The Complexity of the Shapley Value for Path Queries over Graphs

Majd Khalil
The Complexity of the Shapley Value for Path Queries over Graphs

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Majd Khalil

Submitted to the Senate of the Technion — Israel Institute of Technology
Tevet 5782 Haifa December 2021
This research was carried out under the supervision of Prof. Benny Kimelfeld, in the Faculty of Computer Science.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Benny Kimelfeld, for his consistent willingness to help, guidance, and patience throughout this journey. Your support and creative ideas helped me tremendously in realizing this work and is truly appreciated, it was a great honor to work with you.

I’m grateful for my family and friends for always believing in me and keeping me motivated and confident through tough times. Thank you for the unconditional love and support throughout the years.

The Technion’s funding of this research is hereby acknowledged.
Contents

List of Figures

Abstract 1

1 Introduction 3

2 Preliminaries 7
  2.1 Graphs and Path Queries 7

3 Shapley Value of Edges for Path Queries 11

4 Complexity of Exact Computation 15
  4.1 Main Result 15
  4.2 Hardness 16
  4.3 Tractability 20

5 Complexity of Approximation 23
  5.1 Main Result 23
  5.2 Hardness 24
    5.2.1 Open Problem: Directed Acyclic Graphs 28
  5.3 Tractability 28

6 Shapley Value of Vertices for Path Queries 31
  6.1 Complexity of Exact Computation 31
  6.2 Complexity of Approximation 32

7 Conclusions and Open Problems 33

Hebrew Abstract i
List of Figures

2.1 The graph of the running example. An edge \((v_i, v_j)\) will be denoted by \(e_{ij}\). 8

4.1 An example for the construction in the reduction of the proof of Lemma 4.2.2. 18

4.2 An example for the reduction in Lemma 4.2.3, for a regular expression that accepts the word \(abcde\), source node \(s = v_1\), target node \(t = v_9\). . . 19

5.1 An example for the construction in the reduction of the proof of Lemma 5.2.3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. 25
Abstract

A path query extracts from a labeled graph vertex tuples based on the words that are formed from the paths that connect the vertices. We study the computational complexity of measuring the contribution of edges and vertices to an answer of a path query. We focus on conjunctive regular path queries. To measure the contribution we adopt the traditional Shapley value from cooperative game theory. This value has been recently proposed and studied for relational database queries, machine-learning classifiers, and so on.

We first study edge contribution 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 examine the analogous situation where we wish to determine the contribution of a vertex, rather than an edge, and establish results of a similar nature.
Chapter 1

Introduction

In this thesis, we study the complexity of measuring the contribution of graph components to the result of a query. We focus on path and regular path queries, and the Shapley value as a measure of contribution.

Path queries. Graph databases are crucial for many applications where relationships between data are as equally important as the data itself. With unstructured and connected real-time data piling up by the minute, graph databases offer a more flexible, dynamic and efficient alternative to the classic relational database in many cases. Its usage now spans across many fields including, semantic web [AP11], social networks [Fan12], biological networks [LRS+16, YKK17], data provenance [ABL10], fraud detection [SR14], recommendation engines [YLFS17], and many more.

In their simplest form, graph databases are finite, directed, edge-labeled graphs. Nodes are used to store data entities, and edges to store relationships between entities or attributes. The highly connected nature of such databases raises the need for tools that allow inspecting and analyzing the structure and patterns present in the data, this usually comes in the form of queries that allow users to specify the types of paths in which they are interested. One canonical example of such a tool are regular path queries (RPQs) [CMW87, CGLV99, CM90, Yan90].

RPQs allow the specification of paths using regular expression over the edge labels. When evaluated on a graph, the output is usually a set of source-target pairs of nodes that are connected by a path that conforms to the specified regular expression. This allows users to inspect complex connections in graphs by enabling them to form queries that match arbitrarily long paths. An important generalization of RPQs are conjunctive regular path queries which extend regular path queries semantics by allowing a conjunction of atoms where each atom is an RPQ that should hold between two specified variables.

The simplicity and the power of expressiveness of this type of queries made them an integral part of many querying languages over graphs (GraphLog, Cypher, XPath, SPARQL 1.1), which in turn lead to it being a topic with a vast amount of research. This
includes some natural problems and variations on them [FLS98, MW95, MT18, MT19]:

- The decision problem: Does an RPQ $r$ match a path from $s$ to $t$ in $G$?
- The counting problem: How many paths from $s$ to $t$ match an RPQ $r$?
- The computation problem: Compute the set of paths from $s$ to $t$ that match the RPQ $r$.
- The containment problem: Given two RPQs, is the result set of one contained in the result set of the other.

In addition to the matching problems for CRPQs [CGLV00, DT01], and many more. For example, for RPQs, combined complexity of the decision problem is in PTIME while for CRQs it is NP-complete. Data complexity, however, is NLOGSPACE-complete for both [Bae13]. The containment problem for RPQs is PSPACE-complete, and for CRPQs, it was proved to be EXPSPACE-hard [FLS98, CGLV00].

With the surge in the availability of information, there is a great demand for tools that assist users in understanding their data, and allow them to go deeper and interpret particular observations. Over the last few years there have been several efforts in the Database and Machine Learning communities to serve this cause. In a database context for example, we would like to find the causes of answers or non-answers to a query, which would help in explaining unexpected answers to a query or understanding causal relationships the database.

In this work, we focus on the problem of quantifying the responsibility and contribution of different components in the graph like vertices and edges, to an answer of a CRPQ. This problem was studied in the context of queries on relational databases. Some attempts have been made addressing this, for example Meliou et al. [MGMS10] suggested the quantity of responsibility, which is a value that is inversely related to the number of facts that need to be removed along with the fact we reason about for the query result to go from true to false. A fact with fewer facts needed is more responsible for the result. Causal effect is another alternative measure proposed by Salimi et al. [SBSdB16] This is done by transforming the database into a probabilistic one, where tuples have independent probabilities of $\frac{1}{2}$ of being present; and performing counterfactual interventions on it. The causal effect is then the difference of the expected values of the query being true and false. Lastly, and most relevant to our work, recent work has been done to study the adoption of the Shapley value, a solution concept from game theory, to tackle this task [LBKS21, RKL20, LK21].

**The Shapley value.** The Shapley value [Sha53] is a formula for wealth distribution in a cooperative game that is based on counterfactual interventions. It is shown to be a unique method that satisfies certain good properties that we generally desire. This makes it a useful tool for assigning shares of a jointly acquired reward between
participants, so it has been used in many real-world cases like, sharing profits between ISPs [MCL+10], influence measurement in social network analysis [NN11], determining the most important genes for specific body functions [MPB07], identifying key players in terrorist networks [vCHHL18]. Another interesting adaptation is that in the field of interpretability of machine learning models, SHAP [LL17] is a method to explain individual predictions of a model by computing the relative contribution of each feature to the prediction. It is based on the Shapley values in a cooperative game where features of a data instance act as players and jointly obtain the "prediction". It was also used in database theory for quantifying the responsibility each fact has on the inconsistency of a database [LK21]. Despite its benefits, one major drawback that limits its applicability is the computational time that generally increases exponentially with the number of players. This generally leads to resorting to non-trivial representations that allow efficient computations, or alternatively, approximations.

The Shapley value in [LBKS21] is used to measure the extent to which a database tuple contributes to a query answer. Naturally, several tuples/facts are required for a boolean query to become true, or in the case of a numerical query, several facts might contribute to the value it gets in a given database. To get a numerical sense of this contribution, this is modelled as a cooperative game where the players are the facts in the database and the wealth function is the result of the query.

**Our contribution.** As continuation of the line of work of contribution measures for database queries, we wish to do something similar in nature, in the domain of graph databases, where the queries are CRPQs and the players are the edges of the graph. (We consider the complementary view of *vertices* as players later in this thesis.) As done in previous work, we view the graph as consisting of two types of edges: endogenous edges and exogenous edges. Exogenous edges are static items that we take for granted and do not participate in our counterfactual game, typically determined by external, unconcerned factors, deemed not to be possible causes, while endogenous edges are what we reason about their contribution. The classification into endogenous/exogenous is application-dependent, and may even be chosen by the user at query time.

We investigate the complexity of computing the Shapley values of edges in a graph to the answer of a CRPQ, in a later section we address the changes that need to be made when we reason about the contribution of vertices instead of edges. We are interested in the data complexity of the computational problems we mention. The input to the problem of computing the Shapley value in our context usually consists of two components. The first component is a query \(q\) (e.g., an RPQ or a CRPQ). The second component consists of an input graph \(G\), an answer tuple \(t\) of vertices of \(G\), and an edge \(e\) of which contribution we seek to measure. Here we adopt the conventional yardstick of *data complexity* [Var82], where we consider the query fixed. More formally, each fixed query \(q\) is associated with a distinct computational problem, for which the input consists of only \(G\), \(t\) and \(e\).
For exact computation of the Shapley values, we show that it is generally hard. It is sufficient for the CRPQ to have an atom that is non-redundant and contains a word of length three or more for the computation to be hard. Intuitively, non-redundant atom means that it is interesting and adds information to the CRPQ. In addition, for single-atom CRPQs or RPQs, we identify that for the family of queries where this hardness rule does not apply; a language that only contains words of length two or less; Shapley values can be computed in polynomial time.

Next, we study the complexity of approximation. In our context, we look at a standard notion for approximation, which is FPRAS (i.e., 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 expectation that Shapley defines. 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 multiplicative approximation is intractable. In every other case (assuming no redundant atoms), we show that an additive FPRAS can also be used to obtain a multiplicative FPRAS, due to the gap property that was previously known in the relational model [LK21, RKL20]: if the Shapley value is nonzero, then it must be at least the reciprocal of a polynomial.

For the case of vertices, we show that the situation is very similar to what we have for 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 is also tractable for vertices (yet, for vertices we do not complete a full classification). For approximation, we show that we have an identical dichotomy for when queries admit a multiplicative FPRAS.

**Thesis organization.** The rest of the thesis is organized as follows. In the next chapter we introduce some basic terminology that will be used throughout the thesis. In Chapter 3, we formally define how the Shapley value is applied in our setting for edges in graph databases. In Chapter 4 we study the complexity of computing exact Shapley values for CRPQs, and we consider approximations in the form of FPRAS in Chapter 5. We present the complexity results for the case when measuring contribution of vertices instead of edges in Chapter 6 and outline the changes that should be made to the proofs. Lastly, we summarize our results and discuss directions for future work in Chapter 7.
Chapter 2

Preliminaries

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

2.1 Graphs and Path Queries

We use $\Sigma$ to denote a finite alphabet (i.e., a finite set of symbols) that is used for labeling edges of graphs. A word is a finite sequence of symbols from $\Sigma$. As usual, $\Sigma^*$ denotes the set of all words. A language $L$ is a (finite or infinite) subset of $\Sigma^*$. 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.

For our use, a regular expression is defined as follows: $\emptyset$, $\epsilon$, $\sigma \in \Sigma$ are regular expressions. In addition, given $r$ and $s$ that are regular expressions, $(r \mid s), (r \cdot s), (r^*)$ are also regular expressions. We sometimes omit braces when it is appropriate and we may use $(rs)$ instead of $(r \cdot s)$ for better readability. The language $L(r)$ that $r$ accepts is defined as usual. We allow the use of a special regular expression $\Sigma^*$ that accepts every word. Recall that a deterministic finite automaton (DFA) $A$ is defined as a tuple $(Q, \Sigma, \delta, q_0, F)$, where $Q$ is a finite set of states, $\Sigma$ 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 $w$. The automaton accepts a word $w$ if $\delta^*(w) \in F$. For every regular expression there is a DFA that recognizes it and vice versa.

In this paper, a graph is an edge-labeled directed graph $G = (V, E)$ where $V$ is the finite set of nodes and $E \subseteq V \times \Sigma \times V$ is the set of edges. We will consistently denote by $n$ and $m$ the number of nodes and edges, respectively; that is, $n = |V|$ and $m = |E|$. A path $p$ from node $u$ to node $v$ in $G$ is a sequence $p = (v_0, a_1, v_1)(v_1, a_2, v_2) \ldots (v_{k-1}, a_k, v_k)$ of edges in $G$ such that $u = v_0$ and $v = v_k$. By $|p|$ we denote the length $k$ of $p$, and by $\text{lbl}(p)$ we denote the word $a_1 \cdots a_k$. In addition, we will use $G[E']$ to denote the sub-graph $G' = (V, E')$ of $G$.

Let $L$ be a language. We define a path query to be a query of the form $q := (x, L, y)$, when evaluated on a graph $G$, $q(G)$ return all pairs of instantiations $(s, t)$ for variables
Figure 2.1: The graph of the running example. An edge \((v_i, v_j)\) will be denoted by \(e_{ij}\).

\((x, y)\), such that \(s\) and \(t\) are nodes in graph \(G\) and there exists a path \(p\) from \(s\) to \(t\) such that \(\text{lbl}(p) \in L\). For convenience, we may view \(q\) as a function that also takes as input a pair of nodes, such that \(q(G, s, t) = 1\) if \((G, s, t)\) is a “yes” instance and \(q(G, s, t) = 0\) otherwise. We define similarly the special case of the queries, which we call regular path queries (RPQs), where \(L\) is a regular language that is defined via a regular expression \(r\) or an automaton \(A\). We sometimes use the shorthand \(L\) for the query \((x, L, y)\), or \(r\) for the case of a regular expression.

Example 2.1.1. Figure 2.1 depicts the graph \(G\) of our running examples with labels in \(\Sigma = \{a, b, c\}\). We will show a few examples of path queries on the graph \(G\).

- \(q_1 = \Sigma^*\). This query basically tests if there exists any path from \(s\) to \(t\) in \(G\). For example, we have that \(q_1(G, v_1, v_2) = 1\), and \(q_1(G, v_1, v_6) = 1\), as there are paths from \(v_1\) to both \(v_2\) and \(v_6\). But \(q_1(G, v_3, v_1) = 0\), as there is no path from \(v_3\) to \(v_1\).

- \(q_2 = \{abc\}\). This query tests if there is a path from \(s\) to \(t\) in \(G\) that matches the word \(abc\). For example, we have that \(q_2(G, v_1, v_6) = 1\), as there is a path \(v_1 \rightarrow v_3 \rightarrow v_5 \rightarrow v_6\) that matches \(abc\). But we have that \(q_2(G, v_3, v_5) = 0\), as the only path from \(v_3\) to \(v_5\) consists of a single edge with label \(b\).

- \(q_3 = ab^*\). This query tests if there is a path from \(s\) to \(t\) in \(G\) that matches regular expression \(ab^*\). For example, we have that \(q_3(G, v_1, v_6) = 1\), as there are paths: \(v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_6\), or alternatively, \(v_1 \rightarrow v_2 \rightarrow v_6\), that match \(ab^*\). But we have that \(q_2(G, v_3, v_5) = 0\), as the only path from \(v_3\) to \(v_5\) consists of a single edge with label \(b\) which does not match the regular expression.

Conjunctive Regular Path Queries

A conjunctive regular path query or CRPQ for short, is a query with \(k\) variables \(x_1, \ldots, x_k\) that is a conjunction of atomic regular path queries between pairs of nodes.
that can be assigned to \( x_1, \ldots, x_k \). A general CRPQ \( q \) is in the form:

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

(2.1)

Where \( y_i \) and \( z_i \) are variables from \( \{ x_1, \ldots, x_k \} \) and \( r_i \) is a regular expression. The RPQ \((y_i, r_i, z_i)\) is referred to as the atom \( i \) of \( q \) and is denoted by \( q_i \). As before, when evaluated on a graph \( G \), we denote by \( q(G) \) all of the assignments \((u_1, \ldots, u_k)\) for variables \((x_1, \ldots, x_k)\), such that all atomic RPQs are true. We apply an abuse of notation similar to the case of RPQs, such that for a particular assignment \( u_1, \ldots, u_k \), \( q[u_1, \ldots, u_k](G) = 1 \) if it is a “yes” instance and \( q[u_1, \ldots, u_k](G) = 0 \) otherwise.

**Example 2.1.2.** Let us look at \( q[x_1, x_2, x_3] = (x_1, a^*, x_2) \land (x_2, b^*, x_3) \). This is a CRPQ according to our definition and when evaluated on a graph, it returns triplets \((u_1, u_2, u_3)\) such that there is a path from \( u_1 \) to \( u_2 \) that matches \( a^* \) and a path from \( u_2 \) to \( u_3 \) that matches \( b^* \).

In our example graph, 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 \( b^* \). But we have that \( q[v_1, v_3, v_6](G) = 0 \), as the only path from \( v_3 \) to \( v_6 \) has label \( bc \) which does not match \( b^* \).

We say that an atom is redundant if removing it from \( q \) does not change the answers of \( q \) on all graphs. Formally, we denote by \( q \setminus j \) the query after removing atom \( j \).

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

Then atom \( j \) is redundant if: \( q \equiv q \setminus j \), or in other words: \( \forall G : q(G) = q \setminus j(G) \).

**Example 2.1.3.** Let us look at \( q[x_1, x_2, x_3] = (x_1, a, x_2) \land (x_2, b, x_3) \land (x_1, a^*b^*, v_3) \). 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, if one of them does not then that means that the CRPQ does not return true, in that sense, the third atom is redundant and does not add information.

**Observation 2.1.4.** Let \( q \) be a CRPQ. If atom \( i \) is non-redundant, then there exists graph \( G_i \) and assignment \((v_1, \ldots, v_k)\) such that there are matching paths for each \( q_j \) for \( j \neq i \) \( (q_j(G_i, s_j, t_j) = 1) \) and no matching path for \( q_i \) \( (q_i(G_i, s_i, t_i) = 0) \).

**Proof.** By definition, if atom \( i \) is non-redundant, it means that there exists a graph \( G_i \) such that \( q(G_i) \neq q \setminus i(G_i) \). And since every answer tuple \( t \in q(G_i) \) is also an answer tuple for \( q \setminus i(G_i) \), this means that there exists an answer tuple \( t' = (v_1, \ldots, v_k) \in q \setminus i(G_i) \) such that \( t' \notin q(G_i) \). In other words, for assignment \((v_1, \ldots, v_k)\), there are matching paths for each \( q_j \) for \( j \neq i \) \( (q_j(G_i, s_j, t_j) = 1) \) and no matching path for \( q_i \) \( (q_i(G_i, s_i, t_i) = 0) \).
In the sequel, we say that \( q \) is \textit{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.

\section*{Shapley Value}

Let \( A \) be a finite set of players. A cooperative game is a function \( v : P(A) \rightarrow \mathbb{R} \) such that \( v(\emptyset) = 0 \). 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:

\[
\text{Shapley}(A, v, a) = \frac{1}{|A|!} \sum_{\sigma \in \Pi_A} (v(\sigma_a \cup \{a\}) - v(\sigma_a))
\]

where \( \Pi_A \) is the set of all possible permutations over the players in \( A \), and for each permutation \( \sigma \) we denote by \( \sigma_a \) 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 without replacement. The Shapley value is known to be unique up to some rationality axioms [Sha53].
Chapter 3

Shapley Value of Edges for Path Queries

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

Given a conjunctive regular path query $q$, our goal is to quantify the contribution of edges in the input graph $G$ to an answer of the path query. 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) [SBSdB16, LBKS21, MGMS10, LBKS21]. Hence, in our setup we view the graph as consisting of two types of edges—endogenous edges and exogenous edges. Notationally, 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 $\vec{u} = (u_1, \ldots, u_k)$ of a conjunctive regular path query, 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 $\text{Shapley}(q)(G, \vec{u}, e)$.

$$\text{Shapley}(q)(G, \vec{u}, e) \overset{\text{def}}{=} \text{Shapley}(E_n, v_{pq}, e)$$

Where the Shapley function is as defined in Equation (2.2) and $v_{pq}$ is the numerical function that takes as input a subset of the endogenous edges and is defined as follows:

$$v_{pq}(B) \overset{\text{def}}{=} q[\vec{u}](G[B \cup E_x]) - q[\vec{u}](G[E_x])$$
In other words, we have the following.

\[
\text{Shapley}(q)(G, \vec{u}, e) = \sum_{B \subseteq E_n \setminus \{e\}} \frac{|B|!(|E_n| - |B| - 1)!}{|E_n|!} (q[\vec{u}](G[B \cup E_x \cup \{e\}]) - q[\vec{u}](G[B \cup E_x]))
\]

Given a CRPQ \(q\), the computational problem CRPQShapley\(q\) is defined as follows:

<table>
<thead>
<tr>
<th>Problem CRPQShapley(q)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
</tr>
<tr>
<td><strong>Goal:</strong></td>
</tr>
</tbody>
</table>

When \(q\) has only one atom, which is basically an RPQ, we may denote the computational problem by RPQShapley\(r\), and the value computed by \(\text{Shapley}(r)(G, s, t, e)\). Where \(r\) is the only regular expression in the query and it defines the computational problem.

**Example 3.0.1.** We will now show a few examples for the computation of the Shapley values for different inputs. We refer to the graph \(G\) of Figure 2.1. By default, all edges are endogenous unless stated otherwise. (Note that we denote by \(e_{ij}\) the edge \((v_i, v_j)\).)

- **Shapley\((abc)\)(G, v₁, v₆, 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. This is true since adding such an edge to \(G[B \cup E_x]\), where \(B \subseteq E_n\), will not change the result of the path query. 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:

\[
\text{Shapley}(abc)(G, v_1, v_6, e_{13}) = \text{Shapley}(abc)(G, v_1, v_6, e_{35}) = \text{Shapley}(abc)(G, v_1, v_6, e_{56}) = \frac{1}{3}.
\]

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

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

- **Shapley\((ab^*)\)(G, v₁, v₆, e)\). This is a more complicated computation, since there are two paths that match the regular expression \(ab^*\) (as we have seen in Example 2.1.1). Again, any edge that is not on any of these paths will have a Shapley value of 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. Or 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,

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

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,

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

For the last edge $e_{12}$, it needs to appear after $e_{26}$ or after both $e_{24}$ and $e_{46}$. Permutations where this happens are either permutations where $e_{12}$ appears after both $e_{24}$ and $e_{46}$ but before $e_{26}$, after both $e_{26}$ and $e_{46}$ but before $e_{24}$, and there are $3 \cdot \sum_{i=0}^{5} (i+2)!\binom{5}{i}(8-i-2)! = \frac{3}{12} \cdot 9!$ such permutations. Or permutations where $e_{12}$ appears after all three of $e_{26}, e_{24}, e_{46}$, and there are $\sum_{i=0}^{5} (i+3)!\binom{5}{i}(8-i-3)! = \frac{3}{12} \cdot 9!$ such permutations. Or permutations where $e_{12}$ appears after $e_{26}$ 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. So overall we get that,

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

Note that that $\sum_{e \in E_n} \text{Shapley}(ab^*)(G, v_1, v_6, e) = 1$, which is expected since the sum of the Shapley values of all players is always equal to the value of the game for the whole set of players [Sha53].
Chapter 4

Complexity of Exact Computation

In this section, we study the complexity of 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, and hence, every \(q\) defines a separate computational problem CRPQShapley\(\langle q \rangle\).

4.1 Main Result

Our main result for this section is the following theorems, showing that the problem is computationally intractable for almost every CRPQ \(q\), except for very limited cases.

**Theorem 4.1** (Hardness). Let \(q\) be a CRPQ. If \(q\) has a non-redundant atom \(i\) with a language that contains a word of length three or more, then CRPQShapley\(\langle q \rangle\) is FP\(^{\#}\text{P}\)-complete.

Recall that FP\(^{\#}\text{P}\) is the class of functions computable in polynomial time with an oracle to a problem in \#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 [Tod91]).

We do not know whether the condition of Theorem 4.1 is necessary for hardness. However, we can show that it is, indeed, necessary, in the case of a single atom as shown in the following theorem.

**Theorem 4.2** (Tractability). Let \(q\) be an RPQ with the regular expression \(r\). If every word in \(L(r)\) is of length at most two, then RPQShapley\(\langle q \rangle\) is solvable in polynomial time.

We conclude the following corollary that gives a full classification in the case of an atomic regular path query.

**Corollary 4.3.** Let \(q\) be an atomic regular path query with the regular expression \(r\). Assuming P \(\neq\) NP, the following are equivalent:
1. Every word in $L(r)$ is of length at most two.

2. CRPQShapley$(q)$ is solvable in polynomial time.

In Section 4.2 we prove Theorem 4.1, and in Section 4.3 we prove Theorem 4.2.

### 4.2 Hardness

Membership in FP$^{\#P}$ is straightforward from the definition of the Shapley value in Equation (2.2). Indeed, Shapley$(q)(G, \bar{u}, e)$ can be computed using an oracle to the problem of counting the number of permutations over the edge set such that $e$ changes the query from zero (false) to one (true). For the FP$^{\#P}$-hardness, we prove it in a sequence of reductions. We begin with hardness for the special case where the regular language (or any language) consists of a single three-letter word. For that, we will use a result by Livshits et al. [LBKS21] on the computation of Shapley values for facts in relational databases. Then we use that to prove hardness for the general case of a regular language (or any language) which has at least one word of length at least three, even when restricted to a special kind of graphs.

We first recall the result of Livshits et al. [LBKS21]. They considered relational databases $D$ where some of the facts are endogenous and the rest exogenous. 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:

$$\text{Shapley}(q)(D, f) = \text{Shapley}(D_n, v_{db}, f)$$

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

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

In addition, we define a special kind of graphs that will help us in some of the proofs.

**Definition 4.2.1 (leveled graph).** A graph $G = (V, E)$ is called a leveled graph if there exists a split of vertices into levels $V_0, \ldots, V_k$, such that:

1. The set of vertices $V$ is the disjoint union of $V_0, \ldots, V_k$.

2. Edges are only between vertices in consecutive levels.

Using that, we prove the following.
Lemma 4.2.2. Let $\sigma_i \in \Sigma$ for $i = 1, 2, 3$. $\text{RPQShapley}(\sigma_1\sigma_2\sigma_3)$ is FP$^p$-hard, even when restricted to leveled graphs.

Proof. The proof is by reduction from the problem of computing $\text{Shapley}(Q_{RST})(D, f)$, where $D$ is a database over scheme $S = \{R(x), S(x, y), T(x)\}$. Since, as mentioned earlier, this problem is FP$^p$-hard. Given a database $D$ over $S$, we will construct a leveled graph database $G = (V, E)$ over $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$. We will map facts to edges using the following one-to-one mapping function (bijection), the matching edge for fact $f$ will be denoted by $e_f$: For each fact $R(x)$ we will add an edge $(s_0, \sigma_1, x_1)$. For each fact $S(x, y)$ in we will add an edge $(x_1, \sigma_2, y_2)$. For each fact $T(y)$, we will add an edge $(y_2, \sigma_3, t_3)$. It can be seen that this graph is a three-level graph. It is constructed such that if the query is satisfied in database $D$, then there is a path from $s_0$ to $t_3$ matching $\sigma_1\sigma_2\sigma_3$. An edge $e_f$ is exogenous/endogenous according to the classification of the fact $f$. To complete the proof, we will show that:

\[
\text{Shapley}(q_{RST})(D, f) = \text{Shapley}(\sigma_1\sigma_2\sigma_3)(G, s_0, t_3, e_f)
\]

We will prove this by showing that both games are isomorphic. As stated before, the mapping function is a bijection and there is a one-to-one mapping between facts and edges, now we will show that the wealth functions get the same values for each subset $F$ of facts and its mapping to edges $E$ in the reduction graph $G$, i.e., $v_{\text{db}}(F) = v_{pq}(E)$.

We denote by $q_{\sigma_1\sigma_2\sigma_3}$ the RPQ with regular expression $\sigma_1\sigma_2\sigma_3$.

Since $v_{\text{db}}(F) = Q_{RST}(F \cup D_x) - Q_{RST}(D_x)$ and $v_{pq}(E) = q_{\sigma_1\sigma_2\sigma_3}(G[E \cup E_x], s_0, t_3) - q_{\sigma_1\sigma_2\sigma_3}(G[E_x], s_0, t_3)$, it is sufficient to show that $Q_{RST}(F') = q_{\sigma_1\sigma_2\sigma_3}(G[E'], s_0, t_3)$ for a subset $F'$ of facts and its mapping to edges $E'$.

\[
Q_{RST}(F') = 1 \iff F' \models R(x), S(x, y), T(y) \tag{1}
\]

\[
\iff \exists a, b : R(a), S(a, b), T(b) \in F' \tag{2}
\]

\[
\iff (s_0, \sigma_1, a_1), (a_1, \sigma_2, b_2), (b_2, \sigma_3, t_3) \in E' \tag{3}
\]

\[
\iff q_{\sigma_1\sigma_2\sigma_3}(G[E'], s_0, t_3) = 1 \tag{4}
\]

Transitions (1) and (2) follow from definitions, transition (3) follows from the construction of the reduction graph $G$, and transition (4) follows from that if $(s_0, \sigma_1, x_1), (x_1, \sigma_2, y_2), (y_2, \sigma_3, t_3) \in E'$ then these edges connect a path from $s_0$ to $t_3$ matching $\sigma_1\sigma_2\sigma_3$.

So as a conclusion, for any fact $f$ we can compute its Shapley value efficiently, in polynomial time, given a solution for the second problem. The construction of the graph can be done in polynomial time. ■
Lemma 4.2.3. Let $r$ be a regular expression. If there exists a word in $L(r)$ of length at least three, then $\text{RPQShapley}(r)$ is $\text{FP}^\#P$-hard, even when restricted to leveled graphs.

Proof. Let $w_0 = \sigma_1 \ldots \sigma_n \in L(r)$, such that $n = |w_0| \geq 3$. We will show a reduction from the problem $\text{RPQShapley}(\sigma_1 \sigma_2 \sigma_3)$ on leveled graphs to $\text{RPQShapley}(r)$. Given an input graph $G$, source node $s$, target node $t$ and edge $e$ for $\text{RPQShapley}(\sigma_1 \sigma_2 \sigma_3)$, we construct an input graph $G'$ for $\text{RPQShapley}(r)$. It will be constructed in the following way:

Starting from $s$, we will keep edges in the first level with label $\sigma_1$, edges in the second level with label $\sigma_2$, and edges in the third level with label $\sigma_3$. These edges will be classified according to their original classification. All other edges will be removed. If $|w| > 3$, we will also add a path consisting of exogenous edges, from node $t$ to a new node $t'$ matching the remainder of $w$, i.e., $\sigma_4 \ldots \sigma_n$. Such a reduction is illustrated in Figure 4.2. Now we will show that $\text{Shapley}(\sigma_1 \sigma_2 \sigma_3)(G, s, t, e) = \text{Shapley}(r)(G', s, t', e)$.

Now we can see that in the new graph $G'$ there can only be paths matching $w_0 = \sigma_1 \ldots \sigma_n$ from $s$ to $t'$; starting with a sub-path that was originally in $G$ that matches $\sigma_1 \sigma_2 \sigma_3$, and getting to $t'$ through the only path from $t$ to $t'$ that we added. In addition, any edge that was removed $e'$ has $\text{Shapley}(\sigma_1 \sigma_2 \sigma_3)(G, s, t, e') = 0$, since it does not participate in any path from $s$ to $t$ matching $\sigma_1 \sigma_2 \sigma_3$, otherwise it would have been added to $G'$ by construction. So adding such an edge to $G[B \cup E_x]$, where $B \subseteq E_n$, will not change the result of the path query.

Since the sub-path that matches the suffix $\sigma_4 \ldots \sigma_n$ of $w_0$, consists only of exogenous edges and can only be from $t$ to $t'$, the path query reduces to a path from $s$ to $t$. 

Figure 4.1: An example for the construction in the reduction of the proof of Lemma 4.2.2.
matching $\sigma_1\sigma_2\sigma_3$ instead of a path from $s$ to $t'$ matching $r$. Pairing that with the fact the in both graphs we have the same set of endogenous edges, except for endogenous edges in the original graph that were not added to $G'$, those edges can not be in a path from $s$ to $t$ matching $\sigma_1\sigma_2\sigma_3$ by construction, so they have a Shapley value of 0 and can be removed from the game without affecting values of the others, we get that $\text{Shapley}^\langle\sigma_1\sigma_2\sigma_3\rangle(G, s, t, e) = \text{Shapley}^\langle r\rangle(G', s, t', e)$.

Proof of Theorem 4.1. We know that there exists an atom $i$ of $q$ such that is non-redundant and $L(r_i)$ contains a word of length at least three. We will show a reduction from the problem $\text{RPQShapley}^\langle r_i \rangle$ on leveled graphs to $\text{CRPQShapley}^\langle q \rangle$ which proves the theorem.

Given an input graph $G$, source node $s$, target node $t$ and edge $e$ for $\text{RPQShapley}^\langle r_i \rangle$, we show how to construct an input instance $G^*$ for $\text{CRPQShapley}^\langle q \rangle$. Since atom $i$ is non-redundant, we can use Observation 2.1.4 and conclude that there exists a graph $G_i$ and assignment $\vec{v}$ to $\vec{x}$ such that all RPQ atoms return true but the $i$th atom, i.e, $q_j(G_i, s_j, t_j) = 1$ for every $j \neq i$ and $q_i(G_i, s_i, t_i) = 0$. Here, $s_j$ and $t_j$ are the nodes assigned to the variables $y_j$ and $z_j$, respectively, from Equation (2.1).

To construct $G^*$, we will have $G_i$ with all edges exogenous, and we will connect to it the graph $G$ by connecting source node $s$ and target node $t$ to nodes $s_i$ and $t_i$, respectively. Connecting as in both are the same node in the graph. Edges in $G$ will be classified according to their original classification.

Since in $G_i$ for every $j \neq i$ there is a matching path for every atom RPQ $j$,
q[v_1, \ldots, v_k](G^*) already reduces to q_i(G^*, s_i, t_i). We know there are no paths from s_i to t_i matching r_i in G_i, pairing that with the fact that s_i and t_i are not part of a cycle, we get that the such paths can only be in G. Since also, s \equiv s_i and t \equiv t_i, we get that the query reduces to q_i(G, s, t). Since we also have the same set of endogenous edges in G and G^*, we get that:

\[
\text{Shapley}(r_i)(G, s, t, e) = \text{Shapley}(q_i)(G^*, \vec{v}, e)
\]

From Lemma 4.2.3, we know that \(\text{RPQShapley}(r_i)\) is FP\#P-hard so we get that \(\text{CRPQShapley}(q_i)\) is FP\#P-hard. This completes the proof of the hardness side of the proof. Next, we show the tractability side.

### 4.3 Tractability

We now present a polynomial-time algorithm for computing \(\text{RPQShapley}(r)\) where \(L = L(r)\) consists of words of length at most two. We denote by \(\mathcal{M}(G, s, t, L, k)\) the set of all subsets \(E'\) of \(E_0\) such that \(G[E_0 \cup E']\) contains a path of \(L\) from \(s\) to \(t\). If we group subsets of edges of the same size, we can also get the following form for the Shapley value:

\[
\text{RPQShapley}(r)(G, s, t, e) = \sum_{k=0}^{|E_n|-1} \frac{k!([|E_n| - k - 1]!)}{|E_n|!} \cdot |\mathcal{M}(G, s, t, L(r), k)|

- \sum_{k=0}^{|E_n|-1} \frac{k!([|E_n| - k - 1]!)}{|E_n|!} \cdot |\mathcal{M}(G \setminus \{e\}, s, t, L(r), k)|
\]

This shows that the problem of computing \(\text{RPQShapley}(r)(G, s, t, e)\) can be reduced to computing \(|\mathcal{M}(G, s, t, L(r), k)|\); the number of subsets of \(E_n\) (endogenous edges) of size \(k\) which when added to \(E_x\), connects a path from \(s\) to \(t\) that matches a word \(w \in L(r)\). We will make use this form later in the analysis.

**Proof of Theorem 4.2.** As mentioned before, the problem of computing Shapley values in our setting reduces to computing \(|\mathcal{M}(G, s, t, L, k)|\). We will now show that \(|\mathcal{M}(G, s, t, L, k)|\) can be computed in polynomial time in this case. First, let us observe that we can compute \(|\mathcal{M}(G, s, t, L, k)|\) by computing the complement set \(|\overline{\mathcal{M}(G, s, t, L, k)}|\) which is defined similarly but for subsets where there is no path, and subtracting from the overall number of subsets of size \(k\) of endogenous edges:

\[
|\mathcal{M}(G, s, t, L, k)| = \binom{m_n}{k} - |\overline{\mathcal{M}(G, s, t, L, k)}|
\]

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

Let \(L = \{w_0, \ldots, w_l\}\). For a subset of endogenous edges to be in \(\overline{\mathcal{M}(G, s, t, L, k)}\), it should not connect, with \(E_x\), any path from \(s\) to \(t\) matching \(w \in L\), i.e., matching one
of \(w_0, \ldots, w_l\). In other words, it should not connect any path of size 2 matching some \(|w_i| = 2\), or any path of size 1 matching some \(|w_i| = 1\).

This divides endogenous edges of the graph into 4 categories:

- **Permitted.** Edges which are not part of any such path.
- **OnPath2E.** Edges which are on a path of size 2 matching some \(w_i\), where both edges are endogenous.
- **OnPath2X.** Edges which are on a path of size 2 matching some \(w_i\), where one of the edges is exogenous.
- **OnPath1.** Edges which are on a path of size 1 matching some \(w_i\).

These are disjoint paths, as we assume that there are no parallel edges, and edges on a path of size 1 can not also be on paths of size 2 and vice versa.

Each subset of size \(k\) in \(\mathcal{M}(G, s, t, L, k)\) has no edges from OnPath1 as such edges make a path, and no edges from OnPath2X as such edges connect a path along with \(E_x\), \(i\) edges from Permitted, and \(k-i\) edges from OnPath2 such that no whole path is added, meaning a maximum of one edge is added from each path of size 2. For each \(i\), this translates to \(\binom{|\text{Permitted}|}{i}\) options for edges from Permitted, and \(2\binom{|\text{OnPath2}|}{k-i}\) options for edges from OnPath2; choosing \(k-i\) paths from \(\frac{|\text{OnPath2}|}{2}\) paths, and for each path there are 2 possibilities for edges. All in all we get that:

\[
|\mathcal{M}(G, s, t, L, k)| = \sum_{i=0}^{k} 2^{\binom{|\text{Permitted}|}{i}} \binom{|\text{OnPath2}|}{k-i}
\]

Which can be computed in polynomial time.
Chapter 5

Complexity of Approximation

In this section, we study the complexity of approximating CRPQ\text{Shapley}⟨q⟩. Our goal is a fully polynomial randomized approximation scheme, known as an FPRAS. Formally, an FPRAS for a numeric function \( f \) is a randomized algorithm \( A(x, \epsilon, \delta) \), where \( x \) is an input for \( f \) and \( \epsilon, \delta \in (0, 1) \), that returns an \( \epsilon \)-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:

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

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 Main Result

Our main result for this section is a simple Monte-Carlo based algorithm that guarantees an additive approximation for any CRPQ, that also serves as a multiplicative FPRAS in some cases that we present in a dichotomy for when a given CRPQ admits a multiplicative approximation. 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 2.1).

**Theorem 5.1.** Let \( q \) be a CRPQ without redundancy. If \( L(r_i) \) is finite for every atom \( i \) of \( q \), then CRPQ\text{Shapley}⟨q⟩ has a multiplicative FPRAS. Otherwise, CRPQ\text{Shapley}⟨q⟩ has no multiplicative approximation (of any ratio) or else \( NP \subseteq BPP \).
5.2 Hardness

Lemma 5.2.1. The decision problem that takes as input, a graph \( G \), source node \( s \), target node \( t \), and edge \( e \), and decides whether \( e \) lies on a simple path from \( s \) to \( t \) in graph \( G \), is \( \text{NP}\)-complete.

Proof. In [FHW80], the set of pattern graphs for which the fixed directed sub-graph homeomorphism problem is \( \text{NP}\)-complete is characterized. An immediate result of that is that the decision problem that takes as input, a graph \( G \), source node \( s \), target node \( t \), and node \( v \), and decides whether \( v \) lies on a simple path from \( s \) to \( t \) in graph \( G \), is \( \text{NP}\)-complete. We will show a simple reduction from the problem of determining whether a node lies on a simple path to the problem of whether an edge lies on a simple path. Given a graph \( G \), source node \( s \), target node \( t \), and node \( v \), we modify \( G \) such that node \( v \) is split into two nodes \( v_{\text{in}} \) and \( v_{\text{out}} \) and we connect them by an edge \((v_{\text{in}}, v_{\text{out}})\).

All ingoing edges into \( v \) will be added as ingoing edges to \( v_{\text{in}} \), and all outgoing edges from \( v \) will be added as outgoing edges from \( v_{\text{out}} \), the new graph will be denote by \( G' \).

We now argue that \( v \) lies on a simple path from \( s \) to \( t \) in \( G \) if and only if edge \((v_{\text{in}}, v_{\text{out}})\) lies on a simple path from \( s \) to \( t \) in \( G' \), which completes the reduction. \[\square\]

Lemma 5.2.2. Let \( G \) be a graph, \( s \) and \( t \) two vertices of \( G \), and \( e \) an endogenous edge of \( G \). Shapley\( (\Sigma^*) (G, s, t, e) > 0 \) if and only if \( e \) belongs to a simple path from \( s \) to \( t \).

Proof. We handle each direction separately.

\(\Leftarrow\Rightarrow\): If Shapley\( (\Sigma^*) (G, s, t, e) > 0 \) then there exists some subset of edges \( S \) that adding \( e = (x, y) \) to it connects some path from \( s \) to \( t \), otherwise the marginal contribution of \( e \) to all subsets of edges is zero and we get that Shapley\( (\Sigma^*) (G, s, t, e) = 0 \). We argue that adding \( e \) connects at least one path from \( s \) to \( t \) that is simple. Since adding \( e \) to the subset of edges \( S \) connects a path from \( s \) to \( t \), then there already exist two sub-paths, \( l_1 \) from \( s \) to \( x \), and \( l_2 \) from \( y \) to \( t \) with all edges in \( S \). If \( l_1 \) is not simple, we can get a simple path \( l'_1 \) by removing cycles from \( l_1 \), same applies for \( l_2 \). The path that combines \( l'_1, e, l'_2 \) is a simple path from \( s \) to \( t \), since \( l'_1 \) and \( l'_2 \) are simple, in addition, let us assume that the path visits some vertex in \( l'_2 \) that it already visited in \( l'_1 \), then in contradiction to that \( e \) has non-zero marginal contribution to \( S \), we can get rid of the cycle that we have, and get path with all edges in \( S \), meaning that \( e \) was not needed to connect such path.

\(\Rightarrow\Rightarrow\): \( e \) lies on a simple path \( l \) from \( s \) to \( t \) in \( G \). If we look at \( L \) the set of edges in path \( l \) not including \( e \) then adding \( e \) to that subset of edges connects a path from \( s \) to \( t \), that path matches some word \( w \in \Sigma^* \). So the marginal contribution of \( e \) to that subset is \( 1 \) and that means that Shapley\( (\Sigma^*) (G, s, t, e) > 0 \).

Corollary 5.2. The problem of determining whether Shapley\( (\Sigma^*) (G, s, t, e) > 0 \) is \( \text{NP}\)-complete.
Lemma 5.2.3. Let $r$ be a regular expression. If $L(r)$ is infinite, then the problem of determining whether $\text{Shapley}(r)(G, s, t, e) > 0$ is NP-complete.

Proof. It is straight-forward to show that the problem is in NP for a start, 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 $\text{Shapley}(\Sigma^*)(G, s, t, e) > 0$ which we showed to be NP-complete.

Given an input instance $(G, s, t, a)$, we will show how to construct an instance $(G', s', t', a_k)$ for our problem such that:

$$\text{Shapley}(\Sigma^*)(G, s, t, a) > 0 \iff \text{Shapley}(r)(G', s', t', a_k) > 0$$

Since $L(r)$ is infinite, we know that its corresponding DFA graph that we will denote by $G_{\text{DFA}}$ has at least one cycle. We find a path from an initial state to an accepting state that passes through a node $v_i$ that is part of a cycle. We will denote the path by:
\( l : v_0 \rightarrow \ldots \rightarrow v_i \rightarrow \ldots \rightarrow v_k. \)

We assumed the node \( v_i \) is part of a cycle, we denote the labels which are along the cycle starting from \( v_i \) by \( w_{\text{cycle}} = \sigma_0 \ldots \sigma_c. \) The graph \( G' \) will be constructed so that it any path in it from \( s' \) to \( t' \) matches \( r \) in the following way, containing 3 sub-graphs:

- The path \( s' = v_0 \rightarrow \ldots \rightarrow v_{i}^{\text{in}} = s, \) with the same labels as in the DFA, the edge \((v_{i-1}, v_{i}^{\text{in}})\) will have the label of \((v_{i-1}, v_i)\). (exogenous).

- A copy of the graph \( G \) where each edge \( e \) is split into \( c \) edges with labels matching \( w_{\text{cycle}} \) which we will denote by \( e_1 \ldots e_c. \) The node \( s = v_i^{\text{in}} \) will serve as the source \( s \) in the original graph, and the node \( t = v_i^{\text{out}} \) will serve as the target \( t \) in the original graph. (endogenous/exogenous according to original edge).

- The path \( t = v_i^{\text{out}} \rightarrow \ldots \rightarrow v_k, \) with the same labels as in the DFA, the edge \((v_i^{\text{out}}, v_{i+1})\) will have the label of \((v_i, v_{i+1})\). (exogenous).

We will now show that for each \( a \in E \) and \( a_k \in E' \) that is any edge that sits on the path that replaced the edge \( a \) in \( G' \) this holds:

\[
\text{Shapley}(\Sigma^*)(G, s, t, a) > 0 \iff \text{Shapley}(r)(G', s', t', a_k) > 0
\]

\[\implies: \]

\[
\text{Shapley}(\Sigma^*)(G, s, t, a) > 0 \implies \exists S \subset E_n : q_{\Sigma^*}(G[S \cup E_\chi \cup \{a\}], s, t) - q_{\Sigma^*}(G[S \cup E_\chi], s, t) > 0 \implies \text{Adding } a \text{ to } S \cup E_\chi \text{ connects a path from } s \text{ to } t \text{ in } G \]

Adding \( a_k \) to \( S' \cup E_\chi' \) connects a path from \( s' \) to \( t' \) in \( G' \) that matches \( r, \) where \( S' \) is the subset containing all edges \( e_i \) for each edge \( e \in S \) and edges \( a_i \) for \( i \neq k \)

\[
\exists S' \subset E'_n : q_r(G'[S' \cup E'_\chi \cup \{a_k\}], s', t') - q_r(G'[S' \cup E'_\chi], s', t') > 0 \implies \text{Shapley}(r)(G', s', t', a_k) > 0
\]

Transitions (1), (2), (4), (5), follow from definitions. We need to prove transition (3): if adding \( a \) to \( S \cup E_\chi \) connects a path from \( s \) to \( t \) in \( G, \) then, adding \( a_k \) to \( S' \cup E_\chi' \) connects a path from \( s' \) to \( t' \) in \( G' \) that matches \( r, \) where \( S' \) is the subset containing all edges \( e_i \) for each edge \( e \in S \) and edges \( a_i \) for \( i \neq k. \)

First, we will need to prove that there is no path from \( s' \) to \( t' \) in \( G' \) using only \( S' \cup E_\chi' \) matching \( r. \) Let us assume that there is, then this path contains a sub-path from \( s \) to \( t \) of edges \( e_i, \) for each such edge \( e_i, e \in S \) by the way of reduction, so we get that there is a path from \( s \) to \( t \) in \( G \) using \( S \cup E_\chi \) by contradiction to that \( a \) is needed to connect such path. Now we will prove that adding \( a_k \) connects a path from \( s' \) to \( t' \) matching \( r. \) We know that adding \( a \) to \( S \cup E_\chi \) connects a path from \( s \) to \( t \) in \( G \) so from that we get that adding \( a_k \) to \( S' \cup E_\chi' \) connects a path from \( s \) to \( t \) in \( G' \) that is the same path as in \( G \) but each edge is split into \( c \) edges, the overall path from \( s' \) to \( t' \) will be composed the only path from \( s' \) to \( s \) and then the path we mentioned from \( s \)
to $t$ and finally the path from $t$ to $t'$. We will now show that this path matches $r$ by simulating a run of the DFA for $r$ and showing that it ends in an accepting state.

Starting from the initial state $v_0$, the path from $s'$ to $s$ would transition the automaton into state $v_1$, as this path was constructed according to the transitions of the automaton. Then, each pass through a collection of edges $e_k$ that match an edge $e$ in the original graph keeps the automaton in the same state, this is because going through their labels creates a cycle. Since the path from $s$ to $t$ goes through a finite number of such transitions, we get that simulating the path until $t$ keeps the automaton in state $v_k$. Finally, simulating the last part is similar to the first part, and it would transition the automaton into state $v_k$ which is an accepting state. This means that adding $e_k$ connects a path from $s'$ to $t'$ matching $r$.

\[
\text{Proof.} \quad \text{Lemma 5.2.4.} \quad \text{Let } q \text{ be a CRPQ without redundancy. If there exists an atom } i \text{ of } q \text{ such that } L(r_i) \text{ is infinite, then the problem of determining whether } \text{Shapley}(q)(G, \vec{u}, e) > 0 \text{ is NP-complete.}
\]

\[
\text{Proof.} \quad \text{We know that there exists an atom } i \text{ of } q \text{ such that is non-redundant and } L(r_i) \text{ is infinite. We will show a reduction from the problem of deciding whether } \text{Shapley}(r_i)(G, s, t, e) > 0 \text{ which we showed to be NP-hard and that proves the lemma.}
\]

\[
\text{Given an input graph } G, \text{ source node } s, \text{ target node } t \text{ and edge } a, \text{ the reduction would work the same way as in the proof of Theorem 4.1 and we construct a } G^* \text{ that is a union of the graph } G_i \text{ that we get from that atom } i \text{ is non-redundant and the original}
\]

27
graph $G$. The same claims hold as in the previous proof and we get:

$$\text{Shapley}(r_i)(G, s, t, a) = \text{Shapley}(q)(G^*, \bar{v}, a)$$

That concludes the proof of the reduction as given a solution for the problem of determining whether $\text{Shapley}(q)(G, \bar{u}, e) > 0$ we can solve the problem of determining whether $\text{Shapley}(r_i)(G, s, t, e) > 0$ in polynomial time. ■

5.2.1 Open Problem: Directed Acyclic Graphs

It is worth noting that the proof as shown in this section, does not work when the graph is acyclic as it relies on Lemma 5.2.1 as a basis. Which states that the decision problem that takes as input, a graph $G$, source node $s$, target node $t$, and edge $e$, and decides whether $e$ lies on a simple path from $s$ to $t$ in graph $G$, is NP-complete. While this is true in the case of a general graph $G$, it is not when the graph is acyclic, where the problem can be solved in polynomial time. This leaves the problem of whether there is a different dichotomy when restricted to DAGs open. However, for exact computation there is no change even when restricted to DAGs as the proofs work as is.

5.3 Tractability

We will now show that for any query that the condition for hardness does not hold, a multiplicative FPRAS can be obtained. We will start by showing that in this case, the gap property holds: if the Shapley value is nonzero, then it must be at least the reciprocal of a polynomial.

Claim 5.3.1. Let $q$ be a fixed CRPQ without redundancy. If for each atom $i$ of $q$: $L(r_i)$ is finite, then $\text{Shapley}(q)(G, \bar{u}, e)$ is either zero or at least $1/p(|E|)$.

Proof. If there is no subset $S$ of $E_n$ such that adding $e$ to it along with $E_x$ changes the value of query $q$ from false to true, then $\text{Shapley}(q)(G, \bar{u}, e) = 0$. Otherwise, let $S$ be a minimal such set, it holds that $|S| \leq k_1 + \ldots + k_m = k$, where $k_i$ 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. And since each $L(r_i)$ is finite, each $k_i$ is a finite constant. Thus, $k$ also is a finite constant.

The probability to choose a permutation $\sigma$, such that $\sigma(e)$ is exactly $S \setminus \{e\}$ is

$$\frac{1}{m_n!} \prod_{i=1}^{m_n} \left(1 - \frac{k_i}{m_n}\right)! \geq \frac{(m_n - k)!}{m_n!}$$

Hence, we have

$$\text{Shapley}(q)(G, \bar{u}, e) \geq \frac{(m_n - k)!}{m_n!} = \frac{1}{p(|E|)}, \text{ as } |E| = m_n + m_e.$$ ■

Lemma 5.3.2. Let $q$ be a CRPQ without redundancy. If for each atom $i$ of $q$: $L(r_i)$ is finite, then CRPQShapley$(q)$ has both an additive and a multiplicative FPRAS.
Proof. Using the Chernoff-Hoeffding bound, we can get an additive FPRAS of Shapley\(q(G, \vec{u}, e)\), by simply taking the ratio of successes over \(O(\log(1/\delta)/\epsilon^2)\) trials of the following experiment:

- Select a random permutation \((e_1, ..., e_m)\) over the set of endogenous edges \(E_n\).
- Suppose that \(e = e_i\), and let \(E_{i-1} = \{e_1, ..., e_{i-1}\}\). 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, "failure".

Now from Claim 5.3.1 (that the gap property holds), we can easily get that an additive FPRAS also serves as a multiplicative one.

Proof of Theorem 5.1. Lemma 5.2.4 shows the hardness side, as it implies that under conventional complexity assumptions, there is no polynomial-time multiplicative approximation when there is an atom with an infinite language (as it would allow to determine whether the Shapley value is nonzero). Lemma 5.3.2 shows an FPRAS for the tractable case where all atom have finite languages.

\[ \text{Technion - Computer Science Department - M.Sc. Thesis MSC-2022-14 - 2022} \]
Chapter 6

Shapley Value of Vertices for Path Queries

In this section, we discuss the differences between the computation complexity of the Shapley value for edges and the Shapley values for vertices in the graph. Similarly to the case for edges, given a conjunctive regular path query $q$, our goal is to quantify the contribution of vertices in the input graph $G$ to an answer of the path query. The graph consists of two types of vertices—endogenous vertices, and exogenous vertices.

Notationally, 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 $V_x$. We denote by $\text{Shapley}^v(q)(G, \vec{u}, v)$ the Shapley value of a vertex $v \in V_n$.

$$\text{Shapley}^v(q)(G, \vec{u}, v) \overset{\text{def}}{=} \text{Shapley}(V_n, v^\text{vertex}_{pq}, v)$$

Where $v^\text{vertex}_{pq}$ is defined as follows:

$$v^\text{vertex}_{pq}(B) \overset{\text{def}}{=} q[\vec{u}](G[B \cup V_x]) - q[\vec{u}](G[V_x])$$

We denote by $\text{CRPQShapley}^v(q)$ and $\text{RPQShapley}^v(r)$, The corresponding computational problems to those we defined earlier for computing the Shapley values of edges. We now state the results we have with some notes on the changes that should be made in the proofs.

6.1 Complexity of Exact Computation

The hardness part is almost the same as 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 instead of three. For that, we use the same result by Livshits et al. [LBKS21] on the computation of Shapley values for facts in relational databases. From this we continue the same series of reductions as done for edges to get the hardness for a general
CRPQ. The tractable part is also tractable when looking at vertices. So we have the following results:

**Theorem 6.1.** The following hold for a CRPQ $q$.

1. If $q$ has a non-redundant atom $i$ with a language that contains a word of length four or more, then $\text{CRPQShapley}^\text{vertex}(q)$ is $\text{FP}^\#P$-complete.

2. If $q$ has only one atom with regular expression $r$. If every word in $L(r)$ is of length at most two, then $\text{CRPQShapley}^\text{vertex}(q)$ is solvable in polynomial time.

Note that in the case of vertices, we leave a gap in the classification of RPQs. Theorem 6.1 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 4.3).

### 6.2 Complexity of Approximation

For approximation, we get the exact same dichotomy on CRPQs. We know from before that also the decision problem that decides whether a vertex $v$ lies on a simple path from $s$ to $t$ in graph $G$, is $NP$-complete. From that we get that the problem of determining whether $\text{Shapley}(\Sigma^*)(G, s, t, v) > 0$ is also $NP$-complete. From that we continue with a series of reduction that is almost identical to what we have for the case of edges.

**Theorem 6.2.** Let $q$ be a CRPQ without redundancy. If $L(r_i)$ is finite for every atom $i$ of $q$, then $\text{CRPQShapley}^\text{vertex}(q)$ has a multiplicative $\text{FPRAS}$. Otherwise, $\text{CRPQShapley}^\text{vertex}(q)$ has no multiplicative approximation (of any ratio) or else $NP \subseteq BPP$.

In conclusion, we establish that the complexity for both exact computation and approximation of the Shapley value, for the case of vertices, is very similar to the case of edges we investigated. 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 show that we have an identical dichotomy for when queries admit a multiplicative FPRAS.
Chapter 7

Conclusions and Open Problems

In this work, we continued past and ongoing efforts for explainability and responsibility in databases. We presented an instance of the problem in graph databases where the queries are conjunctive regular path queries which are an extension of regular path queries, and the responsibility measure is based on the known Shapley value.

We investigated the data complexity of computing the Shapley values for edges in a graph, for a given CRPQ. For an exact computation of Shapley values, we showed that it is generally hard, while we also show a specific family of CRPQs where the computation can be done in polynomial time. While this is not a full dichotomy on CRPQs, the tractable case we showed basically defines a dichotomy on the class of RPQs. We also believe that the condition we have for hardness should define a dichotomy on CRPQs, but that is yet to be proved. We have also studied approximation of the Shapley values 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, meaning, an additive approximation also serves as a multiplicative one. For the case where the gap property does not hold we showed that it is hard to obtain a multiplicative FPRAS. Thus, we achieved a dichotomy on CRPQs for the case of approximation (assuming no redundant atoms).

This naturally leaves some problems open. We mentioned for example that we still do not have a full dichotomy for exact computation of Shapley values, this is one area that needs to be revisited. In addition, as we mentioned before, the proof for the hardness of approximation in Section 5.2 is not valid when the graph in hand is acyclic, this raises the question of whether there can a more "forgiving" dichotomy for approximations for the problem restricted to graphs that are acyclic. Another direction, is investigating different variations on CRPQs as we defined them. For example, to allow existentially quantified variables in the query, or to allow having negated RPQ atoms, or other extensions of C/RPQs.
Bibliography


לאחר כך, אנו בוחנים את הסיבוכיות של קירוב. אנו צמ {* שמתutherland בקירוב פולינומי מחלקה של קמח קירוב כמקובעת. כך שאילתה הבאה בקירוב בלא sexuales של קמח קירוב, או שה問題 של פונקציות פולינומיות של קמח קירוב, שניהו פונקציות פולינומיות של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב. או שהמרחב של קמח קירוב, או שהמרחב של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משולשים-חלקה של קמח קירוב, ש海运 פונקציות באלגוריתמים משו...
בבעייתה הכלה: בהינתן שבישאהillas מסלול רגולרי, האם אחת מכילות את השניה?

ובנוסף הבעיות המקילות לשבישאותillas מסלול רגולרי משולבות.

בעבודה זו ניאומתמקדבים בבעיה של כמות האחיות והתרומות של רוכבים של שניות بغדי כמות
קשיחות, השיאול ושאילות מסלול רגולרי משולב. בהיות זה חקירה בדיקה של שיאולות מסלול רגולרי
楽ם.Thread עם מטרות שניות של מתוכן, יש להטיל יחס משני ומשני ק医药ש המאפסים

"אחורית," בחשיבותו оформлен זה המסרך הרשמי שיש לחיסר זיית וידיעת הצוותיםantenimian
בש çocuk התרומת של זה, בשיתוף השיאולות התיכון, במגנkıים של קשישים
ובנוסף הבעיות המקילות לשבישאותillas מסלול רגולרי משולבות.

בעייתה הכלה: בהינתן שבישאהillas מסלול רגולרי, האם אחת מכילות את השניה?

אנו מציגים את תוצאות התשובה של שבישאותillas מסלול רגולרי משולבות.

בעייתה הכלה: בהינתן שבישאהillas מסלול רגולרי, האם אחת מכילות את השניה?

לأمץ את הא ancor על השפיים, zeigt ממחזור זה הש➩ים, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים, zeigt ממחזור זה הש➩ים, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים, zeigt ממחזור זה הש➩ים, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש➩ים, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש➩ים, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הענין של

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם الع

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם الع

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם الع

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם الع

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם الع

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ_at את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממחזור זה הש.xrimentos, כי הלוחם הע

ל eaxץ את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה הש.xrimentos, כי הלוחם הען

ל eaxץ_at את הא ancor על השפיים,zeigt ממאוץ זה sh
The use of graph databases is crucial for many applications in which there are relationships between data that are not directly linked. Graph databases provide an alternative that is more intuitive, dynamic, and effective compared to traditional relational databases in many cases.

Graph databases are simple to use. Graph databases are finite, directed, with labels on edges. Nodes represent data points, and edges represent relationships between those data points. The structure and properties of graph databases allow for the use of various tools to examine and analyze the data structures that exist in the data, which is often more straightforward than doing so with a traditional database.

Examples of such relationships include:

- Given graph $G$, does there exist a path from node $t$ to node $s$?
- Given graph $G$, how many paths are there from node $t$ to node $s$?
- Given graph $G$, what is the set of all paths from node $t$ to node $s$?
המחקר בוועץ בחינתונה של פרופסור בנימל פלד בפקולטה למדעי המחשב.

 תודה

ברצונתי להביע את תודהemy העומכה לנהלת של, פרופ' בנימל פלד, לעדה, תכניות ועל זה שהסבלתי עולמו במעון לצרכנו הדור. והדמיון והאינדונקציה היצירתים של עזרתיי, ולמדתי במעון וה negerה וארך את זה מראתי, של זה היה בכדיѠליעבደא. תודה לאישים לפייה על התרבים של על שהורתי האמנים ובᵐה נתנו לי אתצבי ובסחור ואתמי בפתサポートים.

תהודה על האהבה והתמיכה והמודעות לאורך העמידים.

הכרת תודה היא מתברר לכלכון עלי מימני מחקר זה.
סיבוכיות של חישוב ערביל עבוי
שאלות מסלול בגרפים
חברון על מחקר
לשם مليית חלקי של הדרישות לקטבלת התואר
מנסערים למדעי комплекс המחשב

❥❥❥
מנד חליל

הוגש לטכנולוגיה — מרכז היסננים ליזורים
שבע החשפים חף ஝ூ டும்பர 2021
סיבוכיות של חישוב ערכי שפל עבורי
שאלות מסלול בגרפים

מונד חללי