

# One Algorithm to Evaluate Them All: Unified Linear Algebra Based Approach to Evaluate Both Regular and Context-Free Path Queries

Egor Orachyov  
egororachyov@gmail.com  
Saint Petersburg State University  
St. Petersburg, Russia

**Rustam Azimov**  
rustam.azimov19021995@gmail.com  
Saint Petersburg State University  
St. Petersburg, Russia  
JetBrains Research  
St. Petersburg, Russia

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

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

## ABSTRACT

We propose a new algo for CFPQ! Abstract is very abstract.  
Abstract is very abstract. Abstract is very abstract. Abstract  
is very abstract. Abstract is very abstract. Abstract is very  
abstract. Abstract is very abstract. Abstract is very abstract.  
Abstract is very abstract. Abstract is very abstract. Abstract  
is very abstract. Abstract is very abstract. Abstract is very  
abstract. Abstract is very abstract. Abstract is very abstract.  
Abstract is very abstract. Abstract is very abstract. Abstract  
is very abstract. Abstract is very abstract. Abstract is very  
abstract. Abstract is very abstract. Abstract is very abstract.  
Abstract is very abstract. Abstract is very abstract. Abstract  
is very abstract. Abstract is very abstract. Abstract is very  
abstract. Abstract is very abstract. Abstract is very abstract.  
Abstract is very abstract. Abstract is very abstract. Abstract  
is very abstract.

## CCS CONCEPTS

- **Information systems** → Query languages for non-relational engines;
- **Theory of computation** → Grammars and context-free languages; *Parallel computing models*;
- **Computing methodologies** → Massively parallel algorithms;
- **Computer systems organization** → *Single instruction, multiple data*;

**ACM Reference Format:**

Egor Orachyov, Ilya Epelbaum, Rustam Azimov, and Semyon Grigorev. 2021. One Algorithm to Evaluate Them All: Unified Linear Algebra Based Approach to Evaluate Both Regular and Context-Free Path Queries. In . ACM, New York, NY, USA, 7 pages.

## 1 INTRODUCTION

Language-constrained path querying. Regular ath querying (RPQ) is !!!! Context-free is more specific, but actively developing last years.

CFPQ as a separated algorithms. Matrix is the fastest.

To make it usable... Integration with graph DB.

Integration with query languages. The problem. We cannot separate regular and context-free queries in general case.

Moreover, grammar transformation for matrix-based (the fastest existing algorithm) is required, !!!

Linear algebra, GraphBLAS, !!!! is a right way.

## Contribution

- (1) New algorithm. Based on operation over Booleana matrices. Correctness and time complexity.
- (2) The way to optimize queries.
- (3) Evaluation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

SIGMOD'21, ,

© 2021 Association for Computing Machinery.

ACM ISBN XXX-X-XXXXX-XXX-X...\$15.00

## 2 PRELIMINARIES

In this section we introduce basic definitions which are used in our work.

- Graph
- Grammar
- CFPQ with different semantics
- Finite state machine. Regexp to FSM.
- Recursive state machine. Grammar to RSM.
- Tensor product definition.
- Tensor product for FSM intersection.
- Rsm and FSM intersection classical theorem proof?

## 3 CONTEXT-FREE PATH QUERYING BY KRONECKER PRODUCT

In this section, we introduce the algorithm for CFPQ which is based on Kronecker product of Boolean matrices. The algorithm provides the ability to solve all-pairs CFPQ in all-paths semantics (according to Hellings [? ]) and consists of two the following parts.

- (1) Index creation. In the first step, the algorithm computes an index which contains information which is necessary to restore paths for specified pairs of vertices. This index can be used to solve the reachability problem without paths extraction. Note that this index is finite even if the set of paths is infinite.
- (2) Paths extraction. All paths for the given pair of vertices can be enumerated by using the index computed at the previous step. As far as the set of paths can be infinite, all paths cannot be enumerated explicitly, and advanced techniques such as lazy evaluation are required for implementation. Anyway, a single path can be always extracted by using standard techniques.

We describe both these steps, prove correctness, and provide time complexity estimations. After that we provide step-by-step example of query evaluation by using the proposed algorithm.

### 3.1 Index Creation Algorithm

In this section, we introduce the algorithm for the computation of context-free reachability in a graph  $\mathcal{G}$ . The algorithm determines the existence of a path, which forms a sentence of the language defined by the input RSM  $R$ , between each pair of vertices in the graph  $\mathcal{G}$ . The algorithm is based on the generalization of the FSM intersection for an RSM, and an input graph. Since a graph can be interpreted as a FSM, in which transitions correspond to the labeled edges between vertices of the graph, and an RSM is composed of a set of FSMs, the intersection of such machines can be computed using the classical algorithm for FSM intersection, presented in [? ].

The intersection can be computed as a Kronecker product of the corresponding adjacency matrices for an RSM and a graph. Since we are only determining the reachability of vertices, it is enough to represent intersection result as a Boolean matrix. It simplifies the algorithm implementation and allows one to express it in terms of basic matrix operations.

Listing 1 shows main steps of the algorithm. The algorithm accepts context-free grammar  $G = (\Sigma, N, P)$  and graph  $\mathcal{G} = (V, E, L)$  as an input. An RSM  $R$  is created from the grammar  $G$ . Note, that  $R$  must have no  $\varepsilon$ -transitions.  $M_1$  and  $M_2$  are the adjacency matrices for the machine  $R$  and the graph  $\mathcal{G}$  correspondingly.

Then for each vertex  $i$  of the graph  $\mathcal{G}$ , the algorithm adds loops with non-terminals, which allows deriving  $\varepsilon$ -word. Here the following rule is implied: each vertex of the graph is reachable by itself through an  $\varepsilon$ -transition. Since the machine  $R$  does not have any  $\varepsilon$ -transitions, the  $\varepsilon$ -word could be derived only if a state  $s$  in the box  $B$  of the  $R$  is both initial and final. This data is queried by the `getNonterminals()` function for each state  $s$ .

The algorithm terminates when the matrix  $M_2$  stops changing. Kronecker product of matrices  $M_1$  and  $M_2$  is evaluated for each iteration. The result is stored in  $M_3$  as a Boolean matrix. For the given  $M_3$  a  $C_3$  matrix is evaluated by the `transitiveClosure()` function call. The  $M_3$  could be interpreted as an adjacency matrix for an directed graph with no labels, used to evaluate transitive closure in terms of classical graph definition of this operation. Then the algorithm iterates over cells of the  $C_3$ . For the pair of indices  $(i, j)$ , it computes  $s$  and  $f$  — the initial and final states in the recursive automata  $R$  which relate to the concrete  $C_3[i, j]$  of the closure matrix. If the given  $s$  and  $f$  belong to the same box  $B$  of  $R$ ,  $s = q_B^0$ , and  $f \in F_B$ , then `getNonterminals()` returns the respective non-terminal. If the condition holds then the algorithm adds the computed non-terminals to the respective cell of the adjacency matrix  $M_2$  of the graph.

The functions `getStates` and `getCoordinates` (see listing 2) are used to map indices between Kronecker product arguments and the result matrix. The Implementation appeals to the blocked structure of the matrix  $C_3$ , where each block corresponds to some automata and graph edge.

The algorithm returns the updated matrix  $M_2$  which contains the initial graph  $\mathcal{G}$  data as well as non-terminals from  $N$ . If a cell  $M_2[i, j]$  for any valid indices  $i$  and  $j$  contains symbol  $S \in N$ , then vertex  $j$  is reachable from vertex  $i$  in grammar  $G$  for non-terminal  $S$ .

**LEMMA 3.1.** *Let  $\mathcal{G} = (V, E, L)$  be a graph and  $G = (\Sigma, N, P)$  be a grammar. Let  $\mathcal{G}_k = (V, E_k, L \cup N)$  be graph and  $M_k$  its adjacency matrix of the execution some iteration  $k \geq 0$  of the*

**Listing 1** Kronecker product based CFPQ

---

```

1: function CONTEXTFREEPATHQUERYING( $\mathcal{G}, \mathcal{G}$ )
2:    $R \leftarrow$  Recursive automata for  $\mathcal{G}$ 
3:    $M_1 \leftarrow$  Adjacency matrix for  $R$ 
4:    $M_2 \leftarrow$  Adjacency matrix for  $\mathcal{G}$ 
5:   for  $s \in 0..dim(M_1) - 1$  do
6:     for  $i \in 0..dim(M_2) - 1$  do
7:        $M_2[i, i] \leftarrow M_2[i, i] \cup \text{getNonterminals}(R, s, s)$ 
8:   while Matrix  $M_2$  is changing do
9:      $M_3 \leftarrow M_1 \otimes M_2$  ▷ Evaluate Kroncker product
10:     $C_3 \leftarrow \text{transitiveClosure}(M_3)$ 
11:     $n \leftarrow \text{dim}(M_3)$  ▷ Matrix  $M_3$  size =  $n \times n$ 
12:    for  $(i, j) \in [0..n - 1] \times [0..n - 1]$  do
13:      if  $C_3[i, j]$  then
14:         $s, f \leftarrow \text{getStates}(C_3, i, j)$ 
15:        if  $\text{getNonterminals}(R, s, f) \neq \emptyset$  then
16:           $x, y \leftarrow \text{getCoordinates}(C_3, i, j)$ 
17:           $M_2[x, y] \leftarrow M_2[x, y] \cup \text{getNonterminals}(R, s, f)$ 
18:   return  $M_2$ 

```

---

**Listing 2** Help functions for Kronecker product based CFPQ

---

```

1: function GETSTATES( $C, i, j$ )
2:    $r \leftarrow \text{dim}(M_1)$  ▷  $M_1$  is adjacency matrix for automata  $R$ 
3:   return  $\lfloor i/r \rfloor, \lfloor j/r \rfloor$ 
4: function GETCOORDINATES( $C, i, j$ )
5:    $n \leftarrow \text{dim}(M_2)$  ▷  $M_2$  is adjacency matrix for graph  $\mathcal{G}$ 
6:   return  $i \bmod n, j \bmod n$ 

```

---

algorithm ?? . Then for each edge  $e = (m, S, n) \in E_k$ , where  $S \in N$ , the following statement holds:  $\exists m\pi n : S \rightarrow_G l(\pi)$ .

PROOF. (Proof by induction)

**Basis:** For  $k = 0$  and the statement of the lemma holds, since  $M_0 = M$ ,  $M$  where is adjacency matrix of the graph  $G$ . Non-terminals, which allow to derive  $\varepsilon$ -word, are also added at algorithm preprocessing step, since each vertex of the graph is reachable by itself through an  $\varepsilon$ -transition.

**Inductive step:** Assume that the statement of the lemma holds for any  $k \leq (p - 1)$  and show that it also holds for  $k = p$ , where  $p \geq 1$ .

For the algorithm iteration  $p$  the Kronecker product  $K_p$  and transitive closure  $C_p$  are evaluated as described in the algorithm. By the properties of this operations, some edge  $e = ((s, m), (f, n))$  exists in the directed graph, represented by adjacency matrix  $C_p$ , if and only if  $\exists s\pi'f$  in the RSM graph, represented by matrix  $M_r$ , and  $\exists m\pi n$  in graph, represented by  $M_{p-1}$ . Concatenated symbols along the path  $\pi'$  form some derivation string  $v$ , composed from terminals and non-terminals, where  $v \rightarrow_G l(\pi)$  by the inductive assumption.

The new edge  $e = (m, S, n)$  will be added to the  $E_p$  only if  $s$  and  $f$  are initial and final states of some box  $B$  of the RSM corresponding to the non-terminal  $S_B$ . In this case, the grammar  $G$  has the derivation rule  $S_B \rightarrow_G v$ , by the inductive assumption  $v \rightarrow_G l(\pi)$ . Therefore,  $S_B \rightarrow_G l(\pi)$  and this completes the proof of the lemma.  $\square$

LEMMA 3.2. Let  $\mathcal{G} = (V, E, L)$  be a graph and  $G = (\Sigma, N, P)$  be a grammar. Let  $\mathcal{G}_k = (V, E_k, L \cup N)$  be graph and  $M_k$  its adjacency matrix of the execution some iteration  $k \geq 1$  of the algorithm ?? . For any path  $m\pi n$  in graph  $\mathcal{G}$  with word  $l = l(\pi)$  if exists the derivation tree of  $l$  for the grammar  $G$  and starting non-terminal  $S$  with the height  $h \leq k$ , then  $\exists e = (m, S, n) : e \in E_k$ .

PROOF. (Proof by induction)

**Basis:** Show that statement of the lemma holds for the  $k = 1$ . Matrix  $M$  and edges of the graph  $\mathcal{G}$  contains only labels from  $L$ . Since the derivation tree of height  $h = 1$  contains only one non-terminal  $S$  as a root and only symbols from  $\Sigma \cup \varepsilon$  as leaves, for all paths, which form a word with derivation tree of the height  $h = 1$ , the corresponding nonterminals will be added to the  $M_1$  via preprocessing step and first iteration of the algorithm. Thus, the lemma statement holds for the  $k = 1$ .

**Inductive step:** Assume that the statement of the lemma hold for any  $k \leq (p - 1)$  and show that it also holds for  $k = p$ , where  $p \geq 2$ .

For the algorithm iteration  $p$  the Kronecker product  $K_p$  and transitive closure  $C_p$  are evaluated as described in the algorithm. By the properties of this operations, some edge  $e = ((s, m), (f, n))$  exists in the directed graph, represented by adjacency matrix  $C_p$ , if and only if  $\exists s\pi_1f$  in the RSM graph, represented by matrix  $M_{RSM}$ , and  $\exists m\pi n$  in graph, represented by  $M_{p-1}$ .

For any path  $m\pi n$ , such that exist derivation tree of height  $h < k$  for the word  $l(\pi)$  with root non-terminal  $S$ , there exists edge  $e = (m, S, n) : e \in E_k$  by inductive assumption.

Suppose, that exists derivation tree  $T$  of height  $h = p$  with the root non-terminal  $S$  for the path  $m\pi n$ . The tree  $T$  is formed as  $S \rightarrow a_1..a_d, d \geq 1$  where  $\forall i \in [1..d]$   $a_i$  is sub-tree of height  $h_i \leq p - 1$  for the sub-path  $m_i\pi_i n_i$ . By inductive hypothesis, there exists path  $\pi_i$  for each derivation sub-tree, such that  $m = m_1\pi_1 m_2..m_d\pi_d m_{d+1} = n$  and concatenation of these paths forms  $m\pi n$ , and the root non-terminals of this sub-trees are included in the matrix  $M_{p-1}$ .

Therefore, vertices  $m_i \forall i \in [1..d]$  form path in the graph, represented by matrix  $M_{p-1}$ , with complete set of labels. Thus, new edge between vertices  $m$  and  $n$  with the respective non-terminal  $S$  will be added to the matrix  $M_p$  and this completes the proof of the lemma.  $\square$

THEOREM 3.3. Let  $\mathcal{G} = (V, E, L)$  be a graph and  $G = (\Sigma, N, P)$  be a grammar. Let  $\mathcal{G}_R = (V, E_R, L)$  be a result graph for the execution of the algorithm ?? . The following statement holds:  $e = (m, S, n) \in E_R$ , where  $S \in N$ , if and only if  $\exists m\pi n : S \rightarrow_G l(\pi)$ .

PROOF. This theorem is a consequence of the Lemma 3.1 and Lemma 3.2.  $\square$

**THEOREM 3.4.** *Let  $\mathcal{G} = (V, E, L)$  be a graph and  $G = (\Sigma, N, P)$  be a grammar. The algorithm ?? terminates in finite number of steps.*

PROOF. The main algorithm *while-loop* is executed while graph adjacency matrix  $M$  is changing. Since the algorithm only adds the edges with non-terminals from  $N$ , the maximum required number of iterations is  $|N| \times |V| \times |V|$ , where each component has finite size. This completes the proof of the theorem.  $\square$

### 3.2 Paths Extraction Algorithm

After index created one can enumerate all paths between specified vertices.

#### Listing 3 Paths extraction algorithm

```

1: function GETPATHS( $v_s, v_f, N, C_3, M_1, M_2$ )
2:    $s \leftarrow$  Start states of automata for  $N$ 
3:    $f \leftarrow$  Final states of automata for  $N$ 
4:    $res \leftarrow$  getPinner( $i, j, C_3, M_1, M_2$ )
5:   return  $res$ 
6: function GETSUBPATHS( $i, j, k, C_3, M_1, M_2$ )
7:    $l \leftarrow \{(i.g, t, k.g) \mid M_1[t][i.r, k.r] = 1 \ \& \ M_2[t][i.g, k.g] \cup$ 
 $\cup_{N[M_1[N][i.r, k.r]]} \text{GETPATHS}(i.g, k.g, N, C_3, M_1, M_2)$ 
 $\text{GETPINNER}(i, k, C_3, M_1, M_2)$ 
8:    $r \leftarrow \{(k.g, t, j.g) \mid M_1[t][k.r, j.r] = 1 \ \& \ M_2[t][k.g, j.g] \cup$ 
 $\cup_{N[M_1[N][k.r, j.r]]} \text{GETPATHS}(k.g, j.g, N, C_3, M_1, M_2)$ 
 $\text{GETPINNER}(k, j, C_3, M_1, M_2)$ 
9:   return  $l \cdot r$ 
10: function GETPINNER( $i, j, C_3, M_1, M_2$ )
11:    $parts \leftarrow \{k \mid C_3[i, k] = 1 \ \& \ C_3[k, j] = 1\}$ 
12:   return  $\bigcup_{k \in parts} \text{GETSUBPATHS}(i, j, k, C_3, M_1, M_2)$ 

```

Ideas and description.

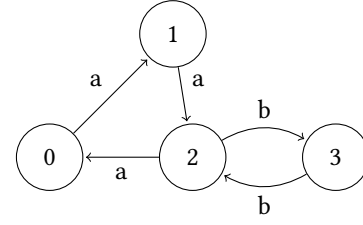
Correctness.

Time complexity.

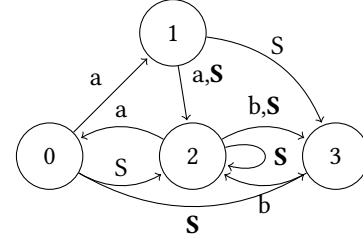
### 3.3 Example

Adjacency matrices  $M_1$  and  $M_2$  for automata  $R$  and graph  $\mathcal{G}$  respectively are initialized as follows:

$$M_1 = \begin{pmatrix} \cdot & \cdot & \{a\} & \cdot \\ \cdot & \cdot & \{S\} & \{b\} \\ \cdot & \cdot & \cdot & \{b\} \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}, M_2 = \begin{pmatrix} \cdot & \{a\} & \cdot & \cdot \\ \cdot & \cdot & \{a\} & \cdot \\ \{a\} & \cdot & \cdot & \{b\} \\ \cdot & \cdot & \{b\} & \cdot \end{pmatrix}.$$



(a) The input graph  $\mathcal{G}$



(b) The result graph  $\mathcal{G}$

Figure 1: The input and result graphs for example

Matrix  $M_1$  can be represented as a set of Boolean matrices as follows:

$$M_1^S = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{matrix}, M_1^a = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{matrix},$$

$$M_1^b = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{matrix}$$

Matrix  $M_2$  can be represented as a set of Boolean matrices as follows:

$$M_2^{S,0} = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{matrix}, M_2^{a,0} = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{matrix},$$

$$M_2^{b,0} = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & 1 & \cdot \end{pmatrix} \end{matrix}$$

First iteration.

$$M_3^1 = M_1^a \otimes M_2^{a,0} + M_1^b \otimes M_2^{b,0} + M_1^S \otimes M_2^{S,0} =$$

	(0,0)(0,1)(0,2)(0,3)	(1,0)(1,1)(1,2)(1,3)	(2,0)(2,1)(2,2)(2,3)	(3,0)(3,1)(3,2)(3,3)
(0,0)	.	.	1	.
(0,1)	.	.	1	.
(0,2)	.	1	.	.
(0,3)	.	.	.	.
(1,0)	.	.	.	.
(1,1)	.	.	.	.
(1,2)	.	.	.	1
(1,3)	.	.	.	1
(2,0)	.	.	.	.
(2,1)	.	.	.	.
(2,2)	.	.	.	1
(2,3)	.	.	1	.
(3,0)	.	.	.	.
(3,1)	.	.	.	.
(3,2)	.	.	.	.
(3,3)	.	.	.	.

$$C_3^1 = tc(M_3^1) =$$

	(0,0)(0,1)(0,2)(0,3)	(1,0)(1,1)(1,2)(1,3)	(2,0)(2,1)(2,2)(2,3)	(3,0)(3,1)(3,2)(3,3)
(0,0)	.	.	1	.
(0,1)	.	.	1	.
(0,2)	.	1	.	.
(0,3)	.	.	.	.
(1,0)	.	.	.	.
(1,1)	.	.	.	.
(1,2)	.	.	.	1
(1,3)	.	.	.	1
(2,0)	.	.	.	.
(2,1)	.	.	.	.
(2,2)	.	.	.	1
(2,3)	.	.	1	.
(3,0)	.	.	.	.
(3,1)	.	.	.	.
(3,2)	.	.	.	.
(3,3)	.	.	.	.

Second iteration.

$$M_3^2 = M_1^a \otimes M_2^{a,0} + M_1^b \otimes M_2^{b,0} + M_1^S \otimes M_2^{S,1} =$$

	(0,0)(0,1)(0,2)(0,3)	(1,0)(1,1)(1,2)(1,3)	(2,0)(2,1)(2,2)(2,3)	(3,0)(3,1)(3,2)(3,3)
(0,0)	.	.	1	.
(0,1)	.	.	1	.
(0,2)	.	1	.	.
(0,3)	.	.	.	.
(1,0)	.	.	.	.
(1,1)	.	.	.	.
(1,2)	.	.	1	.
(1,3)	.	.	.	1
(2,0)	.	.	.	.
(2,1)	.	.	.	.
(2,2)	.	.	.	1
(2,3)	.	.	1	.
(3,0)	.	.	.	.
(3,1)	.	.	.	.
(3,2)	.	.	.	.
(3,3)	.	.	.	.

$$C_3^2 = tc(M_3^2) =$$

	(0,0)(0,1)(0,2)(0,3)	(1,0)(1,1)(1,2)(1,3)	(2,0)(2,1)(2,2)(2,3)	(3,0)(3,1)(3,2)(3,3)
(0,0)	.	.	1	.
(0,1)	.	.	1	.
(0,2)	.	1	.	.
(0,3)	.	.	.	.
(1,0)	.	.	.	.
(1,1)	.	.	.	.
(1,2)	.	.	1	.
(1,3)	.	.	.	1
(2,0)	.	.	.	.
(2,1)	.	.	.	.
(2,2)	.	.	.	1
(2,3)	.	.	1	.
(3,0)	.	.	.	.
(3,1)	.	.	.	.
(3,2)	.	.	.	.
(3,3)	.	.	.	.

$$C_3^3 =$$

	(0,0)(0,1)(0,2)(0,3)	(1,0)(1,1)(1,2)(1,3)	(2,0)(2,1)(2,2)(2,3)	(3,0)(3,1)(3,2)(3,3)
(0,0)	.	.	1	.
(0,1)	.	.	1	.
(0,2)	.	1	.	.
(0,3)	.	.	.	.
(1,0)	.	.	.	.
(1,1)	.	.	.	.
(1,2)	.	.	1	.
(1,3)	.	.	.	1
(2,0)	.	.	.	.
(2,1)	.	.	.	.
(2,2)	.	.	.	1
(2,3)	.	.	1	.
(3,0)	.	.	.	.
(3,1)	.	.	.	.
(3,2)	.	.	.	.
(3,3)	.	.	.	.

$$C_3^4 =$$

	(0,0)(0,1)(0,2)(0,3)	(1,0)(1,1)(1,2)(1,3)	(2,0)(2,1)(2,2)(2,3)	(3,0)(3,1)(3,2)(3,3)
(0,0)	.	.	1	.
(0,1)	.	.	1	.
(0,2)	.	1	.	.
(0,3)	.	.	.	.
(1,0)	.	.	.	.
(1,1)	.	.	.	.
(1,2)	.	.	1	.
(1,3)	.	.	.	1
(2,0)	.	.	.	.
(2,1)	.	.	.	.
(2,2)	.	.	.	1
(2,3)	.	.	1	.
(3,0)	.	.	.	.
(3,1)	.	.	.	.
(3,2)	.	.	.	.
(3,3)	.	.	.	.

$$C_3^5 =$$

	(0,0)(0,1)(0,2)(0,3)	(1,0)(1,1)(1,2)(1,3)	(2,0)(2,1)(2,2)(2,3)	(3,0)(3,1)(3,2)(3,3)
(0,0)	.	.	1	.
(0,1)	.	.	1	.
(0,2)	.	1	.	.
(0,3)	.	.	.	.
(1,0)	.	.	.	.
(1,1)	.	.	1	.
(1,2)	.	.	1	.
(1,3)	.	.	.	1
(2,0)	.	.	.	.
(2,1)	.	.	.	.
(2,2)	.	.	.	1
(2,3)	.	.	1	.
(3,0)	.	.	.	.
(3,1)	.	.	.	.
(3,2)	.	.	.	.
(3,3)	.	.	.	.

$$C_3^6 =$$

	(0,0)(0,1)(0,2)(0,3)	(1,0)(1,1)(1,2)(1,3)	(2,0)(2,1)(2,2)(2,3)	(3,0)(3,1)(3,2)(3,3)
0:(0,0)	.	.	1	.
1:(0,1)	.	.	1	.
2:(0,2)	.	1	.	.
3:(0,3)	.	.	.	.
4:(1,0)	.	.	.	.
5:(1,1)	.	.	1	.
6:(1,2)	.	.	1	.
7:(1,3)	.	.	.	1
8:(2,0)	.	.	.	.
9:(2,1)	.	.	.	.
10:(2,2)	.	.	.	1
11:(2,3)	.	.	1	.
12:(2,0)	.	.	.	.
13:(2,1)	.	.	.	.
14:(2,2)	.	.	.	.
15:(2,3)	.	.	.	.

Reachability is done. Now we can to restore paths. Let we try to restore path from 2 to 2.

```

getPaths(2, 2, S, C36, M1, M20)
---getPInner(2, 14, C36, M1, M20)
---parts= {4}
---getSubpaths(2, 14, 4, C36, M1, M20)
---l={2  $\xrightarrow{a}$  0}
---getPInner(4, 14, C36, M1, M20)
---parts={11}
---getSubpaths(4, 14, 11, C36, M1, M20)
---getPaths(0, 3, S, C36, M1, M20)
---getPInner(0, 15, C36, M1, M20)
---parts={5, 10}
---getSubpaths(0, 15, 5, C36, M1, M20)
---l={0  $\xrightarrow{a}$  1}
---getPInner(5, 15, C36, M1, M20)
---parts={10}
---getSubpaths(5, 15, 10, C36, M1, M20)
---getPaths(1, 2, S, C36, M1, M20)
---getPInner(1, 14, C36, M1, M20)
---parts={6, 11}
---getSubpaths(1, 14, 6, C36, M1, M20)
---l={1  $\xrightarrow{a}$  2}
---getPInner(6, 14, C36, M1, M20)
---parts={11}
---getSubpaths(6, 14, 11, C36, M1, M20)
---getPaths(2, 3, S, C36, M1, M20)
---getPInner(2, 15, C36, M1, M20)
---parts={4, 10}

```



$$1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{a} 1 \} \cdot r_{\infty}^{1 \leadsto 3} \cdot \{3 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 2\})$$

```

getPaths(2, 2, S, C36, M1, M20)
├─ getPIInner(2, 14, C36, M1, M20)
│   └─ parts = {4}
│       └─ getSubpaths(2, 14, 4, C36, M1, M20)
│           └─ l = {2  $\xrightarrow{a}$  0}
│               └─ l · r = {0  $\xrightarrow{a}$  1  $\xrightarrow{a}$  2  $\xrightarrow{a}$  0  $\xrightarrow{a}$  1  $\xrightarrow{a}$  2  $\xrightarrow{b}$  3  $\xrightarrow{b}$ 
│                   2  $\xrightarrow{b}$  3  $\xrightarrow{b}$  2  $\xrightarrow{b}$  3} ∪ (0  $\xrightarrow{a}$  1  $\xrightarrow{a}$  2  $\xrightarrow{a}$  0  $\xrightarrow{a}$ 
│                   1} · r∞1↗3 · {3  $\xrightarrow{b}$  2  $\xrightarrow{b}$  3  $\xrightarrow{b}$  2  $\xrightarrow{b}$  3})}

```

## 4 IMPLEMENTATION DETAILS

Linear algebra, GraphBLAS.

Specific details. Sparsity parameters. How to express some steps efficiently.

Integration with RedisGraph.

Grammar is a file.

On paths extraction algorithm. I think that we should implement single path extraction, and paths without recursive calls. Lazy evaluation is not good idea for C implementation.

## 5 EVALUATION

Questions.

- (1) Compare classical RPQ algorithms and our algorithm
- (2) Compare other CFPQ algorithms and our algorithms
- (3) Investigate effect of grammar optimization

### 5.1 RPQ

Dataset description, tools selection.

5.1.1 *Dataset.* Dataset for evaluation

5.1.2 *Results.* Results of evaluation

### 5.2 CFPQ

Comparison with matrix-based.

5.2.1 *Dataset.* Dataset for evaluation

5.2.2 *Results.* Results of evaluation

### 5.3 Grammar transformation

On query optimization.

Memory aliases.

Synthetic???

## 6 RELATED WORK

CFPQ algorithms: Hellings [? ], Bradford [? ], Azimov [? ], Verbitskaya [? ], Ciro [? ], from static code analysis [? ],

RPQ algorithms: !!!! [? ]

Linear algebra based approaches to evaluate queries (data-log, SPARQL, etc) [? ] Not focused on types of queries. Query optimization.

## 7 CONCLUSION

On RSM optimization and query optimization.

HiCOO format.

GPGPU-based implementation. Multi-GPU version.

Full integration with Graph DB.

Other semantics: shortest path, simple path and so on.

Streaming graph querying.

Specialization on query.

!!!

## REFERENCES