ContextFree Wars: The RedisGraph Strikes Back

Arseniy Terekhov
simpletondl@yandex.ru
Saint Petersburg State University
St. Petersburg, Russia

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

Vadim Abzalov
!!!@!!!
Saint Petersburg State University
St. Petersburg, Russia

Timur Zinnatulin
!!!@!!!
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

A long time ago in a galaxy far far away... Abstract is very abstract.

1 INTRODUCTION

Language-constrained path querying [2] is a way to find paths in edge-labeled graphs when constraints are formulated in terms of language which restrict words formed by paths: the word formed by path's labels concatenation should be in the specified language. This way is very natural for navigational queries in graph databases, and one of the most popular languages which are used for constraints is a regular language. But in some cases, regular languages are not expressive enough, as a result, context-free languages gain popularity. Constraints in the form of context-free languages, or context-free path querying (CFPQ), can be used for RDF analysis [11], biological data analysis [9], static code analysis [8, 12], and in other areas.

Big amount of research done on CFPQ, a number of CFPQ algorithms were proposed, but the application of context-free constraints for real-world data analysis faced with some problems problem. The first problem is a bad performance of proposed algorithms on real-world data, as was shown by Jochem Kuijpers et al. [5]. Moreover, there are no graph databases with full-stack support of CFPQ, the main effort was made in algorithms and their theoretical properties research. This fact hinders research of problems reducible to CFPQ, thus it hinders the development of new solutions for some problems. For example, recently graph segmentation in data provenance analysis was reduced to CFPQ [6], but authors faced the problem during the evaluation of the proposed approach: no one graph database support CFPQ.

In [1] Rustam Azimov propose a matrix-based algorithm for CFPQ. This algoritm is one of promising way to solve the first problem and provide appropriate solution for real-world data analysis, as was shown by Nikita Mishim et al. in [7] and Arseniy Terekhov et al. in [10]. But this algorithm always computes information (reachability facts or single path which satisfies constraints) for all pairs of vertices in the graph, namely it solves *all-pairs* problem. It is unreasonable for some real-world scenarios when one can provide a relatively small set of start vertices or even single start vertex.

While all-pairs context-free path querying is a classical problem that investigates in a number of works, there is no, in our knowledge, solutions for 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.

Also, we provide full-stack support of CFPQ for the RedisGraph[1] [3] graph database. We implement a Cypher query language extension[2] that allows one to express context-free constraints, and extend the RedisGraph to support this extension. In our knowledge, it is the first full-stack implementation of CFPQ.

To summarize, we make the following contribution in this paper.

(1) We modify 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 still based on linear algebra, hance it is simple to implementation and allows one to use high-performance libraries for implmementation.
(2) We evalute the proposed algorithm. Our evaluation shows that !!!
(3) We provide full-stack support of CFPQ by extending the RedisGraph graph database. To do it, we extend Cypher with syntax allows one to express context-free constraints, implement the proposed algorithm in a RedisGraph backend, and support new syntax in the RedisGraph query execution engine. Finally, evaluate the poposed solution.

[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.

2 PRELIMINARIES

In this section we introduce common definitions in graph theory and formal language theory which will be used in this paper. Also, we provide brief description of Azimov's algorithm which is used as a base of our solution.

2.1 Graphs

In this work we use edge-labelled digraph as a data model and define it as follows.

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

- V is a set of vertices
- E is a set of edges
- $\sigma \subseteq \Sigma$ is a set of labels, and a set of edges $E \subseteq V \times \sigma \times V$

An example of the graph is presented in figure 1.

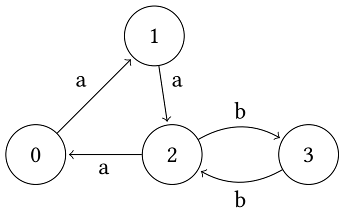

Figure 1: The example of input graph $\mathcal{G}$

We use adjacency matrix decomposed to a set of a boolean matrix as a representation of the graph.

Definition 2.2. An adjacency matrix M of the graph $\mathcal{G} =$ is a square $|V| \times |V|$ matrix, such that $M[i, j] = \{l \mid e = (i, l, j) \in E\}$.

Adjacency matrix M of the graph $\mathcal{G}$ is

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

Definition 2.3. Boolean decomposition of adjacency matrix M of graph $\mathcal{G} =$ is set of Boolean matrix

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

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

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

2.2 Languages

Definition 2.4. Context-free grammar is a 4-tuple $G = (N, \Sigma, R, S)$, where

- N is a set of nonterminals
- Σ is a set of terminals
- R is a finite set of productions of the followings form: $A \to \alpha, A \in N, \ \alpha \in (N \cup \Sigma)^*$
- S - a starting nonterminal

Definition 2.5. Context-free grammar $G = (N, \Sigma, R, S)$ is said to be in *Chomsky normal form* if all productions in R are of the form: $A \to BC$, $A \to a$, or $S \to \varepsilon$, where $A, B, C \in N, a \in \Sigma, \varepsilon$ is an empty string.

For example, we have the following context-free grammar:

$$S \to AB \mid a$$
$$B \to b$$

After transformation to Chomsky Normal Form the resulting grammar:

$$S \to a$$

This production itself is a grammar that has the same result, which is a, as original grammar.

We use a context-free grammar in Chomsky Normal Form that does not include a starting non-terminal, which will be specified in the path queries for the graph. Note that every context-free grammar can be transformed into an equivalent one in Chomsky Normal Form. Since only the empty paths correspond to empty string omitting the rules of the form $A \to \varepsilon$ does not restrict the applicability of the algorithms.

2.3 Matrix-Based Algorithm

Let $D = (V, E)$ be the input graph and $G = (N, \Sigma, P, S)$ be the input grammar. For a given graph and a context-free grammar, we define *context-free relations* $R_A \subseteq V \times V$ for every $A \in N$, such that $R_A = \{(n, m) \mid \exists n \pi m \ (l(\pi) \in L(G_A))\}$. For the context-free path query evaluation, we must provide such a path for each node pair from R_A.

The matrix-based algorithm for CFPQ can be expressed in terms of operations over Boolean matrices (see listing 1) which is an advantage for implementation.

Algorithm 1 Context-free path querying algorithm

```
function EVALCFPQ(D = (V, E), G = (N, Σ, P))
    n ← |V|
    T ← {T^{A_i} | A_i ∈ N, T^{A_i} is a matrix n × n, T^{A_i}_{k,l} ← false}
    for all (i, x, j) ∈ E, A_k | A_k → x ∈ P do T^{A_k}_{i,j} ← true
    for all A_k | A_k → ε ∈ P do
        for all i ∈ {0, . . . , n − 1} do T^{A_k}_{i,i} ← true
    while any matrix in T is changing do
        for A_i → A_j A_k ∈ P do T^{A_i} ← T^{A_i} + (T^{A_j} × T^{A_k})
    return T
```

This CFPQ algorithm allows efficiently apply GPGPU techniques, but it solves all-pairs problem and takes unreasonable amount of memory in scenarios in which we want to find paths from a relatively small set of vertices, since it calculates a lot of redundant information.

3 MATRIX-BASED MULTIPLE-SOURCE CFPQ ALGORITHM

In this section we introduce two versions of multiple-source matrix-based CFPQ algorithm. This algorithm is a modification of Azimov's matrix-based algorithm for CFPQ and the idea is that we cut off those vertices from which we are not interested in paths.

Let $D = (V, E)$ be the input graph, $G = (N, \Sigma, P, S)$ be the input grammar and Src be the input set of vertices. For the multiple-source context-free path query evaluation, we must provide such a path from R_A where the start node is from Src. In other words, for every $n \in Src$ we want to find all node pairs (n, m) such that $\exists n \pi m \ (l(\pi) \in L(G_A))$.

In order to solve the single-source and multiple-source CFPQ problem Azimov's algorithm was modified: operations of Boolean

Algorithm 2 Multiple-source context-free path querying algorithm

1: **function** MultiSrcCFPQ($D = (V, E), G = (N, \Sigma, P, S), Src$)
2: $\quad T \leftarrow \{T^A \mid A \in N, T^A \leftarrow \emptyset\}$ ▷ Matrix in which every element is $\emptyset$
3: $\quad TSrc \leftarrow \{TSrc^A \mid A \in N \setminus S, TSrc^A \leftarrow \emptyset\}$ ▷ Matrix for input vertices in which every element is $\emptyset$
4: $\quad$ **for all** $v \in Src$ **do** ▷ Input matrix initialization
5: $\quad\quad TSrc^S_{v,v} \leftarrow true$
6: $\quad$ **for all** $A \rightarrow x \in P$ **do** ▷ Simple rules initialization
7: $\quad\quad$ **for all** $(v, x, to) \in E$ **do**
8: $\quad\quad\quad T^A_{v,to} \leftarrow true$
9: $\quad$ **while** T *or* $TSrc$ *is changing* **do** ▷ Algorithm's body
10: $\quad\quad$ **for all** $A \rightarrow BC \in P$ **do**
11: $\quad\quad\quad M \leftarrow TSrc^A * T^B$
12: $\quad\quad\quad T^A \leftarrow T^A + M * T^C$
13: $\quad\quad\quad TSrc^B \leftarrow TSrc^B + TSrc^A$
14: $\quad\quad\quad TSrc^C \leftarrow TSrc^C +$ getDst(M)
15: $\quad$ **return** T
16:
17: **function** getDst(M)
18: $\quad A \leftarrow \emptyset$
19: $\quad$ **for all** $(v, to) \in V^2 \mid M_{v,to} = true$ **do**
20: $\quad\quad A_{to,to} \leftarrow true$
21: $\quad$ **return** A

Algorithm 3 Optimized multiple-source context-free path querying algorithm

1: **function** MultiSrcCFPQSmart($index = (D, G, T, TSrc), Src$)
2: $\quad TNewSrc \leftarrow \{TNewSrc^A \mid A \in N \setminus S, TNewSrc^A \leftarrow \emptyset\}$
3: $\quad$ **for all** $v \in Src \mid index.TSrc_{v,v} = false$ **do**
4: $\quad\quad TNewSrc^S_{v,v} \leftarrow true$
5: $\quad$ **while** $index.T$ *or* $TNewSrc$ *is changing* **do**
6: $\quad\quad$ **for all** $A \rightarrow BC \in P$ **do**
7: $\quad\quad\quad M \leftarrow TNewSrc^A * index.T^B$
8: $\quad\quad\quad index.T^A \leftarrow index.T^A + M * index.T^C$
9: $\quad\quad\quad TNewSrc^B \leftarrow TNewSrc^B + TNewSrc^A \setminus index.TSrc^B$
10: $\quad\quad\quad TNewSrc^C \leftarrow TNewSrc^C +$ getDst(M) $\setminus index.TSrc^C$

matrix multiplication $T_A = T_A + T_B \cdot T_C$ for each $A \rightarrow BC \in R$ represented in line **8** of Algorithm 1 was supplemented with one more matrix multiplication $T_A = T_A + (TSrc^A \cdot T_B) \cdot T_C$ for each $A \rightarrow BC \in R$ which saves only vertices we are interested in, where $TSrc^A$ — matrix of vertices to calculate the paths from. It is represented in lines **11-13** of the Algorithm 2. Also, after every iteration of while loop this is nessesary to update the set of vertices paths from which we need to calculate. To do this, the function **getDst**, represented in lines **17-21**, is called at line **14**. Thus, the modified algorithm does not calculate the paths from all vertices in case of query to calculate the paths small set of vertices.

We proposed the variant of the algorithm that can calculate the paths from a certain set of vertices, however there are such scenarios when queries are partially or completely repeated. In such cases it would be useful to add data caching to improve the performance. The problem is that every time we want to find all paths from the certain set of vertices, the Algorithm 2 calculates everything from scratch. Since recalculating might take the significant amount of time, we modified multiple-source CFPQ algorithm to specify it for such scenarios. This version stores all the vertices the paths from which have already been calculated in cash *index*, which is used to filter such vertices in line **3** of Algorithm 3. Thus, modified algorithm calculates paths from the particular vertex only once.

3.1 Implementation Details

All of the above versions have been implemented[3] using GraphBLAS framework that allows you to represent graphs as matrices and work with them in terms of linear algebra. For convenience, all the code is written in Python using pygraphblas[4], which is Python wrapper around GraphBLAS API and based on SuiteSparse:GraphBLAS[5] [4] — the full implementation of GraphBLAS standart. This library is specialized for working with sparse matrices, which most often appear in real graphs. Also, it should be noted that, despite the fact that the function **getDst** does not seem to be expressed in terms of linear algebra, the implementation used the function **reduce_vector** from pygraphblas that reduces matrix to a vector, with which further work takes place.

3.2 Algorithm Evaluation

And comparison. With combinators, GLL (.NET version).

Evaluation setup. Hardware basic description.

Graphs and queries from CFPQ_Data[6] Graphs and queries description: #V, #E, types of queries.

Tables.

Graphics (boxes). 1,2,4,8,16,32,50,100,500,1000,5000

Results.

Conclusion.

4 CFPQ FULL-STACK SUPPORT

In order to provide full-stack support of CFPQ it is necessatry to choose an appropriatr graph database. It was shown by Arseniy Terekhov et al. in [10] that matrix-based algorithm can be naturally integrated into RedisGraph graph database because both, the algorithm and the database, operates over matrix representation of graphs. Moreover, RedisGraph supports Cypher as a query language and there is a proposal which describes Cypher extension which allows one to specify context-free constraints. Thus we choose RedisGraph as a base for our solution.

4.1 Cypher Extending

The first what we should do is to extend Cypher to be able to express context-free constraints. There is a description of the respective Cypher syntax extension[7], proposed by Tobias Lindaaker, but this syntax does not implement yet in Cypher parsers.

This extension introduces path patterns, which are a more powerful alternative to relationship patterns. Path patterns allow

[3]GitHub repository with implemented algorithms: https://github.com/JetBrains-Research/CFPQ_PyAlgo, last accessed 28.08.2020

[4]GitHub repository of PyGraphBLAS library: https://github.com/michelp/pygraphblas

[5]GitHub repository of SuiteSparse:GraphBLAS library: https://github.com/DrTimothyAldenDavis/SuiteSparse

[6]!!!

[7]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.

you to express regular constrains over basic patterns such as relationship and node patterns. Just like relationship patterns they can be specified in the MATCH clause between the node patterns.

Listing 4 Example of using a simple path pattern

```
MATCH (v)-/ [:A (:X) :B] | [:C (:Y) :D] /->(to)
RETURN v, to
```

The listing 4 provides an example of query in extended syntax with a simple path pattern. In this example there are relationship patterns :A, :B, :C :D and node patterns (:X), (:Y). The square brackets are used for grouping parts of the pattern. The | symbol denotes alternative between corresponding paths and the white-space denotes sequence of paths. So the result of executing the query on the graph D will be the following set of vertex pairs:

$$\{(v, to) : \exists \pi = (v, r_1, u, r_2, to) \in Paths(D) : \\ \left[\begin{array}{l} t(r_1) = A, l(u) = X, t(r_2) = B \\ t(r_1) = C, l(u) = Y, l(r_2) = D \end{array}\right. \quad \}$$

Main feature which allows one to specify context-free constraints is a *named path patterns*: one can specify a name for path pattern and after that use it in other patterns, or in the same pattern. Using this feature, structure of query is pretty similar to context-free grammar in the Extended Backus–Naur Form.

Listing 5 Example of using a named path pattern

```
PATH PATTERN S = ()-/ [:A ~S :B] | [:A :B] /->()
MATCH (v)-/ ~S /->(to)
RETURN v, to
```

The listing 5 shows an example of using named path patterns. They can be defined in the PATH PATTERN clause and referenced within any other path pattern. In order to explain the semantics of the query let‘s consider contest-free grammar $G = (N, \Sigma, P, S)$ with $N = \{S\}$, $\Sigma = \{A, B\}$ and $P = \{S \to AB, S \to ASB\}$. Then $L(G) = \{A^n B^n : n \in \mathbb{N}\}$ specifies restrictions on the path labels and query result on the graph D will be as follows:

$$\{(v, to) : \exists \pi = (v, r_1, u_1, ..., r_n, to) \in Paths(D) : \\ t(r_1)t(r_2)...t(r_n) \in L(G)\}$$

Thus this Cypher extension allows one express more complex queries including context-free path queries. RedisGraph database supports subset of Cypher language and uses `libcypher-parser`[8] library to parse queries. We extend this library by intoducing new syntax proposed [7]. We implement[9] full extension, not only part which is necessary for simple CFPQ.

4.2 RedisGraph Extending

Named path patterns described in subsection 4.1 allows one to specify context-free constrains on the paths. In order to support the execution of these types of queries we need to extend back-end of the RedisGraph database and integrate a suitable CFPQ algorithm into it.

[8]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.

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

There are quite a few algorithms that solve CFPQ problem **??**, but their running time makes them unsuitable for practical use **??**. Recent studies **??** have shown that one can achieve high performance through the use of matrix-based algorithms. These studies were conducted to analyze the performance of the Rustam Azimov algorithm described in **??** and have shown that it is acceptable for practical application.

Using the Rustam Azimov algorithm one can only find paths between all pairs of vertexes at once and in some cases it is quite wasteful. Queries to graph databases can be specified so that when they are executed, it is required to find paths from a given set of initial vertices. This set can be quite small due to the different filtering specified in the query. For example in the listing 6 path pattern -/ ~S /-> follows pattern (v)-[r]->(u). The WHERE clause specifies some arbitrary predicate p(v, r, u) which also fixes a set of initial vertexes for a paths that must satisfy path pattern S. Depending on this predicate, this set of vertexes can have different sizes and for proper practical use the running time of the CFPQ algorithm should be sensitive to this.

Listing 6 ...

```
PATH PATTERN S = ()-/ :A [~S | ()] :B /->()
MATCH (v)-[r]->(u)-/ ~S /->(to)
WHERE p(v, r, u)
RETURN to
```

The Multi-Source algorithm described in **??** is sensitive to the initial set of vertices and is therefore well suited for graph database query scenarios. In addition, it is based on matrix operations and works with graphs as sparse matrices, so it is suitable for integration in RedisGraph.

4.3 Evaluation

Small basic evalustion on real-world graph (geo?). In order to show, that performance is reasonable.

Regular quries. Comparison with other DB?

5 CONCLUSION

In this paper we propose a number of multiple-source modifications of Azimov's CFPQ algorithm. Evaluation of the proposed modifications on the real-world examples shows that !!!! Finally, we provide the full-stack support of CFPQ. For our solution we implement corresponding Cypher extension as a part of `libcypher-parser`, integrate the proposed algorithm into RedisGraph, and extend RedisGraph execution plan builder to support extended Cypher queres. We demonstrate, that our solution allows one evaluate not only context-free queryes, but also regular one.

In the future, it is necessary to provide formal translation of Cypher to linear algebra, or find a maximal subset of Cypher which can be translated to linear algebra. There is a number of work on a subset of SPARQL to linear algebra translation, such as [?], but they are very limited. Deep investigation of this topic helps one to realize limits and restrictions of linear algebra utilization for graph databases. Moreover, it helps to improve existing solutions.

We show that evaluation of regular queryes is possible in practice by using CFPQ algorithm, as far as regular queries is a partial cas of the context-free one. But it seems, that the proposed solution is not optimal. For real-world solutions it is important to provide an optimal unified algorithm for both RPQ and CFPQ. One of possible way to solve this provlem is to use tensor-based algorithm [?].

Another imprtant task is to compare non-linear-algebra-based approaches to multiple-source CFPQ with the proposed solution. In [?] Johem Kujpers et.al. shows that all-pairs CFPQ algorithms implemented in Neo4j demonstrate unreasonable performance on real-world data for Neo4j. At the same time, Arseniy Terekhov et.al. shows that matrix-based all-pairs CFPQ algortihm implemented in appropriate linear algebra based graph database (RedisGraph) demonstrates good performance. But in the case of multiple-source scenario, when a number of sources is relatively small, non-linear-algebra-based solutions can be better, because such solutions naturally handle small reqired subgraph.

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] Timothy Davis. 2019. Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra. *ACM Trans. Math. Software* (2019). https://doi.org/10.1145/3322125

[5] 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

[6] 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.

[7] 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

[8] 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

[9] 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

[10] 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

[11] 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.

[12] 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