Context-Free Path Querying Can be Fast if Cooked Properly

Arseniy Terekhov
!!!
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 when formal languages are used as a constraints for navigational path queryes. Namely, in this approach concatenation of the labels along the path is treated as a word, and a specification of the language which should contain specific words is a constraint. While regular path querying (RPQ) is actively used in many different graph query languages and graph analysis engines, more expressive one, contxt-free path querying (CFPQ) [21] at the start stage and activly developed. Context-free constraints allow one to express such important class of queryes as *same generation quey* [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, 14, 16, 17, 19, 22]. But recent research by Jochem Kuijpers et.al. [13] shows that existing solutions are not applicable for real-world graph analysis because running time and memory consumption are poor. At the same time, Nikita Mishin et.al in [15] shows that matrix-based CFPQ algorithm can demonstrate 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. As a result, it is easy to implement, and allows one to utilize modern massive-parallel hardware for CFPQ. But only algorithm performance without integration with graph storage is provided in the paper while J. Kuijpers provides evaluation of algorithms which are integrated with Neo4j[1] graph database. Also, in [13] matrix-based algorithm is evaluated in simple single-thread Java implementation, while N. Mishin shows that the most efficient implementation should utilizes high-performance matrix multiplication libraries which are highly parallel, or even utilize GPGPU. Thus, evaluation of matrix-based algorithm which is integrated with graph storage and implemented in appropriate way is required.

In this work we show that CFPQ in relational semantics (according Hellings [10]) can be fast enough to be applicable to real-world graph analysis. We use RedisGraph[2] [6] garph data base as a storage. This data base use adjacency matrices as arepresenataion of graph and GraphBLAS [12] for matrices manipualtion. 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 a graph storage. The first implementation is CPU-based and utilize SuteSparse[3] [7] implemenation of GraphBLAS API for matrices manipulation. The second implementation is GPGPU-based and include both existing implementation from [15] and our own CUSP[4]-based implenetation. The source code is available on GitHub: https://github.com/YaccConstructor/RedisGraph.
(2) We extend the dataset presented in [15] with new real-world and synthetic cases of CFPQ[5].
(3) We provide evaluation which shows that matrix-based CFPQ implementaton for RedisGraph data base can de fast enough for real-forld data analysis.

2 MATRIX-BASED ALGORITHM FOR CFPQ

Matrix-based algorithm for CFPQ was proposed by Rustam Azimov [2]. This algorithm can be expressed in terms of operations over matrices boolean (see listing 1), and it is a sufficient 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_i}, $T^{A_i}[i, j] = \texttt{true} \iff \exists \pi = v_i \dots v_j$—path in D, such that $A_i \overset{*}{\underset{G}{\Rightarrow}} \omega(\pi)$, where $\omega(\pi)$ is a word formed by the labels along the path π. Thus, this algorithm solves reachability problem, or, according to Hellings [10], processes CFPQs by using relational query semantics.

The performance-critical part of the algorithm is matrix boolean multiplication. Note, that we can apply the next optimization: we can skip update if the matrices T_{N_j} and T_{N_k} have not been

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

[2] RedisGraph is a graph data base which is based on Property Graph Model. Project web page: https://oss.redislabs.com/redisgraph/. Access data: 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 data: 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] The CFPQ_Data data set fro CFPQ algorithms evaluation and comparison. GitHub page: https://github.com/JetBrains-Research/CFPQ_Data. Access data: 12.11.2019.

Listing 1 Context-free path quering algorithm

```
1: function contextFreePathQuerying(D, G)
2:     n ← the number of nodes in D
3:     E ← the directed edge-relation from D
4:     P ← the set of production rules in G
5:     N ← the set of nonterminals in G
6:     T ← {T^{A_i} | A_i ∈ N, T^{A_i} is a matrix n × n in which each
       element is false}
7:     for all (i, x, j) ∈ E do                    ▷ Matrix initialization
8:         for A_k | A_k → x ∈ P do
9:             T^{A_k}_{i,j} ← true
10:    for A_i | A_i → ε ∈ P do
11:        T^{A_i}_{i,i} ← true
12:    while any matrix in T is changing do        ▷ Transitive
       closure calculation
13:        for A_i → A_j A_k ∈ P do
14:            T^{A_i} ← T^{A_i} + (T^{A_j} × T^{A_k})
15:    return T
```

changed at the previous iteration. Also, it is important for applications that real-world data is almost sparse, so it should be better solution to use linraries which manipulates with sparse matrices.

As we can see, CFPQ can be naturally reduced to linead algebra. Linear algebra for graph problems is an actively developed area. One of the most important result is a GraphBLAS API which provides a way to operate over matrices and vectors over user-defined semirings. In this paper we use SuteSparse implementation of GraphBLAS and boolena semiring. All our implementations are based on the optimized version of the algorithm.

3 IMPLEMENTATION

Previous works show [2, 15] that existing linear algebra libraries utilization is a right way to get high-performance CFPQ implementation in minimal effort. But non of these works do not provide evaluation with data storage, only pure time of algorithm execution was mesured.

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 thet currenly we do not provide full integration with querying mechanism: one can not use Cypher, which uses in RedisGraph as a query language. Instead, in current omplementation query provided explicetely as an 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. Details on CPU implemenattion

GPGPU-based implementation. Details on GPGPU implemenattion

4 DATASET DESCRIPTION

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

- CFPQ_Data dataset which provided in[6] [15] 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 [13]. We integrate both Geospecies and Synthetic data sets into CFPQ_Data and use it in our evaluation.
- New bigger RDFs, such as !!!!. In [15] was shown that matrix-based algorithm is performad enough to handle bigger RDFs than used in initial data sets, such as [22]. So, we add a number of big RDFs to CFPQ_Data and use them in our evaluation.

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.

[RDF] The set of the real-world RDF files (ontologies) from [22] and two variants of the same generation query which describes hierarchy analysis. The first query is the grammar G_4:

$$s \to SCOR\ s\ SCO \qquad s \to TR\ s\ T$$
$$s \to SCOR\ SCO \qquad s \to TR\ T$$

The second one is the grammar G_5: $s \to SCOR\ s\ SCO \mid SCO$.

[Worst] The theoretical worst case for CFPQ time complexity proposed by Hellings [11]: the graph is two cycles of coprime lengths with a single common vertex. The first cycle is labeled by the open bracket and the second cycle is labeled by the close bracket. Query is a grammar for the A^nB^n language. The example of such graph and grammar is presented in figure:

$$s \to A\ s\ B$$
$$s \to A\ B$$

[Full] The case when the input graph is sparse, but the result is a full graph. Such a case may be hard for sparse matrices representations. As an input graph, we use a cycle, all edges of which are labeled by the same token. As a query we use two grammars which describe the sequence of tokens of arbitrary length: the simple ambiguous grammar G_2: $s \to s\ s \mid A$, and the highly ambiguous grammar G_3: $s \to s\ s\ s \mid A$.

[Sparse] Sparse graphs from [8] are generated by the GTgraph graph generator, and emulate realistic sparse data. Names of these graphs have the form `Gn-p`, where `n` corresponds to the total number of vertices, and `p` is the probability that some pair of vertices is connected. The query is the same generation query represented by the grammar G_1 (figure **??**).

5 EVALUATION

We evaluate all the described implementations on all the datasets and the queries presented. We compare our implementations with [2]. We exclude the time required to load data from files. The time required for data transfer is included.

For evaluation, we use a PC with Ubuntu 18.04 installed. It has Intel core i7 8700k 3,7HGz CPU, DDR4 32 Gb RAM, and Geforce 1080Ti GPGPU with 11Gb RAM.

The results of the evaluation are summarized in the tables below. Time is measured in seconds unless specified otherwise. The result for each algorithm is averaged over 10 runs. The cell is left blank if the time limit is exceeded, or if there is not enough memory to allocate the data.

[6] CFPQ_Data data set GitHub repository: https://github.com/JetBrains-Research/CFPQ_Data. Access date: 12.11.2019.

Table 1: RDFs querying results (time in milliseconds)

RDF			Query G_4						Query G_5					
Name	#V	#E	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
atm-prim	291	685	3	2	2	1	5	269	1	< 1	1	< 1	2	267
biomed	341	711	3	5	2	1	5	283	4	< 1	1	< 1	5	280
foaf	256	815	2	9	2	< 1	5	270	1	< 1	1	< 1	2	263
funding	778	1480	4	7	4	1	5	279	2	< 1	3	< 1	4	274
generations	129	351	3	3	2	< 1	5	273	1	< 1	1	< 1	2	263
people_pets	337	834	3	3	3	1	7	284	1	< 1	1	< 1	3	277
pizza	671	2604	6	8	3	1	6	292	2	< 1	2	< 1	5	278
skos	144	323	2	4	2	< 1	5	273	< 1	< 1	1	< 1	2	265
travel	131	397	3	5	2	< 1	6	268	1	< 1	1	< 1	3	271
unv-bnch	179	413	2	4	2	< 1	5	266	1	< 1	1	< 1	3	266
wine	733	2450	7	6	4	1	7	294	1	< 1	3	< 1	3	281

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 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.

Table 2: Evaluation results for the worst case

#V	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
16	0.032	< 1	0.008	0.002	0.027	0.309
32	0.118	0.001	0.034	0.008	0.136	0.441
64	0.476	0.041	0.133	0.032	0.524	0.988
128	2.194	0.226	0.562	0.129	2.751	3.470
256	15.299	1.994	3.088	0.544	11.883	15.317
512	121.287	23.204	13.685	2.499	43.563	102.269
1024	1593.284	528.521	88.064	19.357	217.326	1122.055
2048	-	-	-	325.174	-	-

Results of the theoretical worst case (**[Worst]** datatset) are presented in table 2. This case is really hard to process: even for a graph of 1024 vertices, the query evaluation time is greater than 10 seconds even for the most performant implementation. We can see, that the running time grows too fast with the number of vertices.

Table 3: Sparse graphs querying results

Graph	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
G5k-0.001	10.352	0.647	0.113	0.041	0.216	5.729
G10k-0.001	37.286	2.395	0.435	0.215	1.331	35.937
G10k-0.01	97.607	1.455	0.273	0.138	0.763	47.525
G10k-0.1	601.182	1.050	0.223	0.114	0.859	395.393
G20k-0.001	150.774	11.025	1.842	1.274	6.180	-
G40k-0.001	-	97.841	11.663	8.393	37.821	-
G80k-0.001	-	1142.959	88.366	65.886	-	-

The next is the **[Sparse]** datatset 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 behavior of the sparse matrices based implementation is as expected, but for dense matrices we can see, that more sparse graphs are processed faster. Reasons for such behavior demand further investigation. Note that we estimate only the query execution time, so it is hard to compare our results with the results presented in [8]. Nevertheless, the running time of our **[GPU_N]** implementation is significantly smaller than the one provided in [8].

The last dataset is **[Full]**, and results are shown in table 4. As we expect, this case is very hard for sparse matrices based implementations: the running time grows too fast. This dataset also demonstrates the impact of the grammar size. Both queries specify the same constraints, but the grammar G_3 in CNF contains 2 times more rules then the grammar G_2, so, the running time for big graphs differs by more than twice.

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 DISCUSSION

First of all, note that dispite in [**?**] shows that CPU implementations (both dense and sparse matrix based) are relatively slow, our CPU implementation demonstrates good performance on real-world data.

Zeros in sparse matrices.

Overhead on matrices convertion and transferring when run on GPGPU.

No convertion when run on CPU. Beter then Neo4j and other BD with non-matrix representation of graphs.

7 CONCLUSION AND FUTURE WORK

We provide an CPU and GPGPU based context-free path querying implementations for RedisGraph and show that CFPQ can be fast enough to analize real-world data. But our implementations are on prototype stage and we should provide full integration of CFPQ to RedisGraph. First of all it is necessary to extend Cupher

Table 4: Full querying results

#V	Query G_2						Query G_3					
	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
100	0.007	0.002	0.002	< 1	0.003	0.278	0.023	0.076	0.005	0.001	0.007	0.290
200	0.040	0.003	0.002	0.001	0.004	0.279	0.105	0.098	0.004	0.001	0.007	0.296
500	0.480	0.003	0.003	0.001	0.004	0.329	1.636	0.094	0.007	0.001	0.010	0.382
1000	3.741	0.007	0.005	0.001	0.006	0.571	13.071	0.106	0.009	0.001	0.009	0.839
2000	40.309	0.063	0.019	0.003	0.017	1.949	93.676	0.108	0.030	0.005	0.026	3.740
5000	651.343	0.366	0.125	0.038	0.150	99.651	1205.421	0.851	0.195	0.075	0.239	201.151
10000	-	1.932	0.552	0.315	0.840	1029.042	-	4.690	1.055	0.648	1.838	-
25000	-	33.236	7.252	5.314	15.521	-	-	70.823	15.240	10.961	36.495	-
50000	-	360.035	58.751	44.611	129.641	-	-	775.765	130.203	91.579	226.834	-
80000	-	1292.817	256.579	190.343	641.260	-	-	-	531.694	376.691	-	-

graph query language, which uses in RedisGraph, to support respective syntax for context-free constraints specification. There is a proposal whish describes such syntax extension[7] and we plan to support proposed sintax in libcypher-parser[8] which uses in RedisGraph.

In current version we use CUSP matrix multiplication library for GPGPU utilization, but it may be better to use GraphBLAST[9] [20] — Gunrock[10] [18] based implementation of GraphBLAS API for GPGPU. Firs of all, we should evaluate GraphBlast based implementation of CFPQ. Also, we should investigate to implement multi-GPU support for GraphBlast, because it should improve performance of huge real-world data processing.

Our implementations calculate queryes in respect to relational semantics, but in sume cases it is necessary to provide a path which satisfied constraints. As we know, matrix based algorithm for single path or all paths semantix is not prvided yet, and it is a direction for future research.

Another important question for future research is how to update query result dynamically when data changes. Mechanism for result updating allows one to recalculate qury faster and use result as an index for other queryes.

Also, futer improvements of the dataset are requred. For example, it is necessary to include real-world cases from static code analysis [**?**].

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] 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

[13] 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

[14] 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

[15] 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

[16] 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.

[17] 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

[18] 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

[19] 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.

[7] 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 proposrd syntax. Access date: 12.11.2019

[8] Web page of libcypher-parser project: http://cleishm.github.io/libcypher-parser/. Access date: 12.11.2019

[9] GraphBLAST project: https://github.com/gunrock/graphblast. Access date: 12.11.2019.

[10] Gunrock project web page: https://gunrock.github.io/docs/. Access date: 12.11.2019.

[20] 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

[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*. ACM, 230–242.

[22] 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.