the research of responsibility and contribution in databases. We presented the graph-database perspective where the queries are (conjunctive) regular path queries, and the responsibility measure is the Shapley value. We investigated the data complexity of the Shapley value of edges in the graph. For the exact computation, we showed that it is generally hard, while we also showed a specific family of CRPQs where the computation can be done in polynomial time. This is not a full dichotomy for the class of CRPQs, but we establish a dichotomy for the class of RPQs. It remains an open problem whether the condition we have for hardness defines a full dichotomy on CRPQs. We have also studied the complexity of computing an approximation of the Shapley value in the form of an FPRAS. An additive FPRAS is easy to achieve using Monte-Carlo sampling, while a multiplicative approximation is harder. We showed a family of CRPQs where the gap property holds, and hence, an additive FPRAS can be transformed into a multiplicative one. These are the CRPQs where every atom has a finite language. For the other CRPQs, we showed that it is hard to obtain any multiplicative approximation (assuming no redundant atoms). Thus, we achieved a dichotomy on CRPQs for the case of approximation. Finally, we showed that the complexity picture is quite similar (up to a small gap) if we compute the Shapley value of vertices rather than edges.

Several problems remain open. We still do not have a full coverage of all CRPQs for the exact computation of Shapley values. In the case of vertices, we still have a gap already for RPQs. In addition, the proof of the hardness of approximation in Section 5.2 is not valid when the input graph is acyclic; this raises the question of whether there are better opportunities of efficient approximations when the problem is restricted to acyclic graphs. It is also interesting to understand the impact on complexity of adopting other semantics for RPQ evaluation, such as simple paths and shortest paths [22]. Another direction is investigating richer path languages, for example, allowing existentially quantified variables in the query, or negated atoms.

References

- 1 Manish Kumar Anand, Shawn Bowers, and Bertram Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT*, volume 426, pages 287–298. ACM, 2010. doi:10.1145/1739041.1739078.
- 2 Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In Maurizio Lenzerini and Thomas Schwentick, editors, *PODS*, pages 305–316. ACM, 2011. doi:10.1145/1989284.1989312.

11:18 The Complexity of the Shapley Value for Regular Path Queries

- 3 Pablo Barceló Baeza. Querying graph databases. In Richard Hull and Wenfei Fan, editors, *PODS*, pages 175–188. ACM, 2013. doi:10.1145/2463664.2465216.
- 4 Pablo Barceló, Leonid Libkin, Anthony W Lin, and Peter T Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems (TODS)*, 37(4):1–46, 2012.
- 5 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. In *PODS*, pages 194–204. ACM, 1999. doi:10.1145/303976.303996.
- 6 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185. Morgan Kaufmann, 2000.
- 7 Mariano P. Consens and Alberto O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS*, pages 404–416. ACM, 1990. doi:10.1145/298514.298591.
- 8 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In *SIGMOD*, pages 323–330. ACM, 1987. doi:10.1145/38713.38749.
- 9 Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. Computing the Shapley value of facts in query answering. In *SIGMOD*, pages 1570–1583. ACM, 2022.
- 10 Wenfei Fan. Graph pattern matching revised for social network analysis. In *ICDT*, pages 8–21. ACM, 2012. doi:10.1145/2274576.2274578.
- 11 Daniela Florescu, Alon Y. Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In Alberto O. Mendelzon and Jan Paredaens, editors, *PODS*, pages 139–148. ACM, 1998. doi:10.1145/275487.275503.
- 12 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980. doi:10.1016/0304-3975(80)90009-2.
- 13 Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach: Part 1: Causes. In *UAI*, pages 194–202, 2001.
- 14 Anthony Hunter and Sébastien Konieczny. On the measure of conflicts: Shapley inconsistency values. *Artif. Intell.*, 174(14):1007–1026, 2010.
- 15 Majd Khalil and Benny Kimelfeld. The complexity of the Shapley value for regular path queries, 2022.
- 16 Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. Query games in databases. *SIGMOD Rec.*, 50(1):78–85, 2021.
- 17 Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. The Shapley value of tuples in query answering. *Log. Methods Comput. Sci.*, 17(3), 2021.
- 18 Ester Livshits and Benny Kimelfeld. The Shapley value of inconsistency measures for functional dependencies. In *ICDT*, volume 186 of *LIPICs*, pages 15:1–15:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICDT.2021.15.
- 19 Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.
- 20 Artem Lysenko, Irina A. Roznovat, Mansoor Saqi, Alexander Mazein, Christopher J. Rawlings, and Charles Auffray. Representing and querying disease networks using graph databases. *BioData Min.*, 9:23, 2016. doi:10.1186/s13040-016-0102-8.
- 21 Richard T. B. Ma, Dah-Ming Chiu, John Chi-Shing Lui, Vishal Misra, and Dan Rubenstein. Internet economics: The use of Shapley value for ISP settlement. *IEEE/ACM Trans. Netw.*, 18(3):775–787, 2010. doi:10.1109/TNET.2010.2049205.
- 22 Wim Martens and Tina Trautner. Evaluation and enumeration problems for regular path queries. In *ICDT*, volume 98 of *LIPICs*, pages 19:1–19:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICDT.2018.19.
- 23 Wim Martens and Tina Trautner. Dichotomies for evaluating simple regular path queries. *ACM Trans. Database Syst.*, 44(4):16:1–16:46, 2019. doi:10.1145/3331446.

- 24 Corto Mascle, Christel Baier, Florian Funke, Simon Jantsch, and Stefan Kiefer. Responsibility and verification: Importance value in temporal logics. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14. IEEE, 2021.
- 25 Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The complexity of causality and responsibility for query answers and non-answers. *Proc. VLDB Endow.*, 4(1):34–45, 2010.
- 26 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995. doi:10.1137/S009753979122370X.
- 27 Stefano Moretti, Fioravante Patrone, and Stefano Bonassi. The class of microarray games and the relevance index for genes. *Top*, 15(2):256–280, 2007.
- 28 Ramasuri Narayanam and Yadati Narahari. A Shapley value-based approach to discover influential nodes in social networks. *IEEE Trans Autom. Sci. Eng.*, 8(1):130–147, 2011. doi:10.1109/TASE.2010.2052042.
- 29 Alon Reshef, Benny Kimelfeld, and Ester Livshits. The impact of negation on the complexity of the Shapley value in conjunctive queries. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *PODS*, pages 285–297. ACM, 2020. doi:10.1145/3375395.3387664.
- 30 Gorka Sadowski and Philip Rathle. Fraud detection: Discovering connections with graph databases. *White Paper-Neo Technology-Graphs are Everywhere*, 13, 2014.
- 31 Babak Salimi, Leopoldo E. Bertossi, Dan Suciu, and Guy Van den Broeck. Quantifying causal effects on query answering in databases. In *TaPP*. USENIX Association, 2016.
- 32 Lloyd S Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker, editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton, 1953.
- 33 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.
- 34 Tjeerd van Campen, Herbert Hamers, Bart Husslage, and Roy Lindelauf. A new approximation method for the Shapley value applied to the WTC 9/11 terrorist attack. *Soc. Netw. Anal. Min.*, 8(1):3:1–3:12, 2018. doi:10.1007/s13278-017-0480-z.
- 35 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146. ACM, 1982. doi:10.1145/800070.802186.
- 36 Mihalis Yannakakis. Graph-theoretic methods in database theory. In *PODS*, pages 230–242, 1990. doi:10.1145/298514.298576.
- 37 Ningning Yi, Chunfang Li, Xin Feng, and Minyong Shi. Design and implementation of movie recommender system based on graph database. In *WISA*, pages 132–135. IEEE, 2017. doi:10.1109/WISA.2017.34.
- 38 Byoung-Ha Yoon, Seon-Kyu Kim, and Seon-Young Kim. Use of graph database for the integration of heterogeneous biological data. *Genomics & informatics*, 15(1):19, 2017.
- 39 Bruno Yun, Srdjan Vesic, Madalina Croitoru, and Pierre Bisquert. Inconsistency measures for repair semantics in OBDA. In *IJCAI*, pages 1977–1983. ijcai.org, 2018.