Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication

Nikita Mishin
Iaroslav Sokolov
Egor Spirin
mishinnikitam@gmail.com
sokolov.yas@gmail.com
egor@spirin.tech
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, Russia 199034

Vladimir Kutuev
Egor Nemchinov
Sergey Gorbatyuk
vladimir.kutuev@gmail.com
nemchegor@gmail.com
sergeygorbatyuk171@gmail.com
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, Russia 199034

Semyon Grigorev
s.v.grigoriev@spbu.ru
semen.grigorev@jetbrains.com
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, Russia 199034
JetBrains Research
Universitetskaya emb., 7-9-11/5A
St. Petersburg, Russia 199034

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 utilize modern parallel hardware and software to achieve high performance of CFPQ easily. In this work, we provide results of empirical performance comparison of different implementations of this algorithm on both real data and synthetic data for the worst cases.

CCS CONCEPTS

• **Information systems → Query languages for non-relational engines**; • **Theory of computation → Grammars and context-free languages**; *Parallel computing models*; • **Computing methodologies → Massively parallel algorithms**; • **Computer systems organization →** *Single instruction, multiple data*;

KEYWORDS

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

ACM Reference format:
Nikita Mishin, Iaroslav Sokolov, Egor Spirin, Vladimir Kutuev, Egor Nemchinov, Sergey Gorbatyuk, and Semyon Grigorev. 2018. Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication. In *Proceedings of GRADES-NDA 2019: the 2nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) 2019, Amsterdam, Netherlands, June 30, 2019 (GRADES-NDA 2019),* 5 pages.
https://doi.org/10.1145/1122445.1122456

GRADES-NDA 2019, June 30, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Language-constrained path querying [5], and particulary Context-Free Path Querying (CFPQ) [14] widely used for graph-structured data analysis in such areas as biological data analysis, RDF, network analysis. Huge amount of the real-world data makes performance of CFPQ evaluation critical for practical tasks, and number of algorithms for CFPQ evaluation proposed ricently [7, 9, 11–13, 15].

One of the most promising algorithm is a matrix-based algorithm, proposed by Rustam Azimov [4]. This algorith offloads the most critical computattions onto boolean matrices multiplication. As a result, it is pretty simple for implementattion and allows one to utilize modern massive-parallel hardware for CFPQs evaluation. Implenetation provided by authors utilizes GPGPU by using cuSPARSE[1] library which is floating point sparse matrices multiplication library. Even it does not use advanced algorithms for boolean matrices, it outperforms existing algorithms.

It is necessary to investigate an effect of specific algorithms and implementation techniques on performance of CFPQ. One of problems is that there is no publically available standartd dataset for CFPQ algorithms evaluation which includes both graph-structured data and queryes.

In this work, we do empirical performance comparison of different implementations of matrices multiplication based algorithm for CFPQ on both real data and synthetic data for the worst cases. We make the following contributions in this paper.

(1) We provide a number of implementations of the matrix multiplication based CFPQ algoithm, which utilizes different modern software and hardware. Source code is available on GitHub:!!!
(2) We collect and publish a dataset which contains both real data and syntatic data for wirst cases. This dataset contains data and queries in the simplee textual format, so it can be used for other algorithms evaluation easely. We hope that this dataset can be a base for unified benchmark for CFPQ algorithms.
(3) We provide evaluation results which shows that !!!

[1]cuSparse is a library for GPGPU utilization for sparse matrices multiplication. Offical documentation:https://docs.nvidia.com/cuda/cusparse/index.html. Access date: 12.03.2019

2 MATRIX-BASED ALGORITHM FOR CFPQ

Matrix-based algoritm for CFPQ was proposed by Rustam Azimov [4]. This algorithm can be expressed in few lines of code in terms of matrices operations, and it is a sufficient advantage for implementation. It was shown that GPGPU utilization for queryes evaluation can significantly improve performance in comparison with other implementations [4] even float matrices used instead of boolean matrices.

Pseudocode of the algorithm is presented in listing 1.

Algorithm 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
  $T \leftarrow$ the matrix $n \times n$ in which each element is $\varnothing$
  for all $(i, x, j) \in E$ **do** ▷ Matrix initialization
    $T_{i,j} \leftarrow T_{i,j} \cup \{A \mid (A \rightarrow x) \in P\}$
  while matrix T is changing **do**
    $T \leftarrow T \cup (T \times T)$ ▷ Transitive closure calculation
  return T

Here $D = (V, E)$ be the input graph and $G = (N, \Sigma, P)$ be the input grammar. Each cell of the matrix T contains the set of nonterminals such that $N_k \in T[i, j] \iff \exists p = v_i \dots v_j$—path in D, such that $N_k \overset{*}{\underset{G}{\Rightarrow}} \omega(p)$, where $\omega(p)$ is a word formed by labels along path p. Thus, this algorithm solves reachability problem, or, according Hellings [8], process CFPQs by using relational query semantics.

As you can see, performance-critical part of this algorithm is a matrix multiplication. Note, that the set of nonterminals is finite, we can represent the matrix T as e set of boolean matrices: one for each nonterminal. In this case the matrix updeta operation be $T_{N_i} \leftarrow T_{N_i} + (T_{N_j} \times T_{N_k})$ for each production $N_i \rightarrow N_j\ N_k$ in P. Thus we can reduse CFPQ to boolean matrices multiplication. After such transfromation we can apply the next optimization: we can skip update if there are no changes in the matrices T_{N_j} and T_{N_k} at the previous iteration.

Thus, the most important part is efficient implementation of operations over boolean matrices, and in this work we compare effects of utilization of different approaches to matrices multiplication. All our implemetations are based on the optimized version of the algorithm.

3 IMPLEMENTATION

We implement matrix-based algorithm for CFPQ by using a number of different programming languages and tools. Our goal is to investigate effects of the next features of implmentation.

- **GPGPU utilization.** It is mell-known that GPGPUs are sutable for matrices operations, but performance of whole solution depends on task details: overhead on data transferring may negate effect of parallel computations. Can GPGPUs utilization for CFPQ improve performance in comparison with CPU version?
- **Existing libraries utilization** is a good practice in software engeneering. Is it possible to achaive highe performance by using existing libraries for matrices operations or we need to create own solution to get more control?
- **Low-level programming**. GPGPU programming is traditionally low-level programming by using C-based languages (CUDA C, OpenCL C). On the other hand, there are number of approaches to create GPGPU-based solution by ysing such high-level languages as a Python. Can we get high-performance solution by using such approaches?
- **Sparce matrices.** Real graphs often are sparse, but not always. Is it sutable to use sparse matrix representation for CFPQ?

We provide next implementations for investigation.

- CPU-based solutions
  [Scipy] Saprse matrices multiplication by using Scipy [10] in Python programming language.
  [M4RI] Dense matrices multiplication by using m4ri[2] [1] library which implements 4 russian method [3] in C language. This library choosen because it is one of performnat implementation of 4 russian method [2].
- GPGPU-based solutions
  [GPU4R] Manual implemenattion of 4 russian metod in CUDA C.
  [GPU_N] Manual implementation of naïve boolean matrix multiplication in CUDA C.
  [GPU_Py] Manual implementation of naïve boolean matrix multiplication in Pyton by using numba compiler[3].

As far as number of matrices and its size can be statically defined at he start, all GPGPU based implementattions allocate all requred memory on the GPGPU only onece, at the start of computattions. By this way it is possible to significante reduce overhead on data transferring: all input data loads to GPGPU at the start, and result loads from GPGPU to the host at the finish. No active data transferring and memory allocating duting query computation.

4 DATASET DESCRIPTION

We create and publish a dataset for CFPQ algorithms evaluation. This dataset contains both the real data and syntetic data for different cpecific cases, such as theoretical worst case, or matrices representation specific worst cases.

Our goal is querying algorithms evaluation, not a graph storages or graph databases evaluation, so all data is presented in text-based format to simplify usage in different environments. Grammars are in Chomscy Normal Form and are stored in the files with `yrd` extension. Each line is a rule in form of triple or pair. The example of grammar representation is presented in figure 1

Graphs are represented as a set of triples (edges) and are stored in the files with `txt` extension. Example of graph is presented in figure 2.

[2] Actually we use pull request which is not merged yet: https://bitbucket.org/malb/m4ri/pull-requests/9/extended-m4ri-to-multiplication-over-the/diff. The original library implements operations over $GF(2)$, and this pull request conteains operations over boolean semiring

[3] Numba is a JIT compiler which supports GPGPU for subset of Python programming. Offical page: http://numba.pydata.org/. Access date: 03.05.2019

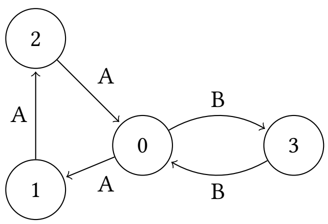
$$s \rightarrow A\ s\ B$$
$$s \rightarrow A\ B$$

(a) Grammar G_1

```
s a b
s a s1
s1 s b
a A
b B
```

(b) Representation of grammar G_1 in yrd file

Figure 1: Example of grammar representation in the yrd file



(a) Example of input graph D_1

```
0 A 1
1 A 2
2 A 0
0 B 3
3 B 0
```

(b) Representation of the input graph D_1 in txt file

Figure 2: Example of graph representation in txt file

Each case is a pair of set of graphs and set of grammars: each query (grammar) should be applied to each graph. Cases are placed in folders with case-specific name. Grammars and graph are placed in subfolders with names `Grammars` and `Matrices` respectively.

It is known that varians of the *same generation query* **??** are classical example of queryes that are context-free but not regular, so we use this type of queryes in our evaluation. The datatset includes data for next cases.

[RDF] The set of real RDF files (ontologies) from [15] and two variants of the same generation query (figures **??**) which describes hyrachy analysis.

[Worst] Theoretical worst case for CFPQ time complexity which is proposed by Hellings [9]: graph is a two cycles of coprime lengths with single common vertex. First cycle labelled by open bracket and the second cysle is labelled by close bracked. Query is a grammar for A^nB^n languge (grammar G_1, figure 1).

[Full] The case when input graph is sparce, but result is a full graph. Such case may be a hard for sparse matrices representation. As an input graph we use a cycle all edges of which is labelled by the same token. As a queryes we use two grammars which describe arbitrary repetition of a token: unambiguous and highly ambiguous grammar (figure **??**).

[Sparse] Sparse graphs from [6] which generated by the GT-graph graph generator, and emulates realistic sparse data. Names of these graphs have a form `Gn-p`, where `n` represents the total number of vertices, each pair of vertices is connected by probability `p`. Query is a same generation query.

5 EVALUATION

We evaluate all described implementattions on all data and queries from presented dataset. Also we provide results for implementation provided in [4] for comparison. Our goal is to comare CFPQ evaluation algorithms, so we exclude time requred for load data from files. Time requred for data transferring is included.

For evaluation we use PC with Ubuntu 18.04 installed. It has Intel core i7 8700k 3,7HGz CPU, Ddr4 32Gb RAM, and Geforce 1080Ti GPGPU with 11Gb RAM.

Results of evaluation are presented in tables below. Time is mesured in seconds. Result for each algorithm is an averege time of 10 runs.

First is a **[RDF]** dataset. Results are presented in a table 1.

We can see, that in this case running time for all our implementations smaller than time for reference implementattion, and that **[GPU_N]** is faster than other implementations while other implementations demonsrtate similar performance. Also it is obvious that performance improvement in comarison with first implementations is huge and it is necessary to select new significantly biggest RDF files.

Results of theoretical worst case (**[Worst]** datatset) is presented in table 2.

Table 2: Worst case evaluation results

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

This case is really hard to process: even for graph with 1024 vertices query evaluation time grather than 10 seconds even for most performant implementation. Also we can see, that time grows fast with grows of vertices number.

Next is a **[Sparse]** datatset. Results are presented in table 3.

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	!!!
G10k-0.001	37.286	2.395	0.435	0.215	!!!	!!!
G10k-0.01	97.607	1.455	0.273	0.138	0.763	!!!
G10k-0.1	!!!	1.050	0.223	0.114	0.859	!!!
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	-	!!!

For such type of graphs !!!! Note that we estimate only query execution time, so it is hard to compare our results with results presented in [6]. But it would be interesting to do such comaprison in future because running time of our **[GPU_N]** implementation is significantly smaller than provided in [6].

The last dataset is a **[Full]**, and results a shown in table 4

Finally, we can cocnlude that

Table 1: RDFs querying results

RDF			Query 1						Query 2					
Name	#V	#E	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
atom-primitive	291	685	0.003	0.002	0.002	0.001	0.005	!!!	0.001	< 0.001	0.001	< 0.001	0.002	!!!
biomed.-mesure-primitive	341	711	0.003	0.005	0.002	0.001	0.005	!!!	0.004	< 0.001	0.001	< 0.001	0.005	!!!
foaf	256	815	0.002	0.009	0.002	< 0.001	0.005	!!!	0.001	< 0.001	0.001	< 0.001	0.002	!!!
funding	778	1480	0.004	0.007	0.004	0.001	0.005	!!!	0.002	< 0.001	0.003	< 0.001	0.004	!!!
generations	129	351	0.003	0.003	0.002	< 0.001	0.005	!!!	0.001	< 0.001	0.001	< 0.001	0.002	!!!
people_pets	337	834	0.003	0.003	0.003	0.001	0.007	!!!	0.001	< 0.001	0.001	< 0.001	0.003	!!!
pizza	671	2604	0.006	0.008	0.003	0.001	0.006	!!!	0.002	< 0.001	0.002	< 0.001	0.005	!!!
skos	144	323	0.002	0.004	0.002	< 0.001	0.005	!!!	< 0.001	< 0.001	0.001	< 0.001	0.002	!!!
travel	131	397	0.003	0.005	0.002	< 0.001	0.006	!!!	0.001	< 0.001	0.001	< 0.001	0.003	!!!
univ-bench	179	413	0.002	0.004	0.002	< 0.001	0.005	!!!	0.001	< 0.001	0.001	< 0.001	0.003	!!!
wine	733	2450	0.007	0.006	0.004	0.001	0.007	!!!	0.001	< 0.001	0.003	< 0.001	0.003	!!!

Table 4: Full querying results

#V	Query 1						Query 2					
	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
100	0.007	0.002	0.002	< 0.001	0.003	!!!	0.023	!!!	0.005	0.001	0.007	!!!
200	0.040	0.003	0.002	0.001	0.004	!!!	0.105	!!!	0.004	0.001	0.007	!!!
500	0.480	0.003	0.003	0.001	0.004	!!!	1.636	!!!	0.007	0.001	0.010	!!!
1000	3.741	0.007	0.005	0.001	0.006	!!!	13.071	!!!	0.009	0.001	0.009	!!!
2000	40.309	!!!	0.019	0.003	0.017	!!!	93.676	!!!	0.030	0.005	0.026	!!!
5000	651.343	0.366	0.125	0.038	0.150	!!!	!!!	0.851	0.195	0.075	0.239	!!!
10000	-	1.932	0.552	0.315	0.840	!!!	!!!	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	!!!	!!!
80000	-	1292.817	256.579	190.343	641.260	!!!	-	-	531.694	376.691	!!!	!!!

- On GPU utilization
- On Existinng libraries
- On Low-level programming
- On sparse matrices

6 CONCLUSION AND FUTURE WORK

We provide a number of implementations of matrix-based algorithm for context-free path querying, collect a dataset for evaluation and provide results of evaluation of our implementation on collected dataset. Our evaluation shows that GPGPU utilization for boolean matrices multiplication can significantely increase perfprmance of CFPQs evaluation, but requires more research on implementattion details.

First direction for future research is a more detailed CFPQ algorithms investigation. We should do more evaluation on sparse matrices on GPGPUs and investigate technics for high-performance GPGPU code creation. Also it is nesessary to implement and evaluate solutions for graphs which is not fit in RAM, and for big queryes which disallow to allocate all requred matrices on single GPGPU. We hope that it is possible to utilize existing technics for huge matrices multiplication for this problem.

Another direcion is a dataset improvement. First of all, it is necessary to collect more data, and more grammars/queries. Especially it would be interesting to add to dataset more real graphs and more real queryes. Secondly, it is nesessary to discuss and fix data format to be able to evaluate different algorithms. We think that it is necessary to create public dataset for CFPQ algorithms evaluation, and collaboration with community is requred.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Martin Albrecht and Gregory Bard. 2019. *The M4RI Library*. The M4RI Team. https://bitbucket.org/malb/m4ri
[2] MR Albrecht, GV Bard, and W Hart. 2008. Efficient multiplication of dense matrices over GF (2). *arXiv preprint arXiv:0811.1714* (2008).
[3] Vladimir L'vovich Arlazarov, Yefim A Dinitz, MA Kronrod, and IgorAleksandrovich Faradzhev. 1970. On economical construction of the transitive closure of an oriented graph. In *Doklady Akademii Nauk*, Vol. 194. Russian Academy of

Sciences, 487–488.

[4] 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

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

[6] 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).

[7] 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

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

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

[10] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–2019. SciPy: Open source scientific tools for Python. http://www.scipy.org/ [Online; accessed 5.3.2019].

[11] 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

[12] 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.

[13] 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

[14] 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.

[15] 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.