Context-Free Path Querying Can be Fast if Done Properly

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

Artyom Khoroshev
arthoroshev@gmail.com
ITMO 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 recent study showed that the applicability of context-free path querying (CFPQ) algorithms integrated with Neo4j database is limited because of low performance and high memory consumption. In this work we implement a matrix-based CFPQ algorithm by using appropriate high-performance libraries for linear algebra and integrate it with RedisGraph graph database. Our evaluation shows that the provided implementation is, in some cases, up to 1000 times faster than the best Neo4j-based one.

KEYWORDS

Context-free path querying, transitive closure, graph databases, linear algebra, context-free grammar, GPGPU, CUDA, boolean matrix, matrix multiplication

1 INTRODUCTION

Formal language constrained path querying, or formal language constrained path problem [4], is a graph analysis problem in which formal languages are used as constraints for navigational path queries. In this approsch a path is viewed as a word constructed by concatenation of edge labels. Paths of interest are constrained with some formal language: a query should find only paths labeled by words from the language. The class of language constraints which is most widely spread is regular: it is used in various graph query languages and engines. Context-free path queruing (CFPQ) [23], while being more expressive, is still at the early stage of development. Context-free constraints allow one to express such important class of queries as *same generation queies* [1] which cannot be expressed in terms of regular constraints.

Several algorithms for CFPQ based on such parsing techniques as (G)LL, (G)LR, and CYK were proposed recently [5, 6, 9, 11, 15, 17, 19, 21, 24]. Yet recent research by Jochem Kuijpers et.al. [14] shows that existing solutions are not applicable for real-world graph analysis because of significant running time and memory consumption. At the same time, Nikita Mishin et.al show in [16] that the matrix-based CFPQ algorithm demonstrates good performance on real-world data. A matrix-based algorithm proposed by Rustam Azimov [3] offloads the most critical computations onto boolean matrices multiplication. This algorithm is easy to implement and to employ modern massive-parallel hardware for CFPQ. The paper measures the performance of the algorithm in isolation while J. Kuijpers provides the evaluation of the algorithms which are integrated with Neo4j[1] graph database. Also, in [14] the matrix-based algorithm is implemented as a simple single-thread Java program, while N. Mishin shows that to achieve the best performance, one should utilize high-performance matrix multiplication libraries which are highly parallel or utilize GPGPU better. Thus, it is required to evaluate a matrix-based algorithm which is integrated with a graph storage and makes use of performant libraries and hardware.

In this work we show that CFPQ in relational semantics (according to Hellings [10]) can be performant enough to be applicable to real-world graph analysis. We use RedisGraph[2] [7] graph database as a storage. This database uses adjacency matrices as a representation of a graph and GraphBLAS [13] for matrices manipulation. These facts allow us to integrate matrix-based CFPQ algorithm with RedisGraph with minimal effort. We make the following contributions in this paper.

(1) We provide a number of implementations of the CFPQ algorithm which is based on matrix multiplication and uses RedisGraph as graph storage. The first implementation is CPU-based and utilizes SuteSparse[3] [8] implementation of GraphBLAS API for matrices manipulation. The second implementation is GPGPU-based and includes both the existing implementation from [16] and our own CUSP[4]-based implementation. The source code is available on GitHub[5].
(2) We extend the dataset presented in [16] with new real-world and synthetic cases of CFPQ[6].
(3) We provide evaluation which shows that matrix-based CFPQ implementation for RedisGraph database is performant enough for real-world data analysis.

2 MATRIX-BASED ALGORITHM FOR CFPQ

The matrix-based algorithm for CFPQ was proposed by Rustam Azimov [3]. This algorithm can be expressed in terms of operations over boolean matrices (see listing 1) which is an advantage for implementation.

Here $D = (V, E)$ is the input graph and $G = (N, \Sigma, P)$ is the input grammar. For each matrix T^{A_k} indexed with a nonterminal $A_k \in N$, a cell holds a true value ($T^{A_k}_{i,j} = \texttt{true}$) if and only if there exists $\pi = v_i \dots v_j$ — a path in D such that $A_k \overset{*}{\underset{G}{\Rightarrow}} \omega(\pi)$, where $\omega(\pi)$ is a word formed by the labels along the path π. Thus,

[1]Neo4j graph database web page: https://neo4j.com/. Access date: 12.11.2019.

[2]RedisGraph is a graph database which is based on Property Graph Model. Project web page: https://oss.redislabs.com/redisgraph/. Access date: 12.11.2019.

[3]SuteSparse is a sparse matrix software which incudes GraphBLAS API implementation. Project web page: http://faculty.cse.tamu.edu/davis/suitesparse.html. Access date: 12.11.2019.

[4]CUSP is an open source library for sparse matrix multiplication on GPGPU. Project site: https://cusplibrary.github.io/. Access date: 12.11.2019.

[5]Sources of matrix-based CFPQ algorithm for RedisGraph database: https://github.com/YaccConstructor/RedisGraph. Access date: 12.11.2019.

[6]The CFPQ_Data dataset fro CFPQ algorithms evaluation and comparison. GitHub page: https://github.com/JetBrains-Research/CFPQ_Data. Access date: 12.11.2019.

Listing 1 Context-free path quering algorithm

1: **function** EVALCFPQ($D = (V, E)$, $G = (N, \Sigma, P)$)
2: $\quad n \leftarrow |V|$
3: $\quad T \leftarrow \{T^{A_i} \mid A_i \in N, T^{A_i}$ is a matrix $n \times n$, $T^{A_i}_{k,l} \leftarrow \texttt{false}\}$
4: $\quad$ **for all** $(i, x, j) \in E$, $A_k \mid A_k \to x \in P$ **do** $T^{A_k}_{i,j} \leftarrow \texttt{true}$
5: $\quad$ **for** $A_k \mid A_k \to \varepsilon \in P$ **do** $T^{A_k}_{i,i} \leftarrow \texttt{true}$
6: $\quad$ **while** any matrix in T is changing **do**
7: $\quad\quad$ **for** $A_i \to A_j A_k \in P$ **do** $T^{A_i} \leftarrow T^{A_i} + (T^{A_j} \times T^{A_k})$
8: $\quad$ **return** T

this algorithm solves the reachability problem, or, according to Hellings [10], implements relational query semantics.

The performance-critical part of the algorithm is boolean matrix multiplication, thus one can achieve better performance by using libraries which efficiently multiply boolean matrices. There is also the following optimization: if the matrices T^{A_j} and T^{A_k} have not changed at the previous iteration, then we can skip the update operation in line 7. Data in real-world problems is often sparse, thus employing libraries which manipulate sparse matrices improves running time even more.

3 MATRIX BASED CFPQ FOR SINGLE PATH QUERYING

New on single path query semantics.

Theorems on correctness.

Complexity analysis.

3.1 Example?

4 IMPLEMENTATION

We showed that CFPQ can be naturally reduced to linear algebra. Linear algebra for graph problems is an actively developed area. One of the most important results is a GraphBLAS API which provides a way to operate over matrices and vectors over user-defined semirings.

Previous works show [3, 16] that existing linear algebra libraries utilization is the right way to achieve high-performance CFPQ implementation with minimal effort. But neither of these works provide an evaluation with data storage: algorithm execution time has been measured in isolation.

We provide a number of implementations of the matrix-based CFPQ algorithm. We use RedisGraph as storage and implement CFPQ as an extension by using the mechanism provided. Note that currently, we do not provide complete integration with the querying mechanism: one cannot use Cypher — a query language used in RedisGraph. Instead, a query should be provided explicitly as a file with grammar in Chomsky normal form. This is enough to evaluate querying algorithms and we plan to improve integration in the future to make our solution easier to use.

CPU-based implementation (RG_CPU) uses SuteSparse implementation of GraphBLAS, which is also used in RedisGraph, and a predefined boolean semiring. Thus we avoid data format issues: we use native RedisGraph representation of the adjacency matrix in our algorithm.

GPGPU-based implementation has two versions. The first one (**RG_M4RI**) uses the Method of Four Russians implemented in [16], and the second one (**RG_CUSP**) utilizes a modified CUSP library for matrix operations. Both implementations require matrix format conversion.

5 DATASET DESCRIPTION

In our evaluation we use combined dataset which contains the following parts.

- CFPQ_Data dataset which is provided in[7] [16] and contains both synthetic and real-world graphs and queries. Real-world data includes RDFs, synthetic cases include theoretical worst-case and random graphs.
- Dataset which is provided in [14]. Both Geospecies (RDF which contains information about biological hierrarchy[8] and same generation query over *broaderTransitive* relation), and Synthetic (the set of graphs generated by using the Barabási-Albert model [2] of scale-free networks and same generation query), are integrated with CFPQ_Data and used in our evaluation.
- It was shown in [16] that matrix-based algorithm is performant enough to handle bigger RDFs than those used in the initial datasets, such as [24]. So, we add a number of big RDFs to CFPQ_Data and use them in our evaluation. New RDFs: *go-hierarchy, go, enzime, core, pathways* are from UniProt database[9], and *eclass-514en* is from eClassOWL project[10].

The variants of the *same generation query* [1] are used in almost all cases because it is an important example of real-world queries that are context-free but not regular. So, variations of the same generation query are used in our evaluation. All queries are added to the CFPQ_Data dataset.

For RDFs (**[RDF]** dataset) we use two queries over *subClassOf* and *type* relations. The first query is the grammar G_1:

$$\begin{array}{ll} s \to subClassOf^{-1}\ s\ subClassOf & s \to type^{-1}\ s\ type \\ s \to subClassOf^{-1}\ subClassOf & s \to type^{-1}\ type \end{array}$$

The second one is the grammar G_2:

$$s \to subClassOf^{-1}\ s\ subClassOf \mid subClassOf$$

For geospecies and free scale graphs querying we use same-generation queries form the original paper.

6 EVALUATION AND DISCUSSION

We evaluate all the described implementations on all the datasets and the queries presented. We compare our implementations with [16] and [14]. We 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.

For evaluation, we use a PC with Ubuntu 18.04 installed. It has Intel core i7-6700 CPU, 3.4GHz, DDR4 32Gb RAM, and Geforce GTX 1070 GPGPU with 8Gb RAM.

The results of the evaluation are summarized in the tables below. We provide results only for a part of the collected dataset because of the page limit. Running time is measured in seconds, RAM memory consumption is measured in megabytes unless specified otherwise. Note that we provide results from the corresponding papers for all implementations except our own. The

[7]CFPQ_Data dataset GitHub repository: https://github.com/JetBrains-Research/CFPQ_Data. Access date: 12.11.2019.

[8]https://old.datahub.io/dataset/geospecies. Access date: 12.11.2019.

[9]Protein sequences data base: https://www.uniprot.org/. RDFs with data are avalable here: ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/rdf. Access date: 12.11.2019

[10]eClassOWL project: http://www.heppnetz.de/projects/eclassowl/. eclass-514en file is available here: http://www.ebusiness-unibw.org/ontologies/eclass/5.1.4/eclass_514en.owl. Access date: 12.11.2019.

cell is left blank if the time limit is exceeded, or if there is not enough memory to allocate the data.

The results of the first dataset **[RDF]** are presented in table 1. We can see that the running time of both CPU and GPGPU versions is small even for graphs with a big number of vertices and edges. The relatively small number of edges of interest may be the reason for such behavior. We believe it is necessary to extend the dataset with new queries which involve more different types of edges. Also, we can see, that *m4ri* version which uses dense bit matrices requires more memory. Thus we recommend to use sparse matrices on GPGPU.

Geospecies dataset currently can be processed only by using CPU version and we compare our matrix-based CPU implementation with the result form [14] for $AnnGram_{rel}$ algorithm[11]. Fortunately, both algorithms calculate queries under relational semantics. The result is provided in the table 2.

Table 2: Evaluation results on geospecies data

RG_CPU		Neo4j_AnnGram$_{rel}$	
Time	Memory (Gb)	Time	Memory (Gb)
6.8	6.83	6 953.9	29.17

As we can see, the matrix-based algorithm implemented for RedisGraph is more than 1000 times faster than the one based on annotated grammar implemented for Neo4j and uses more than 4 times less memory. We can conclude that the matrix-based algorithm is better than other CFPQ algorithms for query evaluation under a relational semantics for real-world data processing. CFPQ evaluation under other semantics (single path, all paths, etc) by using a matrix-based algorithm is a direction for future research.

The next is the **[FreeScale]** datatset. We compare our implementations with two implementations form [14] which evaluate queries under relatonal semantics: $Neo4j\_AnnGram_{rel}$ and *Neo4j_Matrix*. The results are presented in table 3. The evaluation shows that sparsity of graphs (value of parameter p) is important both for implementations which use sparse matrices and for implementations which use dense matrices. Note that the results for implementations for Neo4j are restored from graphics provided in [14]. So, values are not precise, but it is possible to compare implementations.

Evaluation shows that our CPU version is comparable with $Neo4j\_AnnGram_{rel}$ and for relatively dense graphs (each vertex has 10 connections) our implementation is faster. Moreover, while *Neo4j_Matrix* exceeded limits on the biggest graph, our implementation works fine. This demonstrantes the importance of using of appropriate libraries for matrix-based algorithm implementation. Also, we can see, that GPGPU version which utilizes sparse matrices is significantly faster than the other implementations. Note, that for GPGPU versions we include time required for data transferring and formats convertion.

Finally, we conclude that the matrix-based algorithm paired with a suitable database and employing appropriate libraries for linear algebra is a promising way to make CFPQ applicable for real-world data analysis. We show that SuiteSparse-based CPU implementation is performant enough to be comparable with GPGPU-based implementations on real-world data. It means that we can handle more complex data. We can also see, that more complex queries should be added to the dataset to make it more representable.

[11]Only *AnnGram* works correctly and fits limits, other implementations are faster, but either return an incorrect result or do not fit the memory .

7 CONCLUSION AND FUTURE WORK

We implemented a CPU and GPGPU based context-free path querying for RedisGraph and showed that CFPQ can be performant enough to analyze real-world data. However, our implementations are prototypes and we plan to provide full integration of CFPQ to RedisGraph. First of all, it is necessary to extend Cypher graph query language used in RedisGraph to support syntax for specification of context-free constraints. There is a proposal which describes such syntax extension[12] and we plan to support this syntax in libcypher-parser[13] used in RedisGraph.

Current version uses CUSP matrix multiplication library for GPGPU utilization, but it may be better to use GraphBLAST[14] [22] — Gunrock[15] [20] based implementation of GraphBLAS API for GPGPU. We plan to evaluate GraphBLAST based implementation of CFPQ and to investigate how multi-GPU support for GraphBLAST influences the performance of CFPQ in the case of processing huge real-world data.

Our implementations compute relational semantics of a query, but some problems require to find a path which satifies the constraints. To the best of our knowledge, there is no matrix-based algorithm for single path or all path semantics, thus we see it as a direction for future research.

Another important open question is how to update the query results dynamically when data changes. The mechanism for result updating allows one to recalculate query faster and use the result as an index for other queries.

Also, further improvements of the dataset are required. For example, it is necessary to include real-world cases from the area of static code analysis [12, 18, 25].

ACKNOWLEDGMENTS

The research was supported by the Russian Science Foundation grant 18-11-00100 and a grant from JetBrains Research.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. Foundations of Databases.
[2] Réka Albert and Albert lászló Barabási. [n.d.]. Statistical mechanics of complex networks. *Rev. Mod. Phys* ([n. d.]), 2002.
[3] 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
[4] Chris Barrett, Riko Jacob, and Madhav Marathe. 2000. Formal-language-constrained path problems. *SIAM J. Comput.* 30, 3 (2000), 809–837.
[5] Phillip G Bradford. 2007. Quickest path distances on context-free labeled graphs. In *Appear in 6-th WSEAS Conference on Computational Intelligence, Man-Machine Systems and Cybernetics*. Citeseer.
[6] Phillip G Bradford and Venkatesh Choppella. 2016. Fast point-to-point Dyck constrained shortest paths on a DAG. In *2016 IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 1–7.
[7] 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. https://doi.org/10.1109/IPDPSW.2019.00054

[12]Proposal with path pattern syntax for openCypher: https://github.com/thobe/openCypher/blob/rpq/cip/1.accepted/CIP2017-02-06-Path-Patterns.adoc. It is shown that context-free constraints can be expressed with the proposed syntax. Access date: 12.11.2019
[13]Web page of libcypher-parser project: http://cleishm.github.io/libcypher-parser/. Access date: 12.11.2019
[14]GraphBLAST project: https://github.com/gunrock/graphblast. Access date: 12.11.2019.
[15]Gunrock project: https://gunrock.github.io/docs/. Access date: 12.11.2019.

Table 1: RDFs querying results

RDF					Query G_1			Query G_2		
Name	#V	#E	#type	#subClassOf	RG_CPU	RG_M4RI	RG_CUSP	RG_CPU	RG_M4RI	RG_CUSP
funding	778	1480	304	90	0.01	<0.01	0.02	< 0.01	< 0.01	< 0.01
pizza	671	2604	365	259	0.01	<0.01	0.02	< 0.01	< 0.01	< 0.01
wine	733	2450	485	126	0.01	<0.01	0.02	< 0.01	< 0.01	< 0.01
core	1323	8684	1412	178	< 0.01	0.12	0.02	< 0.01	< 0.01	< 0.01
pathways	6238	37196	3118	3117	0.01	0.18	0.03	< 0.01	0.06	< 0.01
go-hierarchy	45007	1960436	0	490109	0.09	-	1.50	< 0.01	-	0.55
enzyme	48815	219390	14989	8163	0.02	61.23	0.10	< 0.01	6.97	0.02
eclass_514en	239111	1047454	72517	90962	0.06	-	0.39	0.01	-	0.10
go	272770	1068622	58483	90512	0.49	-	0.83	0.01	-	0.11

Table 3: Free scale graphs querying results

Graph	RG_CPU		RG_m4ri		RG_CUSP		Neo4j_AnnGram$_{rel}$		Neo4j_Matrix	
	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem
G(100,1)	< 0.01	< 0.01	< 0.01	0.10	0.01	2.00	< 0.02	0.08	0.20	0.03
G(100,3)	< 0.01	< 0.01	< 0.01	0.10	0.04	2.00	0.02	0.15	0.40	0.03
G(100,5)	< 0.01	< 0.01	< 0.01	0.10	0.05	2.00	0.03	0.21	0.40	0.03
G(100,10)	< 0.01	< 0.01	0.01	0.10	0.07	2.00	0.09	0.60	0.60	0.03
G(500,1)	< 0.01	< 0.01	< 0.01	2.00	0.01	2.00	< 0.02	0.20	20.00	0.60
G(500,3)	< 0.01	< 0.01	< 0.01	2.00	0.07	2.00	0.03	0.50	40.00	0.60
G(500,5)	< 0.01	0.17	< 0.01	2.00	0.10	2.00	0.10	1.10	50.00	0.60
G(500,10)	1.24	0.78	0.01	2.00	0.11	4.00	0.50	4.00	55.00	0.60
G(2500,1)	< 0.01	0.11	0.07	30.00	0.03	2.00	0.03	0.70	0.023	14.00
G(2500,3)	0.01	0.11	0.11	30.00	0.10	2.00	0.15	2.50	0.105	14.00
G(2500,5)	2.06	0.11	0.11	30.00	0.12	4.00	0.70	8.00	1.636	14.00
G(2500,10)	3.25	3.77	0.13	30.00	0.31	31.20	5.00	20.00	13.071	14.00
G(10000,1)	< 0.01	0.47	1.55	200.00	0.04	2.0	0.10	2.50	-	-
G(10000,3)	5.439	1.15	3.60	200.00	0.20	3.20	0.40	10.00	-	-
G(10000,5)	7.978	2.64	3.32	200.00	0.25	13.20	3.00	35.00	-	-
G(10000,10)	13.180	21.08	3.60	200.00	1.23	198.00	40.00	240.00	-	-

[8] Timothy A. Davis. 2018. Algorithm 9xx: SuiteSparse:GraphBLAS: graph algorithms in the language of sparse linear algebra.

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

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

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

[12] Nicholas Hollingum and Bernhard Scholz. 2017. Cauliflower: a Solver Generator for Context-Free Language Reachability. In *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning (EPiC Series in Computing)*, Thomas Eiter and David Sands (Eds.), Vol. 46. EasyChair, 171–180. https://doi.org/10.29007/tbm7

[13] J. Kepner, P. Aaltonen, D. Bader, A. Buluc, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, C. Yang, J. D. Owens, M. Zalewski, T. Mattson, and J. Moreira. 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

[14] 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

[15] Ciro M. Medeiros, Martin A. Musicante, and Umberto S. Costa. 2018. Efficient Evaluation of Context-free Path Queries for Graph Databases. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18)*. ACM, New York, NY, USA, 1230–1237. https://doi.org/10.1145/3167132.3167265

[16] 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

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

[18] Jyothi Vedurada and V Krishna Nandivada. [n.d.]. Batch Alias Analysis. ([n. d.]).

[19] Ekaterina Verbitskaia, Ilya Kirillov, Ilya Nozkin, and Semyon Grigorev. 2018. Parser Combinators for Context-free Path Querying. In *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala (Scala 2018)*. ACM, New York, NY, USA, 13–23. https://doi.org/10.1145/3241653.3241655

[20] Yangzihao Wang, Yuechao Pan, Andrew Davidson, Yuduo Wu, Carl Yang, Leyuan Wang, Muhammad Osama, Chenshan Yuan, Weitang Liu, Andy T. Riffel, and John D. Owens. 2017. Gunrock: GPU Graph Analytics. *ACM Trans. Parallel Comput.* 4, 1, Article 3 (Aug. 2017), 49 pages. https://doi.org/10.1145/3108140

[21] Charles B. Ward, Nathan M. Wiegand, and Phillip G. Bradford. 2008. A Distributed Context-Free Language Constrained Shortest Path Algorithm. In *Proceedings of the 2008 37th International Conference on Parallel Processing (ICPP '08)*. IEEE Computer Society, Washington, DC, USA, 373–380. https://doi.org/10.1109/ICPP.2008.67

[22] Carl Yang, Aydin Buluc, and John D. Owens. 2019. GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU. arXiv:cs.DC/1908.01407

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

[24] X. Zhang, Z. Feng, X. Wang, G. Rao, and W. Wu. 2016. Context-free path queries on RDF graphs. In *International Semantic Web Conference*. Springer, 632–648.

[25] Xin Zheng and Radu Rugina. 2008. Demand-driven Alias Analysis for C. *SIGPLAN Not.* 43, 1 (Jan. 2008), 197–208. https://doi.org/10.1145/1328897.1328464