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 Azimov's matrix-based CFPQ algorithm (2018) and the Kronecker product-based CFPQ algorithm proposed by Orachev et al. (2020). However, the Azimov's algorithm still 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 compare our algorithm with other most performant CFPQ algorithms.

1 INTRODUCTION

Language-constrained path querying [2] is a way to search for paths in edge-labeled graphs where constraints are formulated in terms of a formal language. The language restricts the set of accepted paths: the sentence formed by the labels of a path should be in the language. Regular languages are the most popular class of constraints used as navigational queries in graph databases. In some cases, regular languages are not expressive enough and context-free languages are used instead. Context-free path querying (CFPQ), can be used for RDF analysis [23], biological data analysis [18], static code analysis [16, 24], and in other areas.

CFPQ have been studied a lot since the problem was first stated by Mihalis Yannakakis in 1990 [22]. Jelle Hellings investigates various aspects of CFPQ in [6–8]. A number of CFPQ algorithms were proposed: (G)LL and (G)LR-based algorithms by Ciro M. Medeiros et al. [12], Fred C. Santos et al. [17], Semyon Grigorev et al. [5], and Ekaterina Verbitskaia et al. [20]; CYK-based algorithm by Xiaowang Zhang et al. [23]; combinators-based approach to CFPQ by Ekaterina Verbitskaia et al. [21]. Nevertheless, the application of context-free constraints for real-world data analysis still faces many problems. The first problem is bad performance of the proposed algorithms on real-world data, as shown by Jochem Kuijpers et al. [11]. The second problem is that no graph database provides full-stack support of CFPQ, since most effort was made in developing algorithms and researching their theoretical properties. This fact hinders research of problems which can be reduced to CFPQ, thus it hinders the development of new solutions for them. For example, graph segmentation in data provenance analysis was recently reduced to CFPQ [14], but the evaluation of the proposed approach was complicated by the fact that no graph database supported CFPQ.

Rustam Azimov proposed a matrix-based algorithm for CFPQ in [1]. This algorithm provides a solution performant enough for real-world data analyses, as shown by Nikita Mishim et al. in [15] and Arseniy Terekhov et al. in [19]. This algorithm computes reachability and provides a single path which satisfies constraints for *every* vertex pair in the graph. Namely it solves *all-pairs* context-free path querying problem. In many real-world scenarios it is redundant to handle all possible pairs, instead one can provide one or a relatively small set of start vertices.

While all-pairs context-free path querying is a problem well studied, best to our knowledge, there is no solutions for the single-source and multiple-source CFPQ. In this work we propose a matrix-based *multiple-source* (and *single-source* as a partial case) CFPQ algorithm and provide full-stack support of CFPQ based on the proposed algorithm.

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

(1) We modify the Azimov's matrix-based CFPQ algorithm and provide a multiple-source matrix-based CFPQ algorithm. As a partial case, it is possible to use our algorithm in a single-source scenario. Our modification is still based on linear algebra, hence it is simple to implement and allows one to use high-performance libraries and utilize modern parallel hardware for queries evaluation.
(2) We provide full-stack support of CFPQ by extending the RedisGraph[1] [3] graph database. To do it, we implement a Cypher query language extension[2] that makes it possible to use context-free constraints, implemented the proposed algorithm in a RedisGraph backend, and supported the new syntax in the RedisGraph query execution engine. As far as we know, it is the first full-stack implementation of CFPQ. Finally, we evaluate the proposed solution and show that it is performant and memory-efficient enough to be applicable for real-world graph querying.

2 PRELIMINARIES

In this section we introduce common definitions in graph theory and formal language theory which are used in this paper. Also, we

[1] RedisGraph graph database Web-page: https://redislabs.com/redis-enterprise/redis-graph/. Access date: 19.07.2020.
[2] Proposal which describes path patterns specification syntax for Cypher query language: https://github.com/thobe/openCypher/blob/rpq/cip/1.accepted/CIP2017-02-06-Path-Patterns.adoc. The proposed syntax allows one to specify context-free constraints. Access date: 19.07.2020.

provide a brief description of CFPQ problems and AllPathIndex structure which is used as a base of our solution for all-path query semantics.

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 on Figure 1. Here the set of labels $\Sigma = \{a, b\}$.

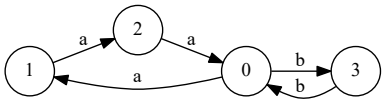

Figure 1: The input graph D_1.

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 paths 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 in 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: $S \to a\ S\ B$; $S \to a\ b$.

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 on Figure 1 from the vertex 0 to the vertex 0, whose labeling forms a word

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

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 $l(\pi) \in L(G)$.

2.4 Matrix-Based Algorithm

Our algorithm is based on the Azimov's CFPQ algorithm [1] which is based on matrix operations. This algorithm reduce 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 which form words derivable from any nonterminal $A \in N$.

We use the idea similar to one that was used for the CFPQ with single-path query semantics in [singlepath]. 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 which 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 MATRIX-BASED CFPQ ALGORITHM FOR ALL-PATH QUERY SEMANTICS

In this section, we propose 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.

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 given i, j, A is equal to $\bot$ then our algorithm returns $\emptyset$ 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, so as to ensure the termination in case of an infinite number of paths.

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
```

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
```

3.1 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 the 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* $= (i, j, middles) \neq \perp$, *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 $T^S_{i,j} = \perp$. Also, we can extract all paths which forms 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.

3.2 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}. After the matrix initialization in lines **3-5** of this algorithm, we have a set of matrices $T^{(1)}$, presented on Figure 2.

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

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

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

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

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

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** $\perp$ **for every** i, j**.**

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, $k = 13$ since $T_{13} = T_{14}$. The resulted matrix for non-terminal S is presented on Figure 4.

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

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 $a^{6k}b^{6k}$, we will go through the cycle with a labels $2k$ times and through the cycle with b labels $3k$ times for all $k \ge 1$.

4 EVALUATION

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. Note, that the time needed to load an input graph into the RAM is excluded from the time measurements.

4.0.1 Data Preparation. We use the graphs and respective queries g_1 and *geo* from [19] to evaluate the RedisGraph-based solution. The graphs are loaded into the RedisGraph database so that each vertex has a unique property `id` in the range $[0, \ldots, |V|-1]$, where $|V|$ is a number of vertices in the graph to load. This allows us to generate queries for a start vertex set with specific size using templates. The template for the g_1 query is provided in listing 3. Here `{id_from}` and `{id_to}` are placeholders for the lower and the upper bounds for `id`.

Listing 3 Cypher query pattern for g_1

```
1: PATH PATTERN S =
        ()-/ [<:SubClassOf [~S | ()] :SubClassOf]
        | [<:Type [~S | ()] :Type] /->()
2: MATCH (src)-/ ~S /->()
3: WHERE {id_from} <= src.id and src.id <= {id_to}
4: RETURN count(*)
```

We implemented a query generator for the queries g_1 and *geo* to create concrete queries for all the start sets which are used in the previous experiment.

4.0.2 Evaluation Results. We use *geo* query for *geospecies* graph as one of the hardest queries, and g_1 query for other graphs. We measure time and memory consumption for each start set.

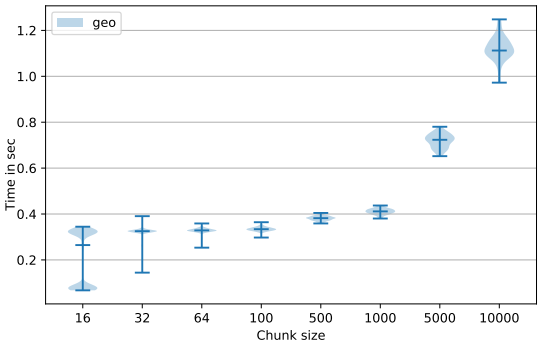

Figure 5: RedisGraph performance on *geospecies* graph

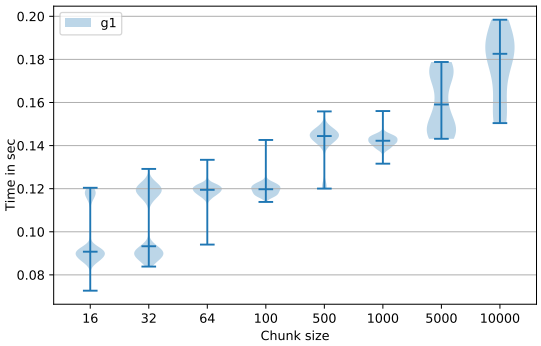

Figure 6: RedisGraph performance on *eclass_514en* graph

The execution time for all sets, except the set of size 10 000 for *geospecies* graph (fig. 5), is less than 1 second. Moreover, for smaller graph (*eclass_514en*), processing time is less than 0.2 second for all chunks (fig. 5).

Memory consumption for the big graphs *eclass_514en* and *geospecies* is presented in figures 7 and 8 respectively. The amount of memory used depends on the graph and the query, but RedisGraph uses less that 50Mb of RAM to process graphs with relatively small chunks ($\leq$ 1000). Note that RedisGraph includes memory management system, thus we measured all allocated memory, not only the memory really used for the query evaluation. As a result, we can conclude that the multiple-source CFPQ is significantly more memory efficient than creation of the complete reachability index and its filtering: processing the set of

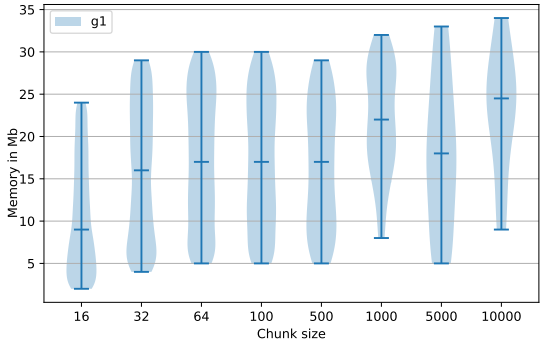

Figure 7: Memory consumption on *eclass_514en*

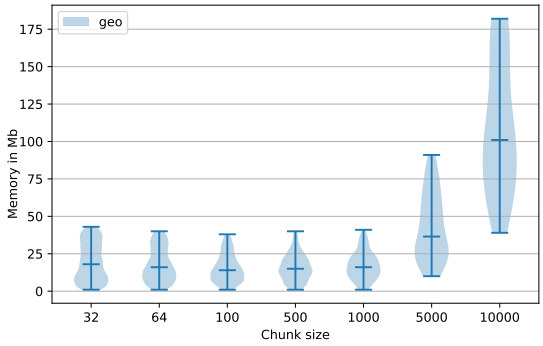

Figure 8: Memory consumption on *geospecies*

Table 1: Full graph processing with chunks of size 1000

Graph	#V	#E	Q	Chunks		Mono [19]
				T (sec)	Mem (Mb)	T (sec)
core	1323	4342	g_1	0.003	2	0.004
pathways	6238	18 598	g_1	0.031	6	0.011
gohierarchy	45 007	980 218	g_1	0.847	62	0.091
enzyme	48 815	109 695	g_1	0.698	13	0.018
eclass_514en	239 111	523 727	g_1	18.825	35	0.067
geospecies	450 609	2 311 461	*geo*	80.979	196	7.146
go	272 770	534 311	g_1	72.034	40	0.604

size 10 000 on *geospecies* graph requires less than 200Mb, while full index creation requires 16Gb [19].

We also evaluate how chunking affects the performance on the all-pairs reachability problem. We fix the size of a chunk to be 1000 for graphs of different sizes and measure time and memory required to process queries. Namely, we evaluate the query which is similar to the query from the previous scenario, but it does not constraint vertices ids (it does not have the `WHERE` clause). We measure total processing time (in seconds) and total required memory (in Mb). Also, we compare our solution with the results of Arseniy Terekhov et al. from [19] in which the Azimov's algorithm was naively integrated with RedisGraph without support of lazy query evaluation and query language. Similar hardware and the same input graphs and queries were used. Results are provided in table 1.

Although chunk-by-chunk processing is slower, it still requires reasonable time. Moreover, if the chunk size is comparable with the graph size (*core* and *pathways* graphs), then the execution time is comparable with the monolithic processing. Thus one can decrease execution time by increasing the chunk size. On

the other hand, even with relatively small chunks (*eclass_514*, *go* and *geospecies* graphs), when for chunk-by-chunk processing is 100 times slower, our results are still reasonable for some cases. For example, it requires over 70 times less time for *geospecies* graph processing than the solution of Jochem Kuijpers et al. [11] which is based on Neo4j and requires more than 6000 seconds. Moreover, while the solution from [19] requires huge amount of memory (more than 16Gb for *geospecies* graph and *geo* query), our solution requires only 196Mb. We argue, that our solution is more suitable for general-purpose graph databases. First of all, the core scenario when the set of start vertices is relatively small can be handled efficiently. Second, all-pairs reachability, which is not a massive case, can be solved in reasonable time with low memory consumption. One can easily tune our solution to get the optimal time and memory consumption for their specific case.

5 CONCLUSION AND FUTURE WORK

In this paper we propose a multiple-source modifications of Azimov's CFPQ algorithm and utilize it to provide full-stack support of CFPQ. For our solution, we implement a Cypher extension as a part of `libcypher-parser`, integrate the proposed algorithm into RedisGraph, and extend RedisGraph execution plan builder to support the extended Cypher queries. We demonstrate that our solution is applicable for real-world graph analyses.

In the future, it is necessary to provide formal translation of Cypher to linear algebra, or to determine a maximal subset of Cypher which can be translated to linear algebra. There is a number of works on the translation of a subset of SPARQL to linear algebra, such as [4, 10, 13]. Most of them are practical-oriented and do not provide full theoretical basis to translate querying language to linear algebra. Others discuss only partial cases and should be extended to cover real-world query languages. Deep investigation of this topic can help to determine the restrictions of linear algebra utilization for graph databases.

ACKNOWLEDGMENTS

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

REFERENCES

[1] 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

[2] 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

[3] P. Cailliau, T. Davis, V. Gadepally, J. Kepner, R. Lipman, J. Lovitz, and K. Ouaknine. 2019. RedisGraph GraphBLAS Enabled Graph Database. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 285–286.

[4] Roberto De Virgilio. 2012. A Linear Algebra Technique for (de)Centralized Processing of SPARQL Queries. In *Conceptual Modeling*, Paolo Atzeni, David Cheung, and Sudha Ram (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 463–476.

[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] ISO/IEC. 1996. International Standard EBNF Syntax Notation. http://www.iso.ch/cate/d26153.html. 14977 edn. Online.

[10] Fuad Jamour, Ibrahim Abdelaziz, and Panos Kalnis. 2018. A Demonstration of MAGiQ: Matrix Algebra Approach for Solving RDF Graph Queries. *Proc. VLDB Endow.* 11, 12 (Aug. 2018), 1978–1981. https://doi.org/10.14778/3229863.3236239

[11] 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

[12] 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

[13] Saskia Metzler and Pauli Miettinen. 2015. On Defining SPARQL with Boolean Tensor Algebra. *CoRR* abs/1503.00301 (2015). arXiv:1503.00301 http://arxiv.org/abs/1503.00301

[14] 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.

[15] Nikita Mishin, Iaroslav Sokolov, Egor Spirin, Vladimir Kutuev, Egor Nemchinov, Sergey Gorbatyuk, and Semyon Grigorev. 2019. Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication. In *Proceedings of the 2Nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA'19)*. ACM, New York, NY, USA, Article 12, 5 pages. https://doi.org/10.1145/3327964.3328503

[16] 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

[17] 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.

[18] 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

[19] 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

[20] 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.

[21] 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

[22] 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

[23] 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.

[24] 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