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

Ekaterina Shemetova
katyacyfra@gmail.com
Saint Petersburg Academic University
St. Petersburg, Russia

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

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

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

SIGMOD'21, ,

ACM ISBN XXX-X-XXXXX-XXX-X…$15.00

CCS CONCEPTS

• **Information systems** → **Graph-based database models**; **Query languages for non-relational engines**; • **Theory of computation** → **Grammars and context-free languages**; **Regular languages**; • **Mathematics of computing** → **Paths and connectivity problems**; *Graph algorithms.*

ACM Reference Format:
Ekaterina Shemetova, Rustam Azimov, Egor Orachyov, Ilya Epelbaum, 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, 9 pages.

1 INTRODUCTION

Language-constrained path querying [?] is one of techniques for graph navigation querying. This technique allows one to use formal languages as constraints on paths in edge-labeled graphs: path satisfies constraints if labels along it form a word from the specified language.

The utilization of regular languages as constraints, or *Regular Path Querying* (RPQ), is most well-studied and widely spread. RPQs are used for .... Support of RPQs s implemented in !!! Even that, improvement of RPQ algorithm efficiency on huge graphs is an actual problem nowdays. For example, !!!!

At the same time, utilization of more powerful languages, namely context-free languages, gain popularity in the last few years. *Context-Free Path Querying* problem (CFPQ) was introduced by M Yannacacis in 1987 in [?]. A number of different algoritms was proposed since that time, Context-free is more specific, but actively developing last years.

To make it usable... Integration with graph DB. But recently, in [?] J Kujpers et al show that state-of-the-art CFPQ algorithms are not performant enoigh to be used in practice. This fact motivates to finde new algorithms for CFPQ.

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

CFPQ as a separated algorithms. Matrix is the fastest.

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

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

Recently, an algortithm was proposed. In this work we improve it, blah-blah-blah

Subcubic CFPQ. Long-time open problem. The best known result is !!!, Also it is shown by Chattergee that !!! For 1-Dyck language : Bradford [?]. Can not be generalized to arbitary CFPQ.s We find a way

Contribution

(1) New algorithm. Based on operation over Boolena matrices. All paths semantics. Previous matrix-based solution only single path. For both regualr and context-free path queries.
(2) Correctness and time complexity.
(3) Interconnection between CFPQ and dynamic transitive closure. Conjecture on sublinear dynamic transitive closure and subcubic CFPQ. We show that dynamic transitive closure is a bottleneck on the way to get subcubic CFPQ algorithm.
(4) Evaluation on real-world data. RPQ, CFPQ. Results show that !!!

2 PRELIMINARIES

In this section we introduce basic notation and definitions from graph theory and formal language theory which are used in our work.

2.1 Context-Free Path Querying Problem

We use a directed edge-labeled graph as a data model. To introduce *Context-Free Path Querying Problem (CFPQ)* over directed edge-labeled graphs we should introduce both graph and grammar definition.

First of all, we introduce edge-labelled diraph $\mathcal{G} = \langle V, E, L \rangle$, where V is a finite set of vertices, $E \in V \times L \times V$ is a finite set of edges, L is a finite set of edge labels. Note that one can always introduce bijection between V and $Q = \{0, \ldots, |V|-1\}$, thus in our work we gues that $V = \{0, \ldots, |V|-1\}$.

The example of a graph which we will use in further examples is presented in figure 1.

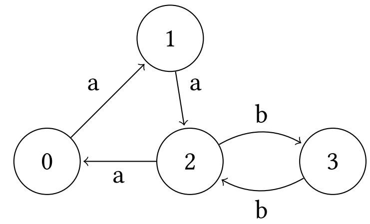

Figure 1: The example of input graph $\mathcal{G}$

Each edge-labeled graph can be representad as adjacency matrix M: square $|V| \times |V|$ matrix, such that $M[i,j] = \{l \mid e = (i,l,j) \in E\}$. Adjacency matrix M_2 of the graph $\mathcal{G}$ is

$$M_2 = \begin{pmatrix} . & \{a\} & . & . \\ . & . & \{a\} & . \\ \{a\} & . & . & \{b\} \\ . & . & \{b\} & . \end{pmatrix}.$$

In our work we use decomposition of the adjacency matrix to a set of Boolean matrices:

$$\mathcal{M} = \{M^l \mid l \in L, M^l[i,j] = 1 \iff l \in M[i,j]\}.$$

Matrix M_2 can be represented as a set of two Boolean matrices M_2^a and M_2^b where

$$M_2^a = \begin{pmatrix} . & 1 & . & . \\ . & . & 1 & . \\ 1 & . & . & . \\ . & . & . & . \end{pmatrix}, M_2^b = \begin{pmatrix} . & . & . & . \\ . & . & . & . \\ . & . & . & 1 \\ . & . & 1 & . \end{pmatrix} \quad (1)$$

This way we reduce operations which are necessary for our algorithm from operations over custom semiring (over edge labels) to operations over a Boolean semiring.

Also, we should define the path in the graph and word formed by the path.

Definition 2.1. Path π in the graph $(G) = \langle V, E, L \rangle$ is a seqence $e_0, e_1, \ldots, e_{n-1}$, where $e_i = (v_i, l_i, u_i) \in E$ and for any e_i, e_{i+1} $u_i = v_{i+1}$. We denote path from v to u as $v\pi u$.

Definition 2.2. The word formed by a path

$$\pi = (v_0, l_0, v_1), (v_1, l_1, v_2), \ldots, (v_{n-1}, l_{n-1}, v_n)$$

is a concatenation of labels along the path: $\omega(\pi) = l_0 l_1 \ldots l_{n-1}$.

The next part is a definitions from formal language theory.

Definition 2.3. Context-free grammar $G = \langle \Sigma, N, S, P \rangle$ where Σ is a finite set of terminals (or terminal alphabet), N is a finite set of nonterminals (or nonterminal alphabet), $S \in N$ is a start nonterminal, and P is a finite set of productions (grammar rules) of form $N_i \to \alpha$ where $N_i \in N$, $\alpha \in (\Sigma \cup N)^*$.

Definition 2.4. The sequence $\omega_2 \in (\Sigma \cup N)^*$ is dervable from $\omega_1 \in (\Sigma \cup N)^*$ in one derivation step, or $\omega_1 \to \omega_2$, in the grammar $G = \langle \Sigma, N, S, P \rangle$ iff $\omega_1 = \alpha N_i \beta$, $\omega_2 = \alpha \gamma \beta$, and $N_i \to \gamma \in P$.

Definition 2.5. Context-free grammar $G = \langle \Sigma, N, S, P \rangle$ specifies a *contex-free languege*: $\mathcal{L}(G) = \{\omega \mid S \xrightarrow{*} \omega\}$, where ($\xrightarrow{*}$) denotes zero or more derivation steps ($\to$).

Now we are ready to introduce CFPQ problem for the given graph $\mathcal{G} = \langle V, E, L \rangle$ and the given grammar $G = \langle \Sigma, N, S, P \rangle$ with reachability and all paths semantics (according Hellings [**?**]).

Definition 2.6. To evaluate context-free path query with reachability semantics is to construct a set of pairs of vertices (v_i, v_j) such that there exists a path $v_i \pi v_j$ in $\mathcal{G}$ which forms the word from the given language:

$$R = \{(v_i, v_j) \mid \exists \pi : v_i \pi v_j, \omega(\pi) \in L(G)\}$$

Definition 2.7. To evaluate context-free path query with all paths semantics is to construct a set of path π in $\mathcal{G}$ which forms the word from the given language:

$$\Pi = \{\pi \mid \omega(\pi) \in L(G)\}$$

Note that Π can be infinite, thus in practice, we should provide a way with reasonable complexity to enumerate such paths, instead of explicit construction of the Π.

2.2 Finite state machine

Definition 2.8. FSM

Regular expression to deterministic FSM.

Intersection of FSM is an RPQ. Regualr languages are clusude under intersection.

2.3 Recursive State Machines

Also known as recursive networks [**?**], recursive automata [**?**], !!!

Definition 2.9. RSM

Properties.

Grammar to RSM convertion algorithm. Eaxmple of convertion.

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

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

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

$$M_1^S = \begin{pmatrix} . & . & . & . \\ . & . & 1 & . \\ . & . & . & . \\ . & . & . & . \end{pmatrix}, \; M_1^a = \begin{pmatrix} . & . & 1 & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{pmatrix},$$

$$M_1^b = \begin{pmatrix} . & . & . & . \\ . & . & . & 1 \\ . & . & . & 1 \\ . & . & . & . \end{pmatrix}$$

Boolean decomposition of adjacency matrix

2.4 Graph Kronecker Product

Definition 2.10. Given two edge-labelled directed graphs $\mathcal{G}_1 = \langle V_1, E_1, L_1 \rangle$ and $\mathcal{G}_2 = \langle V_2, E_2, L_2 \rangle$ the Kronecker product of these two graphs is a edge-labeles directed graph $\mathcal{G} = \langle V, E, L \rangle$ where

- $V = V_1 \times V_2$
- $E = \{((u, v), l, (p, q)) \mid (u, l, p) \in E_1 \wedge (v, l, q) \in E_2\}$
- $L = L_1 \cap L_2$

$\mathcal{G}_1 \otimes \mathcal{G}_2$

Definition 2.11. Matrix tensor product definition. !!!!!

Tensot ptoduct of adjacency matrices. $M(G) = M(G_1) \otimes M(G_2)$

FSM intersection can be calcualeted as tensor product of FSM adjacency matrix.

Example!!!

Tensor product for FSM intersection over Boolean semiring using given definitions.

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 by always extracted by using standard techniques.

We describe both these steps, prove corrctness, and provide time complexity estimations. For the first step we firstly introduce naïve algortihm. Ather that we show how to achieve cubic tyme complexity by using dynamic trensitive closure algorithm and demonstrate that this technoque allow us to get truly subcubic CFPQ algortihm for planar graps.

After thet 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 [4].

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.

3.1.1 Naïve Version. 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 ε-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 ε-word. Here the following rule is implied: each vertex of the graph is reachable by itself through an ε-transition. Since the machine R does not have any ε-transitions, the ε-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, $s = q^0_B$, and $f \in F_B$, then $getNonterminals()$ returns the respective non-terminal. If the 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.

Listing 1 Kronecker product based CFPQ

```
function contextFreePathQuerying(G, 𝒢)
    R ← Recursive automata for G
    M1 ← Adjacency matrix for R
    M2 ← Adjacency matrix for 𝒢
    for s ∈ 0..dim(M1) − 1 do
        for i ∈ 0..dim(M2) − 1 do
            M2[i, i] ← M2[i, i] ∪ getNonterminals(R, s, s)
    while Matrix M2 is changing do
        M3 ← M1 ⊗ M2                                  ▷ Evaluate Kroncker product
        C3 ← transitiveClosure(M3)
        n ← dim(M3)                                   ▷ Matrix M3 size = n × n
        for (i, j) ∈ [0..n − 1] × [0..n − 1] do
            if C3[i, j] then
                s, f ← getStates(C3, i, j)
                if getNonterminals(R, s, f) ≠ ∅ then
                    x, y ← getCoordinates(C3, i, j)
                    M2[x, y] ← M2[x, y] ∪ getNonterminals(R, s, f)
    return M2
```

Listing 2 Help functions for Kronecker product based CFPQ

```
function getStates(C, i, j)
    r ← dim(M1)                        ▷ M1 is adjacency matrix for automata R
    return ⌊i/r⌋, ⌊j/r⌋
function getCoordinates(C, i, j)
    n ← dim(M2)                        ▷ M2 is adjacency matrix for graph 𝒢
    return i mod n, j mod n
```

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 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 ε-word, are also added at algorithm preprocessing step, since each vertex of the graph is reachable by itself through an ε-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.

□

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 leafs, 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 $\exists s\pi_1 f$ 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 π_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.

□

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.

□

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.

□

3.1.2 Application of Dynamic Transitive Closure. In this subsection we show how to reduce the time complexity of the Algorithm 1 by avoiding redundant calculations.

It is easy to see that the most time-consuming steps in the Algorithm 1 are the Kronecker product and transitive closure computations. Recall that the matrix M_2 is always changed in incremental manner i. e. elements (edges) are added to M_2 (and are never deleted from it) on every iteration of the Algorithm 1. So one does not need to recompute the whole product or transitive closure if an appropriate date structure is maintained.

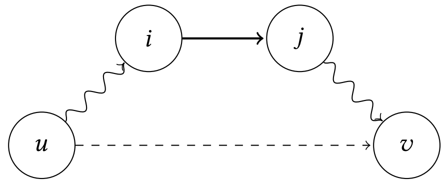

Figure 2: The vertex j become reachable from the vertex u after the addition of edge (i, j). Then the vertex v is reachable from u after inserting the edge (i, j) if v is reachable from j.

To deal with the Kronecker product computation, we use the left-distributivity of the Kronecker product. Let A_2 be a matrix with newly added elements and B_2 be a matrix with the all previously found elements, such that $M_2 = A_2 + B_2$. Then by the left-distributivity of the Kronecker product we have $M_1 \otimes M_2 = M_1 \otimes (A_2 + B_2) = M_1 \otimes A_2 + M_1 \otimes B_2$. Notice that $M_1 \otimes B_2$ is known and is already in the matrix M_3 and its transitive closure also is already in the matrix C_3, because it was calculated on the previous iterations, so it is left to update some elements of M_3 by computing $M_1 \otimes A_2$, which can be done in $O(|A_2||M_1|)$ time, where $|A|$ denotes the number of non-zero elements in a matrix A.

The fast computation of transitive closure can be obtained by using incremental dynamic transitive closure technique. We use an approach by Ibaraki and Katoh [5] to maintain dynamic transitive closure. The key idea of their algoritm is to recalculate reachability information only for those vertices, which become reachable after insertion of the certain edge (see Figure 2 for details). The algorithm is presented in Listing 3 (we have slightly modified it to efficiently track new elements of the matrix C_3).

Listing 3 The dynamic transitive closure procedure

```
1: function ADD(C3, i, j)
2:     n ← Number of rows in C3
3:     C3' ← Empty matrix
4:     for u ∈ 0...n | u ≠ j & C3[u, i] = 1 & C3[u, j] = 0 do
5:         for v ∈ 0...n do
6:             if C3[u, v] = 0 & C3[j, v] = 1 then
7:                 C3'[u, v] ← 1
8:     return C3'
```

Final version of the modified Algorithm 1 is shown in Listing 4.

Listing 4 Kronecker product based CFPQ using dynamic transitive closure

```
1: function CONTEXTFREEPATHQUERYING(G, 𝒢)
2:     R ← Recursive automata for G
3:     M1 ← Adjacency matrix for R
4:     M2 ← Adjacency matrix for 𝒢
5:     A2 ← Adjacency matrix for 𝒢
6:     C3 ← The empty matrix
7:     for s ∈ 0..dim(M1) − 1 do
8:         for i ∈ 0..dim(M2) − 1 do
9:             M2[i, i] ← M2[i, i] ∪ getNonterminals(R, s, s)
10:    while Matrix M2 is changing do
11:        M3' ← M1 ⊗ A2
12:        A2 ← The empty matrix of size n × n
13:        for M3'[i, j] | M3'[i, j] = 1 do
14:            C3[i, j] ← 1
15:            C3' ← ⋃_(i,j) add(C3, i, j)        ▷ Updating the transitive closure
16:            C3 ← C3 + C3'
17:        n ← dim(M3)
18:        for (i, j) ∈ [0..n − 1] × [0..n − 1] do
19:            if C3'[i, j] then
20:                s, f ← getStates(C3', i, j)
21:                if getNonterminals(R, s, f) ≠ ∅ then
22:                    x, y ← getCoordinates(C3', i, j)
23:                    M2[x, y] ← M2[x, y] ∪ getNonterminals(R, s, f)
24:                    A2[x, y] ← A2[x, y] ∪ getNonterminals(R, s, f)
25:    return M2
```

Theorem 3.5. *Let $\mathcal{G} = (V, E, L)$ be a graph and $G = (\Sigma, N, P)$ be a grammar. The Algorithm 4 calculates a result graph $\mathcal{G}_R = (V, E_R, L)$ in $O(n^3)$ time.*

Proof. Let $|A|$ be a number of non-zero elements in a matrix A. Consider the total time which is needed for computing the Kronecker products. The elements of the matrices $A_2^{(i)}$ are pairwise distinct on every i-th iteration of the Algorithm therefore we have $\sum_i T(M_1 \otimes A_2^{(i)}) = |M_1| \otimes \sum_i |A_2^{(i)}| = |M_1| O(n^2)$ operations in total.

Now we derive the time complexity of maintainig the dynamic transitive closure. Notice that C_3 has size of $O(n^2)$ so no more than $O(n^2)$ edges will be added during all iterations of the Algorithm. The condition in the line 4 in Listing 3 is calculated $O(n)$ times for every inserted edge (i, j). Thus we have $O(n^2 n) = O(n^3)$ operations in total. The operation from line 6 requires $O(n)$ time for a given vertex u. This operation is performed for every pair (j, v) of vertices such that a vertex j became reachable from the vertex u. There are no more than $O(n^2)$ such pairs, so line 6 will be executed at most $O(n^2 n) = O(n^3)$ times during the entire computation. Therefore $O(n^3)$ operations are performed to maintain dynamic transitive closure during all iteration of the Algorithm 4.

Notice that the matrix C_3' contains only new elements, therefore C_3 can be updated derectly using only $|C_3'|$ operations and hence $O(n^2)$ operations in total. The same holds for cycle in line 18 of the Algorithm 4, because operations are performed only for non-zero elements of the matrix $|C_3'|$. Finally, we have that the time complexity of the Algorithm 4 is $O(n^2) + O(n^3) + O(n^2) + O(n^2) = O(n^3)$. □

Notice that the obtained cubic time bound is close to the currently best known upper bound for the CFPQ evaluation (the asymptotically fastest known method has a complexity of $O(n^3/\log n)$) [2]. However it is open problem whether a truly sub-cubic algorithm exists for the CFL-reachability problem (and hence, for CFPQ evaluation) [1].

Subcubic for planar graphs using [7].

Cojecture on sublinear dynamic transitive closure and subcubic CFPQ.

3.2 Paths Extraction Algoritm

After index created one can enumerate all paths betwen cpecified vertices.

Listing 5 Paths extraction algorithm

$C_3 \leftarrow$ result of index creation algorithm: final transitive closure
$M_1 \leftarrow$ the set of adjacency matrices of the final graph
$M_2 \leftarrow$ the set of adjacency matrices of the input RSM
function $\textsc{getPaths}(v_s, v_f, N)$
  $s \leftarrow$ Start states of automata for N
  $f \leftarrow$ Final states of automata for N
  $res \leftarrow$ getPathsInner($getVNum(s, v_s)$, $getVNum(f, v_f)$)
  return res
function $\textsc{getSubpaths}(i, j, k)$
  $l \leftarrow \{(i.g, t, k.g) \mid M_1[t][i.r, k.r] = 1 \ \& \ M_2[t][i.g, k.g]\} \cup \bigcup_{N \mid M_1[N][i.r,k.r]} \textsc{getPaths}(i.g, k.g, N, C_3, M_1, M_2) \cup \textsc{getPathsInner}(i, k, C_3, M_1, M_2)$
  $r \leftarrow \{(k.g, t, j.g) \mid M_1[t][k.r, j.r] = 1 \ \& \ M_2[t][k.g, j.g]\} \cup \bigcup_{N \mid M_1[N][k.r,j.r]} \textsc{getPaths}(k.g, j.g, N, C_3, M_1, M_2) \cup \textsc{getPathsInner}(k, j, C_3, M_1, M_2)$
  return $l \cdot r$
function $\textsc{getPathsInner}(i, j)$
  $parts \leftarrow \{k \mid C_3[i, k] = 1 \ \& \ C_3[k, j] = 1\}$
  return $\bigcup_{k \in parts} \textsc{getSubpaths}(i, j, k, C_3, M_1, M_2)$

Ideas and description.

Correcness.

Time complexity of single path extraction. Paths enumeration complexity.

3.3 An example

In this section we introduce detailed example to demonstrate steps of the proposed algorithm. Our example is based on the classical worst case scenario introduced by Jelle Hellings in [?]. Namely, let we have a graph $\mathcal{G}$ presented in figure 1 and the RSM R presented in figure [?].

First step we represent graph as a set of boolean matrices as presented in 1, and RSM as a set of boolean matrices, as presented in **??**. Note, that we should add new empty matrix M_2^S to M_2. After that we should iteratively compute M_1 and C.

First iteration. As far as $M_2^{S,0}$ is empty (no edges with lable S in the input graph), then correspondent block od the Kronecker product will be empty.

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

$$
\begin{array}{cc}
 & \begin{array}{cccc|cccc|cccc|cccc} (0,0)&(0,1)&(0,2)&(0,3)&(1,0)&(1,1)&(1,2)&(1,3)&(2,0)&(2,1)&(2,2)&(2,3)&(3,0)&(3,1)&(3,2)&(3,3) \end{array} \\
\begin{array}{c} (0,0)\\(0,1)\\(0,2)\\(0,3)\\(1,0)\\(1,1)\\(1,2)\\(1,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3) \end{array} &
\left(\begin{array}{cccc|cccc|cccc|cccc}
.&.&.&.&.&1&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&1&.&.&.&.&.&.&.&.&. \\
.&.&.&.&1&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&.
\end{array}\right)
\end{array}
$$

Transitive closure calculation introduces one new path of length 2 (respective cell is filled).

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

$$
\begin{array}{cc}
 & \begin{array}{cccc|cccc|cccc|cccc} (0,0)&(0,1)&(0,2)&(0,3)&(1,0)&(1,1)&(1,2)&(1,3)&(2,0)&(2,1)&(2,2)&(2,3)&(3,0)&(3,1)&(3,2)&(3,3) \end{array} \\
\begin{array}{c} (0,0)\\(0,1)\\(0,2)\\(0,3)\\(1,0)\\(1,1)\\(1,2)\\(1,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3) \end{array} &
\left(\begin{array}{cccc|cccc|cccc|cccc}
.&.&.&.&.&1&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&1&.&.&.&.&.&.&.&.&\mathbf{1} \\
.&.&.&.&1&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&.
\end{array}\right)
\end{array}
$$

This path starts in the vertex $(0, 1)$ and finishes in the vertex $(3, 3)$. We can see, that 0 is a start state of RSM R and 3 is a final state of RSM R. Thus we can conclude that there esists a path between vertices 1 and 3 such that respective word is acceptable by R. As a result we can add the edge $(1, S, 3)$ to the $\mathcal{G}$, namely we should update the matrix M_2^S.

Second iteration. Modified input graph contains edge with lable S. From now !!!

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

$$
\begin{array}{cc}
 & \begin{array}{cccc|cccc|cccc|cccc} (0,0)&(0,1)&(0,2)&(0,3)&(1,0)&(1,1)&(1,2)&(1,3)&(2,0)&(2,1)&(2,2)&(2,3)&(3,0)&(3,1)&(3,2)&(3,3) \end{array} \\
\begin{array}{c} (0,0)\\(0,1)\\(0,2)\\(0,3)\\(1,0)\\(1,1)\\(1,2)\\(1,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3) \end{array} &
\left(\begin{array}{cccc|cccc|cccc|cccc}
.&.&.&.&.&1&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&1&.&.&.&.&.&.&.&.&. \\
.&.&.&.&1&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&\mathbf{1}&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&.
\end{array}\right)
\end{array}
$$

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

$$
\begin{array}{cc}
 & \begin{array}{cccc|cccc|cccc|cccc} (0,0)&(0,1)&(0,2)&(0,3)&(1,0)&(1,1)&(1,2)&(1,3)&(2,0)&(2,1)&(2,2)&(2,3)&(3,0)&(3,1)&(3,2)&(3,3) \end{array} \\
\begin{array}{c} (0,0)\\(0,1)\\(0,2)\\(0,3)\\(1,0)\\(1,1)\\(1,2)\\(1,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3) \end{array} &
\left(\begin{array}{cccc|cccc|cccc|cccc}
.&.&.&.&.&1&.&.&.&.&.&\mathbf{1}&.&.&\mathbf{1}&. \\
.&.&.&.&.&.&1&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&1&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&1&.&.&\mathbf{1}&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&.
\end{array}\right)
\end{array}
$$

$$C_3^3 =$$

$$
\begin{array}{cc}
 & \begin{array}{cccc|cccc|cccc|cccc} (0,0)&(0,1)&(0,2)&(0,3)&(1,0)&(1,1)&(1,2)&(1,3)&(2,0)&(2,1)&(2,2)&(2,3)&(3,0)&(3,1)&(3,2)&(3,3) \end{array} \\
\begin{array}{c} (0,0)\\(0,1)\\(0,2)\\(0,3)\\(1,0)\\(1,1)\\(1,2)\\(1,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3)\\(2,0)\\(2,1)\\(2,2)\\(2,3) \end{array} &
\left(\begin{array}{cccc|cccc|cccc|cccc}
.&.&.&.&.&1&.&.&.&.&.&1&.&.&1&. \\
.&.&.&.&.&.&1&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&1&.&.&.&.&.&\mathbf{1}&.&.&.&.&\mathbf{1} \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\ \hline
.&.&.&.&.&.&.&.&.&.&1&.&.&.&.&\mathbf{1} \\
.&.&.&.&.&.&.&.&.&.&.&1&.&.&1&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&1 \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&1&. \\ \hline
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&. \\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&.&.
\end{array}\right)
\end{array}
$$

$$C_3^4 = \begin{array}{cc}
 & \begin{array}{cccc|cccc|cccc|cccc} (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) \end{array} \\
\begin{array}{c} (0,0) \\ (0,1) \\ (0,2) \\ (0,3) \\ \hline (1,0) \\ (1,1) \\ (1,2) \\ (1,3) \\ \hline (2,0) \\ (2,1) \\ (2,2) \\ (2,3) \\ \hline (2,0) \\ (2,1) \\ (2,2) \\ (2,3) \end{array} &
\left(\begin{array}{cccc|cccc|cccc|cccc}
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \mathbf{1} & \cdot & \cdot & \mathbf{1} & 1 \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \mathbf{1} & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\hline
\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 & \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 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\hline
\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 & \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 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array}\right)
\end{array}$$

$$C_3^5 = \begin{array}{cc}
 & \begin{array}{cccc|cccc|cccc|cccc} (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) \end{array} \\
\begin{array}{c} (0,0) \\ (0,1) \\ (0,2) \\ (0,3) \\ \hline (1,0) \\ (1,1) \\ (1,2) \\ (1,3) \\ \hline (2,0) \\ (2,1) \\ (2,2) \\ (2,3) \\ \hline (2,0) \\ (2,1) \\ (2,2) \\ (2,3) \end{array} &
\left(\begin{array}{cccc|cccc|cccc|cccc}
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \mathbf{1} & 1 & \cdot & \cdot & 1 & \mathbf{1} \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \mathbf{1} \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\hline
\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 & \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 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\hline
\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 & \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 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array}\right)
\end{array}$$

$$C_3^6 = \begin{array}{cc}
 & \begin{array}{cccc|cccc|cccc|cccc} {}^{0}(0,0) & {}^{1}(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) \end{array} \\
\begin{array}{c} 0{:}(0,0) \\ 1{:}(0,1) \\ 2{:}(0,2) \\ 3{:}(0,3) \\ \hline 4{:}(1,0) \\ 5{:}(1,1) \\ 6{:}(1,2) \\ 7{:}(1,3) \\ \hline 8{:}(2,0) \\ 9{:}(2,1) \\ 10{:}(2,2) \\ 11{:}(2,3) \\ \hline 12{:}(2,0) \\ 13{:}(2,1) \\ 14{:}(2,2) \\ 15{:}(2,3) \end{array} &
\left(\begin{array}{cccc|cccc|cccc|cccc}
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \mathbf{1} & \cdot & \cdot & \mathbf{1} & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \mathbf{1} & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\hline
\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 & \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 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\hline
\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 & \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 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array}\right)
\end{array}$$

Result is presented in figure 3. New edges is added to the original graph.

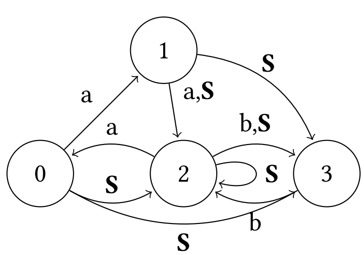

Figure 3: The result graph $\mathcal{G}$

Index creation is finished. Onw can use it to unswer reachcbility queryes, but for some problems it is necessary to restore paths. One can do it by using index created. Let for example we try to restore path from 2 to 2 derived from S.

To do it one should call `getPaths(2, 2, s)`. Partial trace of this call is presented below in figure 4. First, we convert vertices from grap to indeces in matrix and call `getPathsInner`.

getPaths(2, 2, S)
getPathsInner(2, 14)
parts= $\{4\}$
getSubpaths(2, 14, 4)
l=$\{2 \xrightarrow{a} 0\}$
...
getPathsInner(0, 14)
parts = $\{5, 11\}$
getSubpaths(0, 14, 5)
...
getPaths(1, 3, S)
...
getSubpaths(1,15,6)
$l = \{1 \xrightarrow{a} 2\}$
$r = \{2 \xrightarrow{b} 3\}$
return $\{1 \xrightarrow{a} 2 \xrightarrow{b} 3\}$
getSubpaths(0, 14, 11)
...
getPaths(1, 3, S) // An alternative way to get paths from 1 to 3 which leads to infinite set of paths
return $r_\infty^{1 \rightsquigarrow 3}$ // An infinite set of path from 1 to 3
...
return $\{0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{b} 2\} \cup (\{0 \xrightarrow{a} 1\} \cdot r_\infty^{1 \rightsquigarrow 3} \cdot \{3 \xrightarrow{b} 2\})$
return $\{2 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 2\} \cup (\{2 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{a} 1\} \cdot r_\infty^{1 \rightsquigarrow 3} \cdot \{3 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 2\})$

Figure 4: Example of call stack trace

Separation vertex `pats={4}` and try to get patrs of paths going throw vertex with $id = 4$.

Expected path is returned, other paths calcualtion

Lazy evaluation is required.

The paths enumeration problem is actual here: ho can we enumerate paths with small delay.

4 IMPLEMENATION DETAILS

Naïve algortihm is implemented (without dynamic transitive closure).

Linear algebra, GraphBLAS, parallel CPU.

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 shuold implement single path extraction, and paths without recursive calls. Lazy evalustion is not good idea for C implementation.

5 EVALUATION

Questions.

(1) Compare classical RPQ algorithms and our agorithm
(2) Compare other CFPQ algorithms and our algorithms
(3) Ivestigate effect of grammar optimization

5.1 RPQ

In oder to do smthng....

Dataset description, tools selection.

5.1.1 Dataset. Dtatset for evalustion

We evaluate our solution on RPQs We choose templates of the most popular RPQs which are presented in table **??** We generate !!! queryes for each template.

5.1.2 Results. Results of evalustion

Index creation.

Paths extraction

5.1.3 Conclusion.

5.2 CFPQ

Comparison with matrix-based algorithm.

5.2.1 Dataset. Dtatset for evalustion. It should be CFPQ_Data[1].

Same-generation queryes, memory aliases.

5.2.2 Results. Results of evaluation.

Index creation.

Paths extraction.

5.2.3 Conclusion.

5.3 Grammar transformation

On query optimization.

Memory aliases.

Synthetic???

6 RELATED WORK

Language constrained path querying is a whide area !!!!

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

RPQ algorithms: derivatives [**?**], Glushkov [**?**], etc.!!!! [**?**] distributed, not lnear algebra.

Subcubic CFPQ: Bradford, Chattergee, For trees — partial cases, RSM-s — 4 Russians method, Smth else?

Dynamic transitive closure [**?**] [**?**] [**?**] [**?**] algebraic, combinatorial, special graph types.

Implementation side. Linear algebra based approcges to evaluate queryes (datalog, SPARQL, etc) [**?**] Not focused ot types of queryes. Matrices, !!!! SPARQL !!!! Datalog !!!!

7 CONCLUSION AND FUTURE WORK

!!!! Was presented. Evaluation demonstartes that!!! The way to solve teoretical problem is provided.

Subcubic CFPQ in general case — sublinear transitive closure.

On RSM optimization and query optimization. RSM minimization, ther transformations.

We evaluate naïve implemantation. Try to use advanced algorithms for dynamic transotive closure [3].

HiCOO format [**?**] for distributed processing of huge graphs.

GPGPU-based implementation. Multi-GPU version. Unified memory, etc [**?**]

Full integration with Graph DB. For example, with RedisGraph. SuiteSparse as abse and success of matrix algorithm integration [**?**]

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

Streaming graph querying. Both regular [6] and context-free.

Specialization on query. Algebraic opersations specialization (partially static data in Haskell [**?**]). Runtime specialization (Posgres) [**?**]

[1] !!!!

REFERENCES

[1] Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. 2017. Optimal Dyck Reachability for Data-Dependence and Alias Analysis. *Proc. ACM Program. Lang.* 2, POPL, Article 30 (Dec. 2017), 30 pages. https://doi.org/10.1145/3158118

[2] Swarat Chaudhuri. 2008. Subcubic algorithms for recursive state machines. In *POPL '08*.

[3] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. 2020. Faster Fully Dynamic Transitive Closure in Practice. In *18th Symposium on Experimental Algorithms (SEA 2020).* http://eprints.cs.univie.ac.at/6345/

[4] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition).* Addison-Wesley Longman Publishing Co., Inc., USA.

[5] T. Ibaraki and N. Katoh. 1983. On-line computation of transitive closures of graphs. *Inform. Process. Lett.* 16, 2 (1983), 95 – 97. https://doi.org/10.1016/0020-0190(83)90033-9

[6] Anil Pacaci, Angela Bonifati, and M. Tamer Özsu. 2020. Regular Path Query Evaluation on Streaming Graphs. *ArXiv* abs/2004.02012 (2020).

[7] Sairam Subramanian. 1993. A fully dynamic data structure for reachability in planar digraphs. In *Algorithms—ESA '93*, Thomas Lengauer (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 372–383.