Context-Free Path Querying In Terms of Linear Algebra

Rustam Azimov
supervised by Semyon Grigorev
Saint Petersburg State University
St. Petersburg, Russia
st013567@student.spbu.ru

ABSTRACT

Context-Free Path Querying (CFPQ) is an important problem with applications in many areas, for example, graph databases, bioinformatics, static analysis, etc. Historically, regular languages are used as constraints for navigational path queries. However, in some important cases, regular languages are not expressive enough and context-free languages are used instead. Many algorithms for CFPQ were proposed but recently showed that the state-of-the-art CFPQ algorithms are still not performant enough for practical use. One promising way to achieve high-performance solutions for graph querying problems is to reduce them to linear algebra operations (such as matrix multiplication). The active utilization of these operations in the process of context-free path query evaluation makes it possible to efficiently apply a wide class of optimizations and computing techniques, such as GPGPU (General-Purpose computing on Graphics Processing Units), parallel computation, sparse matrix representation, distributed-memory computation, etc. In this Ph.D. work, we aim at: (i) studying the applicability of linear algebra methods to the CFPQ problem, (ii) at devising the algorithms for context-free path query evaluation formulated in terms of linear algebra operations, and (iii) at achieving high-performance implementations of the devised algorithms using parallel computations.

1 INTRODUCTION

Formal language-constrained path querying [2] is a graph analysis problem in which formal languages are used as constraints for navigational path queries. In this problem, a path in an edge-labeled graph is viewed as a word constructed by the concatenation of edge labels. The formal languages are used to constrain the paths of interest: a query should find only paths labeled by words from the language. The most popular class of constraints used as navigational queries in graph databases are the regular ones. However, in some important cases, regular languages are not expressive enough and context-free languages are used instead. The context-free path querying (CFPQ), can be used in many areas, for example, RDF analysis [15], static code analysis [9, 14], biological data analysis [12], graph segmentation [6].

CFPQ has been studied a lot since the problem was first stated by Mihalis Yannakakis in 1990 [21]. Jelle Hellings investigates various aspects of CFPQ in [16–18] and formulates three possible querying semantics: *relational* that requires to find all vertex pairs reachable by some path of interest, *single-path* query semantics also requires to return the example of such path for all vertex pairs, and *all-path* query semantics that requires to return all such paths for all vertex pairs.

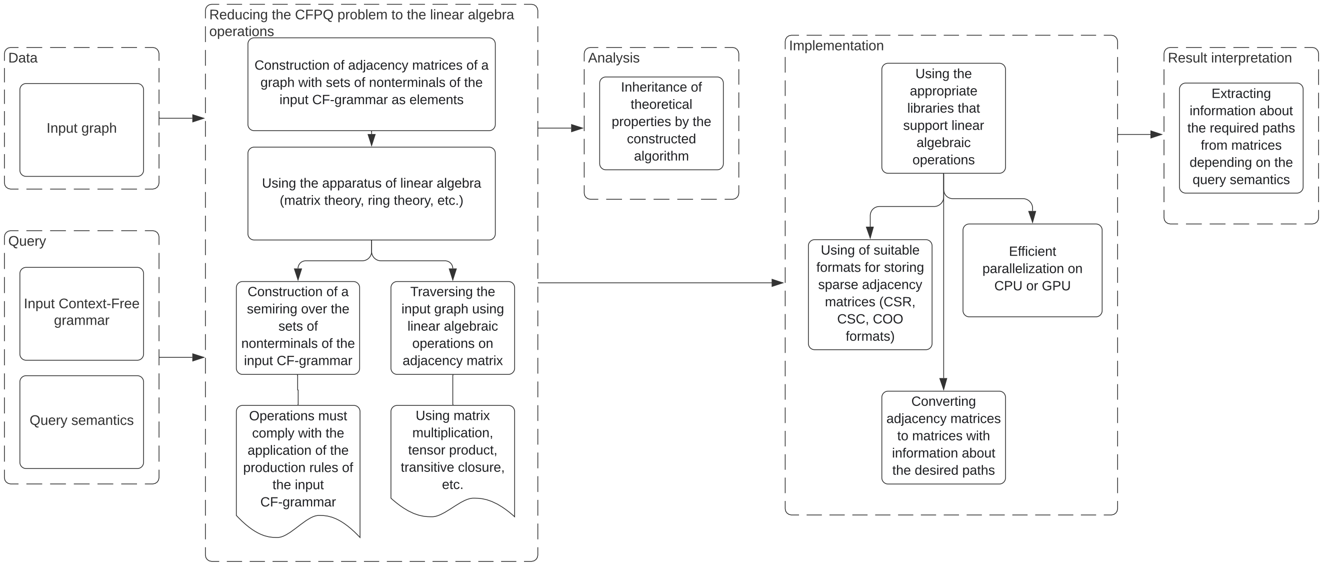
A number of CFPQ algorithms based on parsing techniques were proposed: (G)LL and (G)LR-based algorithms by Ciro M. Medeiros et al. [3], Fred C. Santos et al. [19], Semyon Grigorev et al. [13], and Ekaterina Verbitskaia et al. [4]; CYK-based algorithm by Xiaowang Zhang et al. [15]; combinators-based approach to CFPQ by Ekaterina Verbitskaia et al. [5]. Yet recent research by Jochem Kuijpers et al. [8] shows that existing solutions are not applicable for real-world graph analysis because of significant running time and memory consumption.

Inspired by Valiant's [20] matrix-based algorithm for context-free language recognition, we explore the applicability of linear algebra methods to the CFPQ problem. The linear algebra methods is widely used for various problems of finding paths in graphs, but the CFPQ problem poses additional challenges originating from query-specific information that needs to be captured. Valiant proposed a parsing algorithm, which computes a recognition table by computing matrix transitive closure. These algorithms take a string at the input and decide whether this string is generated from the input context-free grammar. Valiant's algorithm has essentially the same complexity as Boolean matrix multiplication. This is a promising way to achieve high-performance solutions for graph querying problems. The active utilization of these operations in the process of context-free path query evaluation makes it possible to efficiently apply a wide class of optimizations and computing techniques, such as GPGPU (General-Purpose computing on Graphics Processing Units), parallel computation, sparse matrix representation, distributed-memory computation, etc.

In this Ph.D. work, we make the following contributions.

(1) We provide an approach to solving the CFPQ problem using linear algebra operations. The provided approach allows us to use a wide class of optimizations of linear algebra operations for efficient analysis of large graphs.
(2) Using provided approach, we devise the CFPQ algorithms based on linear algebra operations for relational, single-path, and all-path query semantics.
(3) We provide the implementations of the devised algorithms for context-free path query evaluation using different optimizations and computing techniques. Our preliminary results demonstrate that our best CPU and GPU-based implementations that utilize sparse matrix representation and parallel computation outperform the state-of-the-art context-free path querying solutions.

Proceedings of the VLDB 2021 PhD Workshop, August 16th, 2021. Copenhagen, Denmark.



Data
Input graph
Query
Input Context-Free grammar
Query semantics
Reducing the CFPQ problem to the linear algebra operations
Construction of adjacency matrices of a graph with sets of nonterminals of the input CF-grammar as elements
Using the apparatus of linear algebra (matrix theory, ring theory, etc.)
Construction of a semiring over the sets of nonterminals of the input CF-grammar
Traversing the input graph using linear algebraic operations on adjacency matrix
Operations must comply with the application of the production rules of the input CF-grammar
Using matrix multiplication, tensor product, transitive closure, etc.
Analysis
Inheritance of theoretical properties by the constructed algorithm
Implementation
Using the appropriate libraries that support linear algebraic operations
Using of suitable formats for storing sparse adjacency matrices (CSR, CSC, COO formats)
Efficient parallelization on CPU or GPU
Converting adjacency matrices to matrices with information about the desired paths
Result interpretation
Extracting information about the required paths from matrices depending on the query semantics

Figure 1: An overview of our proposed approach to solving the CFPQ problem using linear algebra operations

2 APPROACH

In this section, we describe our approach to solving the CFPQ problem using linear algebra operations. The idea of using a sparse adjacency matrix as a graph representation in graph analysis problems is well-known. Recently, became very popular the GraphBLAS [7] API specification that defines standard building blocks for graph algorithms in the language of linear algebra. Various libraries that implement it provide data structures and functions to compute linear transformations and other linear algebra operations on sparse matrices. Using these libraries or other efficient libraries for such operations is a good recipe for making a high-performance CFPQ solution if we can reduce the CFPQ problem to linear algebra operations. Although such reduction was found for a number of graph algorithms, there are many graph algorithms for which it has not been done. To the best of our knowledge, the reduction of CFPQ problem to linear algebra operation is an open question.

An overview of our proposed approach is shown in Figure 1. The purpose of this approach is to solving CFPQ problem using linear algebra operations. In order to do this, it is necessary to devise the CFPQ algorithms that have the input in the form of a directed edge-labeled graph as a data, a context-free grammar as a query, and the query semantics that determines the type of requested information about paths in the graph. Further, the approach can be divided into the following stages.

Reducing the CFPQ problem to the linear algebra operations. The query is formulated by a *context-free grammar G* which is a tuple (N, Σ, P, S), where N is a finite set of nonterminals; Σ is a finite set of terminals, $N \cap \Sigma = \varnothing$; P is a finite set of productions of the form $A \rightarrow \alpha$, where $A \in N$, $\alpha \in (N \cup \Sigma)^*$; and S is the start nonterminal. To formulate the obtained problem in terms of linear algebra, the input graphs are considered in the form of adjacency matrices, and the input context-free grammar (CF-grammar) is encoded in the form of a certain algebraic structure, namely, in the form of a semiring with non-associative multiplication. This approach takes advantage of the fact that in the production rules of the CF-grammars there are two operations — concatenation and union, which are transformed into the product and the sum of a semiring. The elements of the adjacency matrices must be sets of nonterminals since each of them describe the set of words corresponding to the paths of interest. Further, the resulting semiring is used to define matrix multiplication (or other linear algebra operation), which simulates the step of the input graph traversing. For a complete traversal of the input graph, a transitive closure of adjacency matrices can be defined, which allows us to obtain the information about all paths corresponding to the query. The type of information retrieved depends on the query semantics, which must be taken into account when constructing the adjacency matrices and semirings.

Analysis. To analyze the theoretical properties of the constructed algorithm, the existing theoretical results of linear algebra, graph theory and the theory of formal languages can be used. The properties of the entire CFPQ algorithm largely depend on the properties of linear algebra operations since it evaluates the context-free path queries by offloading the most intensive computations into calls to procedures for these operations. Therefore, the most effective for the algorithm will be optimizations that use the existing results of linear algebra to efficiently compute such operations, for example, sparse matrix and vector operations.

Implementation. From a practical point of view, the algorithms built in this approach are easy to implement, since the most time consuming is the implementation of the necessary linear algebra operations, which have already been implemented in many efficient libraries that support linear algebra operations. For example, can be used the SuiteSparse:GraphBLAS [1] library — is an implementation of the GraphBLAS API. In such libraries, various matrix optimizations are used, which will significantly speed up the computation of the context-free queries for large graphs. One of such optimizations is the use of sparse matrix formats (CSR, CSC, COO), the use of which gives a significant performance gain since real data is often sparse. In addition, many linear algebra operations can be efficiently computed in parallel, for example on a GPU. As a result, the CFPQ implementation will allow to obtain matrices that will store information about the desired paths in the graph. The type of stored information is determined by the query semantics, for example, it can be an answer to the question of the existence of paths of a certain form or their enumeration.

Result interpretation. The final step is to interpret the query result. Depending on the query semantics, it is possible to extract certain information about the paths between the vertices of interest from the resulting matrix. In addition, in the case when the query semantics involves the enumeration of paths between certain vertices in the graph, it also becomes necessary to implement an algorithm for constructing these paths from the resulting matrices.

3 ALGORITHMS

In this section, we briefly describe our devised CFPQ algorithms based on the linear algebra operations. For all our algorithms we formally prove the correctness, termination, and time complexity bounds. Our algorithms can be divided into two groups.

Matrix-based algorithms. The algorithms in the first group utilize the Boolean matrix multiplication for relational query semantics and operate with more complex matrices for single-path and all-path query semantics. Also, these algorithms, like many existing ones, require the CF-grammar transformation to some normal form that allows us to encode one step of the input graph traversing into exactly one matrix multiplication since in this form we have only two nonterminals in the right-hand side of productions rules.

We define a binary operation $(\,\cdot\,)$ for arbitrary subsets N_1, N_2 of N with respect to a CF-grammar $G = (N, \Sigma, P)$ as

$$N_1 \cdot N_2 = \{A \mid \exists B \in N_1, \exists C \in N_2 \text{ such that } (A \to BC) \in P\}.$$

Using this binary operation as subset multiplication, and union as an addition, we can define a *matrix multiplication*, $a \times b = c$, where a and b are matrices of a suitable size, that have subsets of N as elements, as $c_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \cdot b_{k,j}$. Also, we use the element-wise union operation on matrices a and b with the same size: $a \cup b = c$, where $c_{i,j} = a_{i,j} \cup b_{i,j}$. Finally, we define the *transitive closure* of a square matrix a as $a^+ = a_+^{(1)} \cup a_+^{(2)} \cup \cdots$, where $a_+^{(1)} = a$ and $a_+^{(i)} = \bigcup_{j=1}^{i-1} a_+^{(j)} \times a_+^{(i-j)},\ i \geq 2$.

We can evaluate the context-free path queries with relational semantics by computing this transitive closure of an adjacency matrix T of input labeled graph with sets of nonterminals as elements where $A \in T_{i,j}$ only if there is $x \in \Sigma$ such that there is edge from vertex i to j labeled by x and $(A \to x) \in P$. However, described operations can be computed using several Boolean matrix multiplications and additions if we encode the matrix T by the $|N|$ Boolean matrices (one Boolean matrix for each nonterminal how it is done in the work of Valiant [20]).

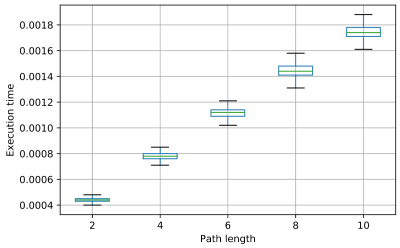


Figure 2: Execution time in seconds of the path extraction algorithm depending on the path length for *geospecies*

For single-path and all-path query semantics, we store additional information in matrices to be able to restore found paths. For single-path query semantics, we store the intermediate vertex k in the element $T_{i,j}$ only if there is a path from i to k corresponding to the nonterminal B, there is a path from k to j corresponding to the nonterminal C, and there is a rule $(A \to BC) \in P$. For all-path query semantics, we store the sets of the intermediate vertices, since we must store the information about all paths between each vertex pair. In that case, we cannot reduce computations to operations on Boolean matrices and we use custom matrix multiplication for matrices with more complex elements (tuples or arrays of integers). There are still libraries that support linear algebra operations for our algorithms for single-path and all-path query semantics, for example, the GraphBLAS implementations that support the creation of custom semirings for the matrix operations.

Kronecker product-based algorithm. On the contrary, the algorithm in the second group is based on the Kronecker product operation and does not require the transformation of the input grammar. The transformation leads to at least a quadratic blow-up in grammar size, thus by avoiding the transformation, this algorithm achieves better time complexity in terms of the grammar size. While regular languages can be expressed as a Finite-State Machine (FSM), a CF-grammar can be expressed as a Recursive State Machine (RSM) in a similar fashion. In these algorithms, we use RSM to represent the context-free path query and evaluate this query using the Kronecker product of the corresponding adjacency matrices of the input graph and the RSM for the input CF-grammar. Although the Kronecker-based algorithm for constructing the matrices with information about required paths (or so-called *index*) is the same for all three query semantics, the paths extraction algorithm for each semantics is different.

4 PRELIMINARY RESULTS

We evaluate implementations of devised algorithms on real-world RDFs. We provide results only for our CPU and GPU-based implementations of matrix-based algorithms for relational and single-path query semantics because of the page limit. We use RedisGraph [11] graph database as storage and measure the full time of query execution including all overhead on data preparation. This way we can estimate the applicability of the matrix-based algorithm to real-world problems. The source code is available on GitHub[1]

[1] Sources of matrix-based CFPQ algorithm for the RedisGraph database: https://github.com/YaccConstructor/RedisGraph. Access date: 27.02.2021.

Table 1: Index creation time for RDFs (time is measured in seconds and memory is measured in megabytes)

Name	V	E	Relational semantics index				Single-path semantics index			
			$\text{RG\_CPU}_{\text{rel}}$		$\text{RG\_GPU}_{\text{rel}}$		$\text{RG\_CPU}_{\text{path}}$		$\text{RG\_GPU}_{\text{path}}$	
			Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)
go-hierarchy	45 007	1 960 436	0.091	16.3	0.108	121.2	0.976	92.0	0.336	125.0
enzyme	48 815	219 390	0.018	5.9	0.018	4.0	0.029	8.1	0.043	6.0
eclass_514en	239 111	1 047 454	0.067	13.8	0.166	16.0	0.195	31.2	0.496	26.0
go	272 770	1 068 622	0.604	28.8	0.365	30.2	1.286	75.7	0.739	45.4
geospecies	450 609	2 311 461	7.146	16 934	0.856	5 274	15.134	35 803	1.935	5 282

The results of the CFPQ evaluation are presented in Table 1. As we can see, our matrix-based algorithm for relational query semantics implemented for RedisGraph is more than 1000 times faster than the one based on annotated grammar implemented for Neo4j [8] and uses more than 4 times less memory. We can conclude that the matrix-based algorithm is more performant than the state-of-the-art CFPQ algorithms for query evaluation under a relational semantics for real-world data processing.

We can conclude, that the cost of computing matrices for single-path query semantics is not high. On average, it is about 2 times slower than for the relational query semantics. The additional running time of the path extraction is presented in Figure 2 (boxplots are standard, outliers are omitted). As we can see, this time is small and linear in the length of the path.

Finally, we conclude that the matrix-based algorithm paired with a suitable database and employing appropriate libraries for linear algebra operations is a promising way to make CFPQ with different query semantics applicable for real-world data analysis.

5 CONCLUSION AND NEXT STEPS

In this Ph.D. work, we provide an approach to solving the CFPQ problem using linear algebra operations. Using provided approach we devise the CFPQ algorithms for all three query semantics and implement them using appropriate libraries for linear algebra operations. Preliminary results show that our CPU and GPU-based implementations that utilize sparse matrix representation and parallel computation outperform the state-of-the-art solutions.

As a next step, we plan to provide a full comparison of our linear algebra-based implementations with all state-of-the-art solutions for CFPQ on the same benchmark and experimental setup. Moreover, our implementations are prototypes and we plan to provide full integration of CFPQ to RedisGraph. Also, we plan to improve the dataset by including more real-world cases with larger graphs, for example, from the area of static code analysis [10, 14].

ACKNOWLEDGMENTS

This work was funded by RFBR, project number 19-37-90101, and grant from JetBrains Research.

REFERENCES

[1] Timothy A. Davis. 2019. Algorithm 1000: SuiteSparse: GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra. *ACM Trans. Math. Softw.* 45, 4 (2019), 44:1–44:25. https://doi.org/10.1145/3322125

[2] C. Barrett et al. 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] Ciro M. Medeiros et al. 2018. Efficient Evaluation of Context-free Path Queries for Graph Databases. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau, France) *(SAC '18)*. ACM, New York, NY, USA, 1230–1237. https://doi.org/10.1145/3167132.3167265

[4] Ekaterina Verbitskaia et al. 2016. Relaxed Parsing of Regular Approximations of String-Embedded Languages. In *Perspectives of System Informatics*, Manuel Mazzara and Andrei Voronkov (Eds.). Springer International Publishing, Cham, 291–302.

[5] Ekaterina Verbitskaia et al. 2018. Parser Combinators for Context-free Path Querying. In *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala* (St. Louis, MO, USA) *(Scala 2018)*. ACM, New York, NY, USA, 13–23. https://doi.org/10.1145/3241653.3241655

[6] H. Miao et al. 2019. Understanding Data Science Lifecycle Provenance via Graph Segmentation and Summarization. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 1710–1713.

[7] J. Kepner et al. 2016. Mathematical foundations of the GraphBLAS. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–9. https://doi.org/10.1109/HPEC.2016.7761646

[8] Jochem Kuijpers et al. 2019. An Experimental Study of Context-Free Path Query Evaluation Methods. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management* (Santa Cruz, CA, USA) *(SSDBM '19)*. ACM, New York, NY, USA, 121–132. https://doi.org/10.1145/3335783.3335791

[9] Jakob Rehof et al. 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

[10] Jyothi Vedurada et al. 2019. Batch Alias Analysis. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (San Diego, California) *(ASE '19)*. IEEE Press, 936–948. https://doi.org/10.1109/ASE.2019.00091

[11] P. Cailliau et al. 2019. RedisGraph GraphBLAS Enabled Graph Database. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 285–286. https://doi.org/10.1109/IPDPSW.2019.00054

[12] Petteri Sevon et al. 2008. Subgraph Queries by Context-free Grammars. *Journal of Integrative Bioinformatics* 5, 2 (2008), 157 – 172. https://doi.org/10.1515/jib-2008-100

[13] Semyon Grigorev et al. 2017. Context-free Path Querying with Structural Representation of Result. In *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia* (St. Petersburg, Russia) *(CEE-SECR '17)*. ACM, New York, NY, USA, Article 10, 7 pages. https://doi.org/10.1145/3166094.3166104

[14] Xin Zheng et al. 2008. Demand-driven Alias Analysis for C. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL '08)*. ACM, New York, NY, USA, 197–208. https://doi.org/10.1145/1328438.1328464

[15] Xiaowang Zhang et al. 2016. Context-Free Path Queries on RDF Graphs. In *The Semantic Web – ISWC 2016*. Springer International Publishing, Cham, 632–648.

[16] Jelle Hellings. 2014. Conjunctive context-free path queries. In *Proceedings of ICDT'14*. 119–130.

[17] Jelle Hellings. 2015. Path Results for Context-free Grammar Queries on Graphs. *CoRR* abs/1502.02242 (2015). arXiv:1502.02242 http://arxiv.org/abs/1502.02242

[18] Jelle Hellings. 2015. Querying for Paths in Graphs using Context-Free Path Queries. *arXiv preprint arXiv:1502.02242* (2015).

[19] Fred C. et al. Santos. 2018. A Bottom-Up Algorithm for Answering Context-Free Path Queries in Graph Databases. In *Web Engineering*, Tommi Mikkonen, Ralf Klamma, and Juan Hernández (Eds.). Springer International Publishing, Cham, 225–233.

[20] Leslie G Valiant. 1975. General context-free recognition in less than cubic time. *Journal of computer and system sciences* 10, 2 (1975), 308–315.

[21] Mihalis Yannakakis. 1990. Graph-theoretic Methods in Database Theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Nashville, Tennessee, USA) *(PODS '90)*. ACM, New York, NY, USA, 230–242. https://doi.org/10.1145/298514.298576