Multiple-Source Context-Free Path Querying in Terms of Linear Algebra

Arseniy Terekhov
simpletondl@yandex.ru
Information Technologies,
Mechanics and Optics University
JetBrains Research
St. Petersburg, Russia

Vlada Pogozhelskaya
pogozhelskaya@gmail.com
Saint Petersburg State University
St. Petersburg, Russia

Vadim Abzalov
vadim.i.abzalov@gmail.com
Saint Petersburg State University
St. Petersburg, Russia

Timur Zinnatulin
teemychteemych@gmail.com
Saint Petersburg State University
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 express path constraints in navigational graph queries as context-free grammars. Although there are many algorithms for CFPQ developed, no graph database provides full-stack support of CFPQ. The Azimov's CFPQ algorithm is applicable for real-world graph analyses, as shown by Arseniy Terekhov. In this work we provide a modification to Azimov's algorithm for multiple-source CFPQ which makes the algorithm more practical and eases the integration into RedisGraph graph database. We also implement a Cypher graph query language extension for context-free constraints. Thus we provide the first full-stack support of CFPQ for graph databases. Our evaluation shows that the provided solution is suitable for real-world graph analyses.

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

[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.

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 provide a brief description of Azimov's algorithm which is used as a base of our solution.

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 of six elements $D = (V, E, \Sigma_V, \Sigma_E, \lambda_V, \lambda_E)$, where

- V is a finite set of vertices. For simplicity, we assume that the vertices are natural numbers ranging from 0 to $|V| - 1$.
- $E \subseteq V \times V$ is a set of edges.
- Σ_V and Σ_E are sets of labels of vertices and edges respectively, such that $\Sigma_V \cap \Sigma_E = \varnothing$.
- $\lambda_V : V \to 2^{\Sigma_V}$ is a function that maps a vertex to a set of its labels, which can be empty.
- $\lambda_E : E \to 2^{\Sigma_E} \setminus \{\varnothing\}$ is a function that maps an edge to a non-empty set of its labels, so each edge must have at least one label.

Labeled graph is the basis of the widely-used *property graph* data model and allows one to use not only edge labels but also vertex labels in navigation queries.

An example of the labeled directed graph D_1 is presented in figure 1. Here the sets of labels $\Sigma_V = \{x, y\}$ and $\Sigma_E = \{a, b, c, d\}$. We omit the sets of vertex labels whenever they are empty.

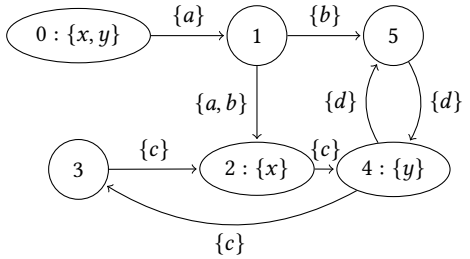

Figure 1: The input graph D_1

Definition 2.2. Path π in the graph $D = (V, E, \Sigma_V, \Sigma_E, \lambda_V, \lambda_E)$ is a finite sequence of vertices and edges $(v_0, e_0, v_1, e_1, ..., e_{n-1}, v_n)$, where $\forall i, 0 \le i \le n : v_i \in V; \forall j, 1 \le j \le n : e_j = (v_j, v_{j+1}) \in E$.

Definition 2.3. An *adjacency matrix* M of the graph D is a matrix of size $|V| \times |V|$, such that

$$M[i,j] = \begin{cases} \lambda_E((i,j)) & , (i,j) \in E \\ \varnothing & , otherwise \end{cases}$$

The adjacency matrix M of the graph D_1 (fig. 1) is the following:

$$M = \begin{pmatrix} \varnothing & \{a\} & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \{a,b\} & \varnothing & \varnothing & \{b\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{c\} & \varnothing \\ \varnothing & \varnothing & \{c\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \{c\} & \varnothing & \{d\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{d\} & \varnothing \end{pmatrix}.$$

Definition 2.4. Let M be an adjacency matrix of the graph D. Then the *adjacency matrix of label* $l \in \Sigma_E$ of graph D is a matrix $\mathcal{E}^l$ of size $|V| \times |V|$, such that

$$\mathcal{E}^l[i,j] = \begin{cases} 1 & , l \in M[i,j] \\ 0 & , otherwise \end{cases}$$

Definition 2.5. A *boolean decomposition of adjacency matrix* M of the graph D is a set of Boolean matrices $\mathcal{E} = \{\mathcal{E}^l \mid l \in \Sigma_E\}$, where $\mathcal{E}^l$ is the adjacency matrix of label l.

For example, the boolean decomposition of the adjacency matrix M of the graph D_1 is the set of matrices $\mathcal{E}^a, \mathcal{E}^b, \mathcal{E}^c, \mathcal{E}^d$:

$$\mathcal{E}^a = \begin{pmatrix} \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}, \ \mathcal{E}^b = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix},$$

$$\mathcal{E}^c = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}, \ \mathcal{E}^d = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}.$$

Definition 2.6. A *vertex label matrix* H of the graph D is a matrix of size $|V| \times |V|$, such that

$$H[i,j] = \begin{cases} \lambda_V(i) & , i = j \\ \varnothing & , otherwize \end{cases}$$

The vertex label matrix H of the example graph D_1 is

$$H = \begin{pmatrix} \{x,y\} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \{x\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{y\} & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \end{pmatrix}.$$

Definition 2.7. Let H be a vertex label matrix of graph D. Then the *vertices matrix of label* l is a matrix $\mathcal{V}^l$ of size $|V| \times |V|$, such that

$$\mathcal{V}^l[i,j] = \begin{cases} 1 & , l \in H[i,j] \\ 0 & , otherwise \end{cases}$$

Definition 2.8. A *boolean decomposition of vertex label matrix* H of the graph D is the set of Boolean matrices $\mathcal{V} = \{V^l \mid l \in \Sigma\}$, where $\mathcal{V}^l$ is a vertices matrix of label l.

Vertex label matrix H of the graph D_1 can be decomposed into a set of the following Boolean matrices:

$$\mathcal{V}^x = \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}, \ \mathcal{V}^y = \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}.$$

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 2.9. 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

Definition 2.10. A *context-free language* is a language generated by a context-free grammar G: $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Longrightarrow}} w\}$

Where $S \overset{*}{\underset{G}{\Longrightarrow}} w$ denotes that a string w can be generated from a starting non-terminal S using a sequence of productions from P.

Definition 2.11. 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 $G_1 = (\{S\}, \{c, d, y\}, P, S)$, where P contains two rules: $S \to c\ S\ d$; $\quad S \to c\ y\ d$.

This grammar generates the context-free language:

$$L(G_1) = \{c^n y d^n, n \in \mathbb{N}\}.$$

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

$$S \to C\ E \qquad E \to Y\ D \qquad C \to c \qquad D \to d$$
$$S \to C\ S_1 \qquad S_1 \to S\ D \qquad Y \to y$$

2.3 Context-Free Path Querying

Definition 2.12. Let $D = (V, E, \Sigma_V, \Sigma_E, \lambda_V, \lambda_E)$ be a labeled graph, $G = (N, \Sigma_V \cup \Sigma_E, P, S)$ be a context free grammar. Then a *context free relation* with grammar G on the labeled graph D is the relation $R_{G,D} \subseteq V \times V$:

$$R_{G,D} = \{(v_1, v_n) \in V \times V \mid \exists \pi = (v_1, e_1, v_2, \ldots, e_n, v_n) \in \pi(D) : l(\pi) \cap L(G) \neq \varnothing\},$$

where $l(\pi) \subset (\Sigma_V \cup \Sigma_E)^*$ is the set of labels along the path π:

$$l(\pi) = \lambda_V(v_1)^* \cdot \lambda_E(e_1) \cdot \lambda_V(v_2)^* \cdot \ldots \cdot \lambda_E(e_n) \cdot \lambda_V(v_n)^*$$

For example, π is a path from vertex 2 to vertex 5 in the labeled graph presented in figure 1: $\pi = 2 : \{x\} \xrightarrow{\{c\}} 4 : \{y\} \xrightarrow{\{d\}} 5$.

Labels along π form the set of sequences $l(\pi) = \{x^m c y^n d \mid n \geq 0, m \geq 0\}$. Only one of these sequences satisfies context-free constraints of the grammar G_1: cyd. Hence $l(\pi) \cap L(G_1) \neq \varnothing$ and the pair $(3, 6) \in R_{G_1,D}$.

Note that the definition of path labels allows for zero or more repetitions of a label of each vertex. This makes it possible to omit vertex labels or, if there are many vertex labels, to use them in an arbitrary order. It also permits to write a query which uses one vertex label multiple times. This definition may appear strange in some cases, but it depends on the semantics of the graph query language. Semantics formalization is planned for a future work, so we will stick to this definition in this paper.

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

Definition 2.13. Context-free path querying problem 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 and this path forms a word from the given language.

The context-free relation R_{G_1,D_1} for the graph D_1 and the context-free free grammar G_1 is the following:

$$R_{G_1,D_1} = \{(2, 4), (2, 5), (3, 4), (3, 5), (4, 4), (4, 5)\}.$$

Note that any relation $R_{G,D}$ can be represented as a Boolean matrix: $T[i, j] = 1 \iff (i, j) \in R_{G,D}$. In our example, R_{G_1,D_1} can be represented as follows:

$$T = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}.$$

Definition 2.14. Suppose Src is a given set of start vertices, then *multiple-source context-free path querying problem* for the given Src, directed labeled graph D and context-free grammar G is to find a context-free relation $R_{G,D}^{Src} \subseteq Src \times V \subseteq R_{G,D}$. Thus we restrict start vertices of the paths of interest to be vertices from the given set.

As a special case, a *single-source CFPQ* is when Src is a singleton set. If we set $Src = \{2\}$ in the previous example, then the result is: $R_{G_1,D_1}^{\{2\}} = \{(2, 4), (2, 5)\}$.

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. Moreover, utilization of Boolean matrices simplifies the implementation of the algorithm.

Note, that the algorithm computes not only the context-free relation $R_{G,D}$ but also a set of context-free relations $R_{A,D} \subseteq V \times V$ for every $A \in N$. Thus it provides information about paths which form words derivable from any nonterminal in the given grammar. Also, this algorithm handles only the edge labels.

As was shown by Nikita Mishin et al. [15] and Arseniy Terekhov et al. [19], this algorithm can be implemented using various high-performance programming techniques (including GPGPU utilization), and it is applicable for real-world graph analysis. But this algorithm solves *all-pairs* version of CFPQ: it finds all pairs of vertices in the given graph, such that there exist a path between them which forms a word in the given language. Thus it is impractical in cases when we are only interested in paths which start from the specific set of vertices, especially if this set is relatively small. Moreover, Azimov's algorithm operates over an adjacency matrix of the whole input graph, and as a result it requires a huge amount of memory which may be a problem for a real-world graph database.

3 MATRIX-BASED MULTIPLE-SOURCE CFPQ ALGORITHM

In this section we introduce a multiple-source matrix-based CFPQ algorithm. This algorithm is a modification of Azimov's matrix-based algorithm for CFPQ and its core idea is to cut off those vertices which are not in the selected set of start vertices.

Let $G = (N, \Sigma, P, S)$ be the input context-free grammar, $D = (V, E, \Sigma_V, \Sigma_E, \lambda_V, \lambda_E)$ be the input graph and Src be the input set of start vertices. The result of the algorithm is a Boolean matrix which represents relation $R_{G,D}^{Src}$.

Algorithm 1 Multiple-source CFPQ algorithm

```
1:  function MultiSrcCFPQNaive(
        D = (V, E, Σ_V, Σ_E, λ_V, λ_E),
        G = (N, Σ, P, S),                          ▷ Grammar in WCNF
        Src)
2:      T ← {T^A | A ∈ N, T^A[i, j] ← false, for all i, j}
3:      TSrc ← {TSrc^A | A ∈ N, TSrc^A[i, j] ← false, for all i, j}
4:      for all v ∈ Src do                         ▷ Input matrix initialization
5:          TSrc^S[v, v] ← true
6:      MSrc ← TSrc^S
7:      for all A → x ∈ P | x ∈ Σ_E do             ▷ Simple rules initialization
8:          for all (v, to) ∈ E | x ∈ λ_E(v, to) do
9:              T^A[v, to] ← true
10:     for all A → x ∈ P  | x ∈ Σ_V do
11:         for all v ∈ V | x ∈ λ_V(v) do
12:             T^A[v, v] ← true
13:     while T or TSrc is changing do             ▷ Algorithm's body
14:         for all A → BC ∈ P do
15:             M ← TSrc^A * T^B
16:             T^A ← T^A + M * T^C
17:             TSrc^B ← TSrc^B + TSrc^A
18:             TSrc^C ← TSrc^C + getDst(M)
19:     return MSrc * T^S
20: function getDst(M)
21:     A[i, j] ← false
22:     for all (v, to) ∈ V^2 | M[v, to] = true do
23:         A[to, to] ← true
24:     return A
```

In order to solve the single-source and multiple-source CFPQ problem, we modified the Azimov's algorithm. Each time, when a grammar rule is applied, only vertices of interest should be stored. To do it, we added one more matrix multiplication: $T^A = T^A + (TSrc^A \cdot T^B) \cdot T^C$, where $TSrc^A$ is a matrix of start vertices for the current iteration (lines **15-16** of the Algorithm 1). In the end of each iteration of the for loop, it is necessary to update the set of source vertices. To do it, we call the function getDst (see lines **20-24**), in line **18**. Thus, the modified algorithm supports the frontier of the vertices of interest and updates it on each iteration. Thus, it only computes the paths which start from the small set of selected vertices.

4 CFPQ FULL-STACK SUPPORT

To provide full-stack support of CFPQ, it is necessary to choose an appropriate graph database. It was shown by Arseniy Terekhov et al. [19] that matrix-based algorithm can be naturally integrated into RedisGraph because the algorithm and the database both operate over a matrix representation of graphs. Moreover, RedisGraph supports Cypher as a query language and there is a proposal which describes Cypher extension for context-free constraints. Thus we chose RedisGraph as a base for our solution.

4.1 Cypher Extension

The first thing to do is to extend the Cypher parser to support the context-free constraints. Tobias Lindaaker proposed an extension for context free constraints to the Cypher syntax[3], which is not implemented in the Cypher parsers yet.

This extension introduces path patterns, which are a powerful alternative to the original Cypher relationship patterns. Path patterns allow one to express regular constrains over the basic patterns such as relationship and node patterns. Like relationship patterns, they can be specified in the `MATCH` clause.

Listing 2 Query based on the example grammar G_1 written in Cypher with path patterns

```
1: PATH PATTERN S = ()-/ [:c ~S :d] | [:c (:y) :d] /->()
2: MATCH (v:x)-[:a | :c]->()-/ :b ~S /->(to)
3: RETURN v, to
```

The feature which allows one to specify context-free constraints is *named path patterns*: a path pattern can be assigned a name which can be used in other patterns or within the same pattern. Named patterns is defined in the `PATH PATTERN` clause. Using this feature, the structure of queries is pretty similar to a grammar in the Extended Backus-Naur Form (EBNF) [9].

An example of a query which uses named path patters is presented in listing 2. This query is based on the context-free grammar G_1. Namely, the path pattern `S` specifies exactly the same constraint as the grammar G_1. The `MATCH` clause consists of the relation pattern `[:a | :c]` and the path pattern `/:b ~S/`, and this path pattern references the named pattern `S`. The constraint specifies that a path of interest starts in a vertex labelled `x`, goes through an edge labelled either `a` or `c`, then the rest of the path is constrainted by a path pattern which starts with an edge `b` and follows with a path matched with `S`. The `RETURN` clause specifies what the result of the query is supposed to be. For the example graph D_1, this query returns the set of vertex pairs $\{(0, 4), (0, 5)\}$.

RedisGraph database supports a subset of the Cypher language and uses `libcypher-parser`[4] library to parse queries. We extend this library with the new syntax described in the proposal. Note that we implement[5] the complete syntax extension, not only the part necessary for simple CFPQ.

4.2 RedisGraph Extension

We implemented the multiple-source algorithm in the RedisGraph. We partially supported the proposed syntax extension in RedisGraph query execution engine so that one can specify the labels of edges and vertices and use named path patterns.

Processing the input as a whole may require a lot of memory. RedisGraph implements lazy evaluation: it creates execution strategy in terms of elementary operations each of which processes the input sequentially in *chunks*. This reduces memory consumption so that it does not depend on the input size which is crucial when dealing with big real-world graphs. However processing chunks comes with a time overhead. By changing the size of a chunk, a developer may adjust the ratio between the time and memory consumption so that it fits their needs.

We use subsets of the start vertices as chunks since it is most natural in the multiple source algorithm. We study how the size of a chunk affects the performance in the evaluation.

4.3 Evaluation

For RedisGraph evaluation, we used a PC with Ubuntu 18.04 installed. It has Intel Core i7-6700 CPU, 3.4GHz, and DDR4 64Gb RAM. RedisGraph with our extensions is installed[6].

[3]Formal syntax specification: https://github.com/thobe/openCypher/blob/rpq/cip/1.accepted/CIP2017-02-06-Path-Patterns.adoc#11-syntax. Access date: 19.07.2020.

[4]The `libcypher-parser` is an open-source parser library for Cypher query language. GitHub repository of the project: https://github.com/cleishm/libcypher-parser. Access date: 19.07.2020.

[5]The modified libcypher-parser library with support of syntax for path patterns: https://github.com/YaccConstructor/libcypher-parser. Access date: 19.07.2020.

[6]Sources of RedisGraph database with full-stack CFPQ support:https://github.com/YaccConstructor/RedisGraph/tree/path_patterns_dev. Access date: 19.07.2020.

4.3.1 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

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

4.3.2 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.

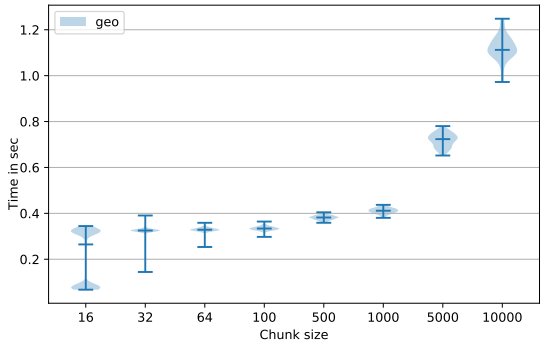

Figure 2: RedisGraph performance on *geospecies* graph

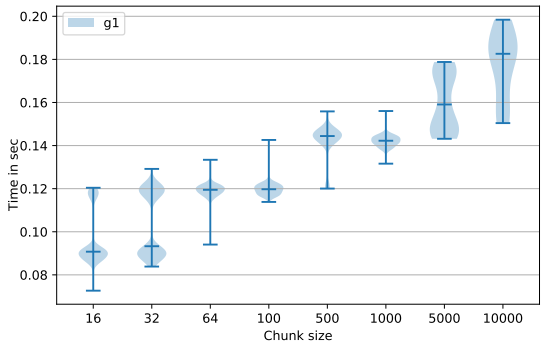

Figure 3: RedisGraph performance on *eclass_514en* graph

The execution time for all sets, except the set of size 10 000 for *geospecies* graph (fig. 2), is less than 1 second. Moreover, for smaller graph (*eclass_514en*), processing time is less than 0.2 second for all chunks (fig. 2).

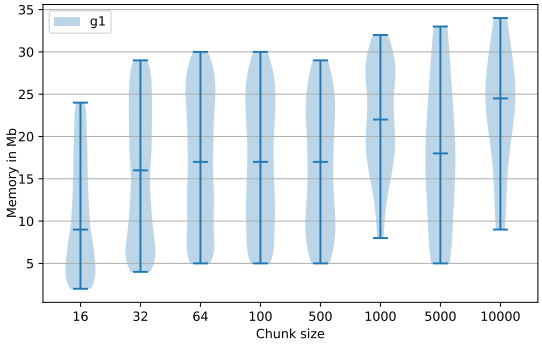

Figure 4: Memory consumption on *eclass_514en*

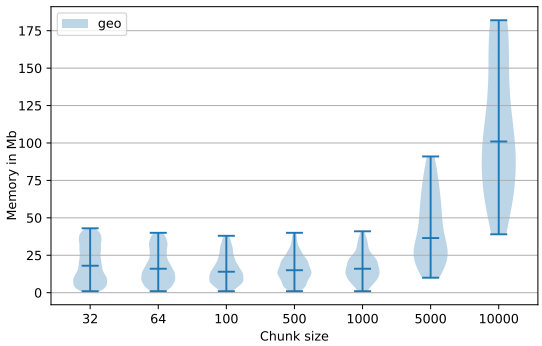

Figure 5: Memory consumption on *geospecies*

Memory consumption for the big graphs *eclass_514en* and *geospecies* is presented in figures 4 and 5 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 (≤ 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 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

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

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.

ACKNOWLEDGEMENTS

The research was supported by the Russian Science Foundation, grant №18-11-00100.

We thank Roi Lipman for his help with investigation of the RedisGraph internals and pointing out the impractical memory consumption of the original Azimov's algorithm which gave us the motivation to develop the presented solution.

We thank Ekaterina Verbitskaia for the fruitful discussion and feedback which helped us to improve the paper.

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