Context-Free Path Querying with All-Path Semantics by Matrix Multiplication

Rustam Azimov
st013567@student.spbu.ru
rustam.azimov@jetbrains.com
Saint Petersburg State University
JetBrains Research
St. Petersburg, Russia

Ilya Epelbaum
iliyepelbaun@gmail.com
Saint Petersburg State University
JetBrains Research
St. Petersburg, Russia

Semyon Grigorev
s.v.grigoriev@spbu.ru
semyon.grigorev@jetbrains.com
Saint Petersburg State University
JetBrains Research
St. Petersburg, Russia

ABSTRACT

Context-Free Path Querying (CFPQ) allows one to use context-free grammars as path constraints in navigational graph queries. Many algorithms for CPFQ were proposed but recently showed that the state-of-the-art CFPQ algorithms are still not performant enough for practical use. One promising way to achieve high-performance solutions for graph querying problems is to reduce them to linear algebra operations. Recently, there are two CFPQ solutions formulated in terms of linear algebra: the matrix-based CFPQ algorithm proposed by Azimov et al. (2018) and the Kronecker product-based CFPQ algorithm proposed by Orachev et al. (2020). However, the matrix-based algorithm still does not support the most expressive all-path query semantics and cannot be truly compared with Kronecker product-based CFPQ algorithm. In this work, we introduce a new matrix-based CFPQ algorithm with all-path query semantics that allows us to extract all found paths for each pair of vertices. Also, we implement our algorithm by using appropriate high-performance libraries for linear algebra. Finally, we provide a comparison of the most performant linear algebra-based CFPQ algorithms.

1 INTRODUCTION

Formal language-constrained path querying [3] is a graph analysis problem in which formal languages are used as constraints for navigational path queries. In this problem, a path in an edge-labeled graph is viewed as a word constructed by concatenation of edge labels. The formal languages are used to constrain the paths of interest: a query should find only paths labeled by words from the language. The most popular class of constraints used as navigational queries in graph databases are the regular ones. However, in some cases, regular languages are not expressive enough and context-free languages are used instead. The context-free path querying (CFPQ), can be used in many areas, for example, RDF analysis [20], static code analysis [13, 21], biological data analysis [15], graph segmentation [11].

CFPQ has been studied a lot since the problem was first stated by Mihalis Yannakakis in 1990 [19]. Jelle Hellings investigates various aspects of CFPQ in [6–8] and formulates three possible querying semantics: *relational* that requires to find all vertex pairs reachable by some path of interest, *single-path* query semantics also requires to return the example of such path for all vertex pairs, and *all-path* query semantics that requires to return all such paths for all vertex pairs.

A number of CFPQ algorithms based on parsing techniques were proposed: (G)LL and (G)LR-based algorithms by Ciro M. Medeiros et al. [10], Fred C. Santos et al. [14], Semyon Grigorev et al. [5], and Ekaterina Verbitskaia et al. [17]; CYK-based algorithm by Xiaowang Zhang et al. [20]; combinators-based approach to CFPQ by Ekaterina Verbitskaia et al. [18]. Yet recent research by Jochem Kuijpers et al. [9] shows that existing solutions are not applicable for real-world graph analysis because of significant running time and memory consumption.

One promising way to achieve high-performance solutions for graph querying problems is to reduce them to linear algebra operations. Rustam Azimov and Semyon Grigorev proposed a matrix-based algorithm for CFPQ with the relational query semantics in [2] and with the single-path query semantics in [16]. This algorithms provide a solution performant enough for real-world data analyses. However, in some cases, such as graph database querying or program analysis, this is important to provide all founded paths for detecting all possible connections in data. Recently this matrix-based algorithm does not support the all-path query semantics and cannot be used for this scenario. Another linear algebra-based CFPQ algorithm was proposed by Orachev et al. in [12]. This Kronecker product-based CFPQ algorithms creates more expressive and complex index that can be used for answering all three query semantics. Thus, the matrix-based CFPQ algorithm cannot be truly compared with the Kronecker product-based algorithm yet.

To sum up, we make the following contributions in this paper.

(1) We modify Azimov's matrix-based CFPQ algorithm and provide a matrix-based CFPQ algorithm for all-path query semantics. Our modification is still based on linear algebra, hence it still allows one to use high-performance libraries and utilize modern parallel hardware for CFPQ evaluation.
(2) We provide a comparison of the most performant linear algebra-based CFPQ algorithms. For this, we implement the proposed algorithm and compare it with other linear algebra-based implementations using a real-world RDF dataset. We show that the proposed algorithm is performant enough for real-world data analysis and is comparable with other most performant CFPQ implementations.

2 PRELIMINARIES

In this section, we introduce common definitions in graph theory and formal language theory which are used in this paper. Also, we provide a brief description of CFPQ problems.

2.1 Basic Definitions of Graph Theory

In this paper, we use a labeled directed graph as a data model and define it as follows.

Definition 2.1. Labeled directed graph is a tuple $D = (V, E, \Sigma)$, where

- V is a finite set of vertices. For simplicity, we assume that the vertices are natural numbers ranging from 0 to $|V| - 1$,
- $E \subseteq V \times \Sigma \times V$ is a set of labeled edges,
- Σ is a set of edge labels.

An example of the labeled directed graph D_1 is presented in Figure 1. Here the set of labels $\Sigma = \{a, b\}$.

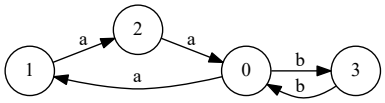


Figure 1: The input graph D_1

Definition 2.2. The *path* π in the graph $D = (V, E, \Sigma)$ is a finite sequence of labeled edges $(e_1, ..., e_n)$, where $\forall i, 1 \leq j \leq n : e_i = (v_{i-1}, l_i, v_i) \in E$.

Definition 2.3. The *word* $l(\pi) \in \Sigma^*$ in the graph $D = (V, E, \Sigma)$ is the unique word $l_1...l_n$, obtained by concatenating the labels of the edges along the path $\pi = (e_1 = (v_0, l_1, v_1), \ldots, e_n = (v_{n-1}, l_n, v_n))$ in the graph D.

2.2 Basic Definitions of Formal Languages

We use context-free grammars as path constraints, thus in this subsection, we define context-free languages and grammars.

Definition 2.4. A *context-free grammar* G is a tuple (N, Σ, P, S), where

- N is a finite set of nonterminals
- Σ is a finite set of terminals, $N \cap \Sigma = \varnothing$
- P is a finite set of productions of the form $A \to \alpha$, where $A \in N,\ \alpha \in (N \cup \Sigma)^*$
- S is the start nonterminal

We use the conventional notation $A \overset{*}{\underset{G}{\Longrightarrow}} w$ to denote, that a string $w \in \Sigma$ can be derived from a non-terminal A by some sequence of production rule applications from P in grammar G.

Definition 2.5. A *context-free language* is a language generated by a context-free grammar $G = (N, \Sigma, P, S)$:

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Longrightarrow}} w\}.$$

Definition 2.6. A context-free grammar $G = (N, \Sigma, P, S)$ is in *weak Chomsky normal form* (WCNF) if every production in P has one of the following forms:

- $A \to BC$, where $A, B, C \in N$
- $A \to a$, where $A \in N, a \in \Sigma$
- $A \to \varepsilon$, where $A \in N$

Note that weak Chomsky normal form differs from Chomsky normal form in the following:

- ε can be derived from any non-terminal;
- S can occur on the right-hand side of productions.

The matrix-based CFPQ algorithms process grammars only in weak Chomsky normal form, but every context-free grammar can be transformed into the equivalent grammar in this form.

Consider the context-free grammar $G_1 = (\{S\}, \{a, b\}, P, S)$, where P contains two rules: $S \to a\ S\ B$; $\quad S \to a\ b$.

This grammar generates the context-free language:

$$L(G_1) = \{a^n b^n, n \in \mathbb{N}\}.$$

The following production rules of the grammar G_1^{wcnf} is a result of the transformation of G_1 to weak Chomsky normal form:

$$S \to A\ B \qquad S_1 \to S\ B \qquad B \to b$$
$$S \to A\ S_1 \qquad A \to a$$

2.3 Context-Free Path Querying

Definition 2.7. Let $D = (V, E, \Sigma)$ be a labeled graph, $G = (N, \Sigma, P, S)$ be a context free grammar. Then a *context-free relation* with grammar G on the labeled graph D is the relation $R_{G,D} \subseteq V \times V$:

$$R_{G,D} = \{(v_0, v_n) \in V \times V \mid \exists \pi = (e_1 = (v_0, l_1, v_1), \ldots, e_n = (v_{n-1}, l_n, v_n)) \in \pi(D) : l(\pi) \in L(G)\}.$$

For example, the vertex pair $(0, 0) \in R_{G_1,D_1}$, since there is a path in the labeled graph D_1 presented in Figure 1 from the vertex 0 to the vertex 0, whose labeling forms a word

$$w = aaaaaabbbbbb = a^6b^6 \in L(G_1).$$

Finally, we can define context-free path querying problems.

Definition 2.8. Context-free path querying problem with relational query semantics is the problem of finding context-free relation $R_{G,D}$ for a given directed labeled graph D and a context-free grammar G.

In other words, the result of context-free path query evaluation is a set of vertex pairs such that there is a path between them that forms a word from the language generated by the given context-free grammar.

Using this definition, we can also define context-free path querying problems with single-path and all-path query semantics.

Definition 2.9. Context-free path querying problem with single-path query semantics for a given directed labeled graph D and a context-free grammar G is the problem of finding context-free relation $R_{G,D}$ and finding for each vertex pair $(v_0, v_n) \in R_{G,D}$ the one example of path π between these vertices such that $l(\pi) \in L(G)$.

Definition 2.10. Context-free path querying problem with all-path query semantics for a given directed labeled graph D and a context-free grammar G is the problem of finding context-free relation $R_{G,D}$ and finding for each vertex pair $(v_0, v_n) \in R_{G,D}$ all paths π between these vertices such that $l(\pi) \in L(G)$.

3 MATRIX-BASED CFPQ ALGORITHM FOR ALL-PATH QUERY SEMANTICS

In this section, we introduce the AllPathIndex structure which is used as a base of our solution for all-path query semantics. Also, we propose the matrix-based algorithm for CFPQ w.r.t. the all-path query semantics.

3.1 AllPathIndex

Our algorithm is based on Azimov's CFPQ algorithm [2] which is based on matrix operations. This algorithm reduces CFPQ to operations over Boolean matrices and as a result, allows one to use high-performance linear algebra libraries and utilize modern parallel hardware for CFPQ.

Note, that the algorithm computes not only the context-free relation $R_{G,D}$ but also a set of context-free relations $R_{G_A,D} \subseteq V \times V$ for every $A \in N$ where $G_A = (N, \Sigma, P, A)$. Thus it provides information about paths that form words derivable from any nonterminal $A \in N$.

We use an idea similar to one that was used for the CFPQ with single-path query semantics in [16]. We store additional information in matrices to be able to restore all paths which form words derivable from any nonterminal in the given grammar.

In order to do this, we introduce the

$$AllPathIndex = (left, right, middles)$$

— the elements of matrices that describe the found paths as concatenations of two smaller paths and help to restore each path after the index creation. Here *left* and *right* stand for the indexes of starting and ending vertices in the founded path, *middles* — the set of indexes of intermediate vertices used in the concatenation of two smaller paths. When we do not find the path for some vertex pair i, j, we use the $AllPathIndex = \bot = (0, 0, \emptyset)$.

Additionally, we will use the notation of *proper matrix* which means that for every element of the matrix with indexes i, j it either $AllPathIndex = (i, j, \_)$ or $\bot$.

For proper matrices we use a binary operation $\otimes$ defined for AllPathIndexes AP_1, AP_2 which are not equal to $\bot$ and with $AP_1.right = AP_2.left$ as

$$AP_1 \otimes AP_2 = (AP_1.left, AP_2.right, \{AP_1.right\}).$$

And if at least one operand is equal to $\bot$ then $AP_1 \otimes AP_2 = \bot$.

For proper matrices we also use a binary operation $\oplus$ defined for AllPathIndexes AP_1, AP_2 which are not equal to $\bot$ with $AP_1.left = AP_2.left$ and $AP_1.right = AP_2.right$ as

$$AP_1 \otimes AP_2 = (AP_1.left, AP_1.right, AP_1.middles \cup AP_2.middles).$$

If only one operand is equal to $\bot$ then $AP_1 \oplus AP_2$ equal to another operand. If both operands are equal to $\bot$ then $AP_1 \oplus AP_2 = \bot$.

Using $\otimes$ as multiplication of AllPathIndexes, and $\oplus$ as an addition, we can define a *matrix multiplication*, $a \odot b = c$, where a and b are matrices of a suitable size, that have AllPathIndexes as elements, as $c_{i,j} = \bigoplus_{k=1}^{n} a_{i,k} \otimes b_{k,j}$.

Also, we use the element-wise + operation on matrices a and b with the same size: $a + b = c$, where $c_{i,j} = a_{i,j} \oplus b_{i,j}$.

3.2 The matrix-based algorithm

We introduce the matrix-based algorithm for CFPQ w.r.t. the all-path query semantics (see Listing 1). This algorithm is a modification of Azimov's matrix-based algorithm for CFPQ and it constructs the set of matrices T with AllPathIndexes as elements. Let $G = (N, \Sigma, P, S)$ be the input context-free grammar, $D = (V, E, \Sigma)$ be the input graph. The result of the algorithm is a set of matrices T which stores information about all paths in the graph D that form a word derivable from some nonterminal of the context-free grammar G. Note that in line 4 we add the special value n to the $T^{A_i}_{k,l}.middles$ to specify that this path is a single-edge path or an empty path π_ε.

After constructing a set of matrices T or so-called *index*, we can construct a set of all paths π between specified vertex pair (i, j) and a non-terminal A such that $A \overset{*}{\underset{G}{\Longrightarrow}} l(\pi)$. The index T already stores data about all paths derivable from each nonterminal. However, the set of such paths can be infinite. From a practical perspective, it is necessary to use lazy evaluation or limit the resulting set of paths in some other way. For example, one can try to query some fixed number of paths or query paths of fixed maximum length.

Listing 1 CFPQ algorithm for all-path query semantics

```
1: function AllPathCFPQ(
       D = (V, E, Σ),
       G = (N, Σ, P, S))                          ▷ Grammar in WCNF
2:     n ← |V|
3:     T ← {T^A | A ∈ N, T^A is a matrix n × n, T^A_{i,j} ← ⊥ }
4:     for all (i, x, j) ∈ E, A | A → x ∈ P do T^A_{i,j} ← (i, j, {n})
5:     for all A | A → ε ∈ P do T^A_{i,i} ← (i, i, {n})
6:     while any matrix in T is changing do
7:         for all A → BC ∈ P where T^B or T^C are changed do
8:             T^A ← T^A + (T^B ⊙ T^C)
9:     return T
```

We propose the algorithm (see Listing 2) for extracting these paths. Our algorithm returns a set with the empty path π_ε only if $i = j$ and $A \to \varepsilon \in P$. If the AllPathIndex for the given i, j, A is equal to $\bot$ then our algorithm returns the empty set since such paths do not exist. Note that in line 19 we use the operation $\cdot$ which naturally generalizes the path concatenation operation by constructing all possible concatenations of path pairs from the given two sets. It is assumed that the sets are computed lazily, to ensure the termination in case of an infinite number of paths.

Listing 2 All paths extraction algorithm

```
1: function extractAllPaths(i, j, A, T = {T^{A_i}}, G = (N, Σ, P, S))
2:     index ← T^A_{i,j}
3:     if index = ⊥ then
4:         return ∅                                ▷ Such paths do not exist
5:     n ← size of the square matrix T^A
6:     resultPaths ← ∅
7:     for all middle ∈ index.middles do
8:         if middle = n then                      ▷ Add single-edge or empty paths
9:             for all x | A → x ∈ P do
10:                if (i, x, j) ∈ E then
11:                    resultPaths ← resultPaths ∪ {((i, x, j))}
12:            if (i = j) ∧ (A → ε ∈ P) then
13:                resultPaths ← resultPaths ∪ {π_ε}
14:        else▷ Add to result the concatenated paths from i to middle
   and from middle to j
15:            for all A → BC ∈ P do
16:                index_B ← T^B_{i,middle}
17:                index_C ← T^C_{middle,j}
18:                if (index_B ≠ ⊥) ∧ (index_C ≠ ⊥) then
19:                    lPaths ← extractAllPaths(i, middle, B, T, G)
20:                    rPaths ← extractAllPaths(middle, j, C, T, G)
21:                    resultPaths ← resultPaths ∪ lPaths · rPaths
22:    return resultPaths
```

3.3 Correctness

The following correctness theorem holds.

Theorem 1. *Let $G = (N, \Sigma, P, S)$ be the input context-free grammar, $D = (V, E, \Sigma)$ be the input graph, and T be a set of matrices returned by the algorithm in Listing 1. Then for any i, j and for any non-terminal $A \in N$, $index = T^A_{i,j}$ and $index = (i, j, middles) \neq \bot$ iff $(i, j) \in R_{G_A,D}$ and there is a path π from vertex i to j such that $l(\pi) \in G_A = (N, \Sigma, P, A)$.*

Proof sketch. At each iteration of the main cycle in lines 6-8 of the algorithm, the new paths corresponding to nonterminals

$A \in N$ are considered using the rules $A \to BC \in P$. These new paths are obtained by the concatenation of two smaller paths corresponding to the nonterminals B and C. At the initialization step of the algorithm in lines 3-5, we consider all single-edge or empty paths corresponding to the derivation tree of height 1. Thus, it can be shown that at iteration l of the main cycle we consider all paths π such that there is a derivation tree of the height $h \le l + 1$ for the string $l(\pi)$ and a context-free grammar G_A. Therefore, the theorem can be proved using the induction on the height of such derivation trees. □

Now, using the theorem 1 and induction on the length of the path, it can be easily shown that the following theorem holds.

Theorem 2. *Let $G = (N, \Sigma, P, S)$ be the input context-free grammar, $D = (V, E, \Sigma)$ be the input graph, and T be a set of matrices returned by the algorithm in Listing 1. Then for any i, j and for any non-terminal $A \in N$ such that index $= T^A_{i,j}$ and index $= (i, j, middles) \neq \bot$, the algorithm in Listing 2 for these parameters will return a set of all paths π from vertex i to j such that $l(\pi) \in G_A = (N, \Sigma, P, A)$.*

We can, therefore, determine whether $(i, j) \in R_{G,D}$ by asking whether $T^S_{i,j} = \bot$. Also, we can extract all paths which form a word from the context-free language $L(G)$ by using our algorithm in Listing 2. Thus, we show how the context-free path query evaluation w.r.t. the all-path query semantics can be solved in terms of matrix operations.

3.4 An Example

In this section, we provide a step-by-step demonstration of the proposed algorithms.

We run the query on a graph D_1, presented in Figure 1. We provide a step-by-step demonstration of the work of algorithm in Listing 1 with the given graph D and grammar G_1^{wcnf} from section 2. After the matrix initialization in lines **3-5** of this algorithm, we have a set of matrices $T^{(1)}$, presented in Figure 2.

$$T^{(1),A} = \begin{pmatrix} \bot & (0,1,\{4\}) & \bot & \bot \\ \bot & \bot & (1,2,\{4\}) & \bot \\ (2,0,\{4\}) & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

$$T^{(1),B} = \begin{pmatrix} \bot & \bot & \bot & (0,3,\{4\}) \\ \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \\ (3,0,\{4\}) & \bot & \bot & \bot \end{pmatrix}$$

Figure 2: The initial matrices for the example query. The PathIndexes $T^{(1),S_1}_{i,j}$ and $T^{(1),S}_{i,j}$ are equal to $\bot$ for every i, j

After the initialization, the only matrices which will be updated are T^{S_1} and T^S. These matrices obtained after the first loop iteration is shown in Figure 3.

$$T^{(2),S} = \begin{pmatrix} \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & (2,3,\{0\}) \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

Figure 3: The first iteration of computing the transitive closure for the example query. The PathIndexes $T^{(1),S_1}_{i,j}$ are equal to $\bot$ for every i, j

When the algorithm at some iteration finds new paths for some non-terminal in the graph D_1, then it adds corresponding AllPathIndexes to the matrix for this non-terminal. For example, after the first loop iteration, AllPathIndex $(2, 3, \{0\})$ is added to the matrix T^S. This AllPathIndex is added to the element with a row index $i = 2$ and a column index $j = 3$. This means, that there is a path π from the vertex 2 to the vertex 3, such that $S \xRightarrow[G_1^{\text{wcnf}}]{*} l(\pi)$ and this path obtained by concatenation of two smaller paths via vertex 0.

The calculation of the index T is completed after k iterations, when a fixpoint is reached: $T^{(k)} = T^{(k-1)}$. For the example query, $k = 14$ since $T_{14} = T_{13}$. The resulted matrix for non-terminal S is presented in Figure 4.

$$T^{(14),S} = \begin{pmatrix} (0,0,\{1\}) & \bot & \bot & (0,3,\{1\}) \\ (1,0,\{2\}) & \bot & \bot & (1,3,\{2\}) \\ (2,0,\{0\}) & \bot & \bot & (2,3,\{0\}) \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

Figure 4: The final matrix for non-terminal S after computing the index

Now, after constructing the index, we can construct the context-free relation

$$R_{G_1^{\text{wcnf}},D_1} = \{(0,0), (0,3), (1,0), (1,3), (2,0), (2,3)\}.$$

In the relation $R_{G_1^{\text{wcnf}},D_1}$, we have all vertex pairs corresponding to paths, whose labeling is in the language $L(G_1^{\text{wcnf}}) = \{a^n b^n \mid n \ge 1\}$. Using the algorithm in Listing 2 we can restore paths for each vertex pair from the context-free relation. For example, given $i = j = 0$, non-terminal S, set of resulted matrices T, and context-free grammar G_1^{wcnf}, the algorithm in Listing 2 returns an infinite set of all paths from vertex 0 to vertex 0 whose labeling form words from the following set $\{a^6b^6, a^{12}b^{12}, a^{18}b^{18}, \ldots\}$. Following the path corresponding to the word $a^{6m}b^{6m}$, we will go through the cycle with a labels $2m$ times and through the cycle with b labels $3m$ times for all $m \ge 1$.

4 EVALUATION

The goal of this evaluation is to investigate the applicability of the proposed matrix-based algorithm to CFPQ with all-path query semantics and to provide a comparison of the most performant linear algebra-based CFPQ algorithms. We will compare the following CFPQ implementations:

- *MtxSingle* — the implementation from [16] of the matrix-based CFPQ algorithm for the single-path query semantics,
- *Tns* — the implementation from [12] of the Kronecker product-based CFPQ algorithm for all three query semantics including the all-path query semantics,
- *MtxAll* — the implementation of the proposed matrix-based CFPQ algorithm for all-path query semantics which utilizes SuiteSparse[1] [4] implementation of GraphBLAS API for matrix manipulations.

All implementations utilize CPU and use matrices in sparse format. First, we measured the execution time and required memory of the index creation. Then we compared the practical applicability of the paths extraction for both implementations *MtxAll* and

[1] SuiteSparse is a sparse matrix software that includes GraphBLAS API implementation. Project web page: http://faculty.cse.tamu.edu/davis/suitesparse.html. Access date: 14.01.2021.

Table 1: Index creation time in seconds and memory in megabytes where we use "*err*" in case of out of memory error

Graph	#V	#E	G1						G2					
			MtxAll		Tns		MtxSingle		MtxAll		Tns		MtxSingle	
			Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem
pathways	6 238	18 598	0.04	91	0.02	123	0.01	671	0.01	49	0.01	122	0.01	671
go-hierarchy	45 007	980 218	22.12	38797	0.17	265	1.41	660	15.66	28447	0.24	252	0.84	671
enzyme	48 815	109 695	0.4	307	0.04	137	0.01	216	0.02	61	0.02	132	0.01	217
eclass_514en	239 111	523 727	25.02	14416	0.24	205	0.23	216	0.22	126	0.27	193	0.16	216
go	272 770	534 311	11.8	8290	1.58	282	1.45	215	1.13	990	1.27	243	0.93	217
geospecies	450 609	2 311 461	4.45	2691	0.08	218	0.06	2250	0.34	156	0.01	196	0.01	2251
taxonomy	5 728 398	14 922 125	*err*	*err*	4.42	2018	2.73	1962	19.13	27232	3.56	1776	1.15	2250

Table 2: Index creation time in seconds and memory in megabytes for *geo* query

Graph	MtxAll		Tns		MtxSingle	
	Time	Mem	Time	Mem	Time	Mem
geospecies	32.06	44235	26.32	19537	15.54	22941

Tns of the CFPQ with all-path query semantics. The source code is available on GitHub[2].

For evaluation, we used a PC with Ubuntu 18.04 installed. It has Intel core i7-6700 CPU, 3.4GHz, and DDR4 64Gb RAM. We only measure the execution time of the algorithms themselves, thus we assume an input graph is loaded into RAM in the form of its adjacency matrix in the sparse format.

4.1 Dataset Description

We use the graphs and respective queries from the CFPQ_Data dataset[3] provided in [16] that contains the real-world RDFs and queries g_1, g_2, geo that are variations of the *same-generation query* [1] — an important example of real-world queries that are context-free but not regular.

4.2 Evaluation Results

The results of the index creation for all three implementations are presented in Tables 1 and 2. We can see that the most performant index creation is in *MtxSingle* implementation, especially on big graphs. But *MtxSingle* applicable only for single-path query semantics and cannot restore all paths of interest. Thus, for single-path query semantics we can use a more simple index to restore only one path for each vertex pair. However, the Kronecker product-based implementation *Tns* uses a more complex but compact index and consumes less memory. The implementation *MtxAll* of the proposed matrix-based CFPQ algorithm for all-path query semantics has comparable to *Tns* execution time on small graphs but significantly slower execution time on big graphs with complex structure. Also, the *MtxAll* consumes significantly more memory than *Tns*. The reason for such behavior is that the proposed matrix-based algorithm is trying to store the information of all founded paths more explicitly. Also, the index constructed by *MtxAll* is less compact than the one constructed by *Tns*. On the biggest *taxonomy* graph and query g_1 we even have the out of memory error for *MtxAll* implementation.

After constructing the index, we compared the execution time of the path extraction for CFPQ with all-path query semantics using both *MtxAll* and *Tns* implementations. The results of path extraction for graphs *go* and *eclass_514en* are presented in Figures 5 and 6 (boxplots are standard, medians are indicated and outliers are omitted). For computation termination, we limit the maximum path length to 10. After that, we extract paths for each vertex pair and group the execution time by the number of paths returned. We can see that the path extraction running time of the implementation *MtxAll* of the proposed matrix-based algorithm is up to 1000 times faster than for the Kronecker product-based implementation *Tns*. As was mentioned above, in the proposed matrix-based CFPQ algorithm we construct an index with more explicit information about all founded paths. Thus, paths can be restored significantly faster than using the Kronecker product-based algorithm.

We can conclude the following.

- Our evaluation together with the evaluation from [12, 16] allow us to conclude that the most performant algorithm for the CFPQ with relational query semantics when the paths extraction is not required is Azimov's matrix-based algorithm from [2].
- For single-path query semantics when only one path per each vertex pair is required, the modification of Azimov's matrix-based CFPQ algorithm [16] is most performant.
- For all-path query semantics, the proposed matrix-based and the Kronecker product-based CFPQ algorithms have the following tradeoffs. If it is necessary to frequently recalculate the index for a changing graph or a path query then the best choice is the Kronecker product-based algorithm [12] with faster and less memory consuming index construction. If it is necessary to extract paths many times for once constructed index then the proposed matrix-based CFPQ algorithm is preferable.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose a new modification of Azimov's matrix-based CFPQ algorithm for all-path query semantics. We implement our algorithm using GraphBLAS API and provide a comparison of the most performant linear algebra-based algorithms.

We compare the CPU-based implementation. In the future, we want to obtain GPU-based and distributed implementations that also uses the GraphBLAS API.

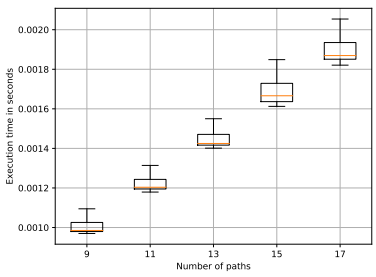
Also, further improvements in index creation and path extraction for both matrix-based and Kronecker product-based algorithms are required.

ACKNOWLEDGMENTS

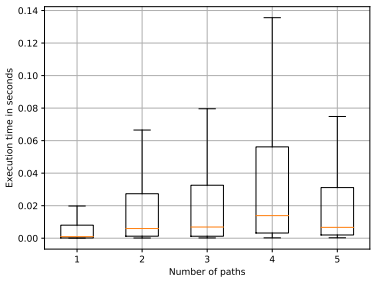
The reported study was funded by RFBR, project number 19-37-90101, and grant from JetBrains Research.

[2]Sources of all CFPQ implementations: https://github.com/JetBrains-Research/CFPQ_PyAlgo. Access date: 14.01.2021.

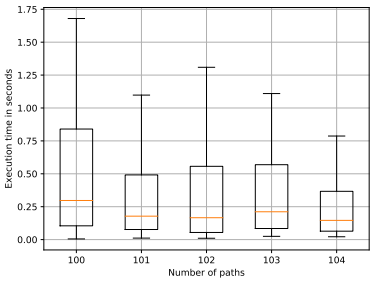
[3]CFPQ_Data dataset GitHub repository: https://github.com/JetBrains-Research/CFPQ_Data. Access date: 14.01.2021.



(a) $eclass\_514en$ and g_1

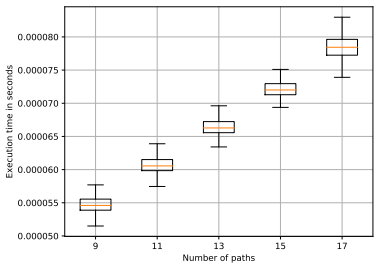


(b) go and g_1 for small number of paths

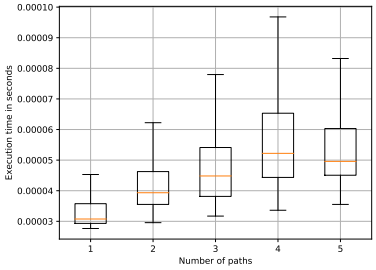


(c) go and g_1 for big number of paths

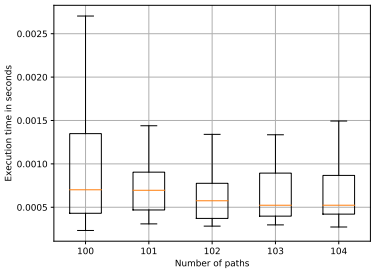
Figure 5: Execution time of the Kronecker product-based path extraction algorithm from [12] implemented in Tns depending on the number of paths returned



(a) $eclass\_514en$ and g_1



(b) go and g_1 for small number of paths



(c) go and g_1 for big number of paths

Figure 6: Execution time of the proposed matrix-based path extraction algorithm implemented in $MtxAll$ depending on the number of paths returned

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases: The Logical Level* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[2] Rustam Azimov and Semyon Grigorev. 2018. Context-free Path Querying by Matrix Multiplication. In *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA '18)*. ACM, New York, NY, USA, Article 5, 10 pages. https://doi.org/10.1145/3210259.3210264

[3] C. Barrett, R. Jacob, and M. Marathe. 2000. Formal-Language-Constrained Path Problems. *SIAM J. Comput.* 30, 3 (2000), 809–837. https://doi.org/10.1137/S0097539798337716 arXiv:https://doi.org/10.1137/S0097539798337716

[4] Timothy A. Davis. 2019. Algorithm 1000: SuiteSparse: GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra. *ACM Trans. Math. Softw.* 45, 4 (2019), 44:1–44:25. https://doi.org/10.1145/3322125

[5] Semyon Grigorev and Anastasiya Ragozina. 2017. Context-free Path Querying with Structural Representation of Result. In *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR '17)*. ACM, New York, NY, USA, Article 10, 7 pages. https://doi.org/10.1145/3166094.3166104

[6] Jelle Hellings. 2014. Conjunctive context-free path queries. In *Proceedings of ICDT'14*. 119–130.

[7] Jelle Hellings. 2015. Path Results for Context-free Grammar Queries on Graphs. *CoRR* abs/1502.02242 (2015). arXiv:1502.02242 http://arxiv.org/abs/1502.02242

[8] Jelle Hellings. 2015. Querying for Paths in Graphs using Context-Free Path Queries. *arXiv preprint arXiv:1502.02242* (2015).

[9] Jochem Kuijpers, George Fletcher, Nikolay Yakovets, and Tobias Lindaaker. 2019. An Experimental Study of Context-Free Path Query Evaluation Methods. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management (SSDBM '19)*. ACM, New York, NY, USA, 121–132. https://doi.org/10.1145/3335783.3335791

[10] Ciro M. Medeiros, Martin A. Musicante, and Umberto S. Costa. 2018. Efficient Evaluation of Context-free Path Queries for Graph Databases. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18)*. ACM, New York, NY, USA, 1230–1237. https://doi.org/10.1145/3167132.3167265

[11] H. Miao and A. Deshpande. 2019. Understanding Data Science Lifecycle Provenance via Graph Segmentation and Summarization. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 1710–1713.

[12] Egor Orachev, Ilya Epelbaum, Rustam Azimov, and Semyon Grigorev. 2020. Context-Free Path Querying by Kronecker Product. In *European Conference on Advances in Databases and Information Systems*. Springer, 49–59.

[13] Jakob Rehof and Manuel Fähndrich. 2001. Type-Base Flow Analysis: From Polymorphic Subtyping to CFL-Reachability. *SIGPLAN Not.* 36, 3 (Jan. 2001), 54–66. https://doi.org/10.1145/373243.360208

[14] Fred C. Santos, Umberto S. Costa, and Martin A. Musicante. 2018. A Bottom-Up Algorithm for Answering Context-Free Path Queries in Graph Databases. In *Web Engineering*, Tommi Mikkonen, Ralf Klamma, and Juan Hernández (Eds.). Springer International Publishing, Cham, 225–233.

[15] Petteri Sevon and Lauri Eronen. 2008. Subgraph Queries by Context-free Grammars. *Journal of Integrative Bioinformatics* 5, 2 (2008), 157 – 172. https://doi.org/10.1515/jib-2008-100

[16] Arseniy Terekhov, Artyom Khoroshev, Rustam Azimov, and Semyon Grigorev. 2020. Context-Free Path Querying with Single-Path Semantics by Matrix Multiplication. In *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA'20)*. Association for Computing Machinery, New York, NY, USA, Article 5, 12 pages. https://doi.org/10.1145/3398682.3399163

[17] Ekaterina Verbitskaia, Semyon Grigorev, and Dmitry Avdyukhin. 2016. Relaxed Parsing of Regular Approximations of String-Embedded Languages. In *Perspectives of System Informatics*, Manuel Mazzara and Andrei Voronkov (Eds.). Springer International Publishing, Cham, 291–302.

[18] Ekaterina Verbitskaia, Ilya Kirillov, Ilya Nozkin, and Semyon Grigorev. 2018. Parser Combinators for Context-free Path Querying. In *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala (Scala 2018)*. ACM, New York, NY, USA, 13–23. https://doi.org/10.1145/3241653.3241655

[19] Mihalis Yannakakis. 1990. Graph-theoretic Methods in Database Theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '90)*. ACM, New York, NY, USA, 230–242. https://doi.org/10.1145/298514.298576

[20] Xiaowang Zhang, Zhiyong Feng, Xin Wang, Guozheng Rao, and Wenrui Wu. 2016. Context-Free Path Queries on RDF Graphs. In *The Semantic Web – ISWC 2016*, Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil (Eds.). Springer International Publishing, Cham, 632–648.

[21] Xin Zheng and Radu Rugina. 2008. Demand-driven Alias Analysis for C. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '08)*. ACM, New York, NY, USA, 197–208. https://doi.org/10.1145/1328438.1328464