Context-Free Path Querying Can be Fast if Cooked Properly

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

Artyom Khoroshev
!!!
!!!
St. Petersburg, Russia

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

ABSTRACT

Recently proposed matrix multiplication based algorithm for context-free path querying (CFPQ) offloads the most performance-critical parts onto boolean matrices multiplication. Thus, it is possible to achieve high performance of CFPQ by means of modern parallel hardware and software. In this paper, we provide results of empirical performance comparison of different implementations of this algorithm on both real-world data and synthetic data for the worst cases.

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 [3], is a graph analysis problem in which formal languages are used as a 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. While being more expressive context-free path queruing (CFPQ) [23] 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 can not be expressed in terms of regular constraints.

Several algorithms for CFPQ based on such parsing techniques as (G)LL, (G)LR, and CYK are proposed recently [4, 5, 9, 11, 15, 17, 19, 21, 24]. But 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 in [16] show that the matrix-based CFPQ algorithm demonstrates good performance on real-world data. Matrix-based algorithm is proposed by Rustam Azimov [2] and offloads the most critical computations onto boolean matrices multiplication. This algorithm is easy to implement, and employ modern massive-parallel hardware for CFPQ. The paper measures the performance of the algorithm, not integrating it with a graph storage, while J. Kuijpers provides an evaluation of algorithms which are integrated with Neo4j[1] graph database. Also, in [14] matrix-based algorithm is evaluated in simple single-thread Java implementation, while N. Mishin shows that the most efficient implementation should utilize high-performance matrix multiplication libraries which are highly parallel or better utilize GPGPU. Thus, evaluation of the matrix-based algorithm which is integrated with a graph storage and implemented in the appropriate way is required.

In this work we show that CFPQ in relational semantics (according to Hellings [10]) can be fast enough to be applicable to real-world graph analysis. We use RedisGraph[2] [6] 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 to RedisGraph with minimal effort. We make the following contributions in this paper.

(1) We provide a number of implementations of the matrix multiplication based CFPQ algorithm which uses RedisGraph as graph storage. The first implementation is CPU-based and utilizes SuteSparse[3] [7] 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 fast enough for real-world data analysis.

2 MATRIX-BASED ALGORITHM FOR CFPQ

The matrix-based algorithm for CFPQ was proposed by Rustam Azimov [2]. 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}, $T^{A_k}[i, j] = \texttt{true} \iff \exists \pi = v_i \dots v_j$—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, this algorithm solves the reachability problem, or, according to Hellings [10], implements relational query semantics.

[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 data set 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

function CONTEXTFREEPATHQUERYING(D, G)
  $n \leftarrow$ the number of nodes in D
  $E \leftarrow$ the directed edge-relation from D
  $P \leftarrow$ the set of production rules in G
  $N \leftarrow$ the set of nonterminals in G
  $T \leftarrow \{T^{A_i} \mid A_i \in N, T^{A_i}$ is a matrix $n \times n$ in which each element is `false`$\}$
  for all $(i, x, j) \in E$ **do** ▷ Matrix initialization
    for $A_k \mid A_k \rightarrow x \in P$ **do**
      $T^{A_k}_{i,j} \leftarrow$ `true`
  for $A_k \mid A_k \rightarrow \varepsilon \in P$ **do**
    $T^{A_k}_{i,i} \leftarrow$ `true`
  while any matrix in T is changing **do** ▷ Transitive closure calculation
    for $A_i \rightarrow A_j A_k \in P$ **do**
      $T^{A_i} \leftarrow T^{A_i} + (T^{A_j} \times T^{A_k})$
  return T

The performance-critical part of the algorithm is boolean matrix multiplication. Note, that if the matrices T_{N_j} and T_{N_k} have not been changed at the previous iteration, then we can skip update operation. Such optimization can improve performance. Also, it is important for applications that real-world data is often sparse, so it should be a better solution to use libraries which manipulate with sparse matrices.

3 IMPLEMENTATION

As we can see, 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 [2, 16] that existing linear algebra libraries utilization is the right way to get high-performance CFPQ implementation in minimal effort. But none of these works do not provide an evaluation with data storage, only pure time of algorithm execution was measured.

We provide a number of implementations of matrix-based CFPQ algorithm. All of them a based on RedisGraph — we use RedisGraph as a storage and implement CFPQ as an extension by using provided mechanism. Note that currently, we do not provide full integration with querying mechanism: one can not use Cypher, which uses in RedisGraph as a query language. Instead, in the current implementation query provided explicitly as a file with grammar in Chomsky normal form. So, we can evaluate querying algorithms, but we should improve integration to make our solution applicable.

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

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

4 DATASET DESCRIPTION

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

- CFPQ_Data dataset which provided in[7] [16] and contains both syntethic and real-world graphs and queryes. Real-world data includes RDFs, syntatic cases include theoretical worst-case and random graphs.
- Dataset which provided in [14]. We integrate both Geospecies — RDF which contains information about bioligical hierarchy[8] and same generation query over *broaderTransitive* relation, and Synthetic — the set of graph generated by using the Barabási-Albert model of scale-free networks and same generation quey, data into CFPQ_Data and use it in our evaluation.
- In [16] was shown that matrix-based algorithm is performad enough to handle bigger RDFs than used in initial data sets, 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* — parts of UniProt database[9], and *eclass-514en* that given from eClassOWL project[10].

The variants of the *same generation query* [1] is 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 is used in out evaluaton. All queryes are added to the CFPQ_Data data set.

For RDFs qurying we use two queryes over *subClassOf* and *type* relations. The first query is the grammar G_1:

$$s \rightarrow subClassOf^{-1}\ s\ subClassOf \qquad s \rightarrow type^{-1}\ s\ type$$
$$s \rightarrow subClassOf^{-1}\ subClassOf \qquad s \rightarrow type^{-1}\ type$$

The second one is the grammar G_2:

$$s \rightarrow subClassOf^{-1}\ s\ subClassOf \mid subClassOf.$$

5 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 full time of query execution, so all overhead on data preparing is incuded. Thus we can estimate applicability of matrix-based algrithm for real-world solutions.

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 part of the collected data set because of the page limit. Time is measured in seconds unless specified otherwise. Note that for all implementations except our own we provide results form related paper. The 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 in this case the running time of all our implementations is smaller than of the reference implementation, and all

[7] CFPQ_Data data set 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.

Table 1: RDFs querying results (time in milliseconds)

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

implementations but **[CuSprs]** demonstrate similar performance. It is obvious that performance improvement in comparison with the first implementation is huge and it is necessary to extend the dataset with new RDFs of the significantly bigger size.

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

Table 2: Evaluation results geospecies data

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

We can see, that the matrix-based algorithm implemented for RedisGraph is more than 1000 times faster than based on annotated grammar implemented for Neo4j and use more than 4 times less memory. Thus 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 implwmwntations form [14] which evaluate queryes under relatonal semantics: *Neo4j_AnnGram*$_{rel}$ and *Neo4j_Matrix*. 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 results for implemenattions for Neo4j are restored from graphics provided in [14]. So, values are not precize, but it is possible to compare implementations.

Evaluation shouws 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* is out of limits on biggest graph, our implementattion works fine. So, it s important ot use an appropriate libraries for matrix-based algorithm implementation. Also we can see, that GPGPU version which utilizes sparse matrices is significantly faster then other implementations. Note, that for GPGPU versions we include time requred for data transferring and formans convertion.

Finally, we can conclude that GPGPU utilization for CFPQ can significantly improve performance, but more research on advanced optimization techniques should be done. On the other hand, the high-level implementation (**[GPU_Py]**) is comparable with other GPGPU-based implementations. So, it may be a balance between implementation complexity and performance. Highly optimized existing libraries can be of some use: the implementation based on m4ri is faster than the reference implementation and the other CPU-based implementation. Moreover, it is comparable with some GPGPU-based implementations in some cases. Sparse matrices utilization demands more thorough investigation. The main question is if we can create an efficient implementation for sparse boolean matrices multiplication.

6 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, which is 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 proposed the syntax in libcypher-parser[13] which is 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. First of all, we plan to evaluate GraphBLAST based implementation of CFPQ. Also, we plan to investigate how multi-GPU support for GraphBLAST influences the performance of CFPQ in the case of processing huge real-world data.

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

[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 are expressible in 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 web page: https://gunrock.github.io/docs/. Access date: 12.11.2019.

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	-	-

Our implementations compute relational semantics of a query, but some problems require to find a path which satifies the constraints. Best to 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] 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

[3] Chris Barrett, Riko Jacob, and Madhav Marathe. 2000. Formal-language-constrained path problems. *SIAM J. Comput.* 30, 3 (2000), 809–837.

[4] 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.

[5] 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.

[6] 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

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

[8] Zhiwei Fan, Jianqiao Zhu, Zuyu Zhang, Aws Albarghouthi, Paraschos Koutris, and Jignesh Patel. 2018. Scaling-Up In-Memory Datalog Processing: Observations and Techniques. *arXiv preprint arXiv:1812.03975* (2018).

[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 *2008 37th International Conference on Parallel Processing*. IEEE, 373–380.

[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. ACM, 230–242.

[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