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),* ?? 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 [**?**], and particulary Context-Free Path Querying (CFPQ) [**?**] widely used for graph-structured data analysis in such areas as biological data analysis, RDF, network analysis. Real-world data is huge, so performance of CFPQ evaluation is important for practical tasks. Resently active research !!! [**?**]

One of the most promising algorithm is a matrix-based algorithm, proposed by Rustam Azimov. This algorith is pretty simple for implementattion and allow to utilize massive-parallel hardware for CFPQs evaluation by offloading the most critical computattions to matrices multiplication. CPU/GPGPU based implementation. Investigate and compare.

There is no publically available standartized dataset for algorithms evaluation. We collect some data and propose possible candidate for it.

Research question: comparison of differend implementations of matrix-based CFPQ. We implement and compare peformance.

We make the following contributions in this paper.

(1) Implementation. Source code is available on GitHub:!!!
(2) Dataset for evaluation. Real and syntethic data. Available. Data format. Reference values.
(3) Evaluation. We show that !!!

2 MATRIX-BASED ALGORITHM FOR CFPQ

Matrix-based algoritm for CFPQ was proposed by Rustam Azimov [**?**]. 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 [**?**] even float matrices used instead of boolean matrices.

Pseudocode of the algorithm is presented in listing 1.

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 [**?**], process CFPQs by using relational query semantics.

Algorithm 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:     T ← the matrix n × n in which each element is ∅
6:     for all (i, x, j) ∈ E do                ▷ Matrix initialization
7:         T_{i,j} ← T_{i,j} ∪ {A | (A → x) ∈ P}
8:     end for
9:     while matrix T is changing do
10:        T ← T ∪ (T × T)                     ▷ Transitive closure calculation
11:    end while
12:    return T
13: end function
```

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 fo 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 thet 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. Moreover, it is believed that GPGPUs is not sutable boolean calculations [**?**]. 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 [**?**] in Python programming language.

[M4RI] Dense matrices multiplication by using m4ri[1] [**?**] library which implements 4 russian method [**?**] in C language. This library choosen because it is one of performnat implementation of 4 russian method [**?**].

- 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[2].

Generic notes on optimizations. Notes on data transferring. On matrix changes tracking (we should multiply pair of matrices only if one of them changed in last iteration)

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

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

(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

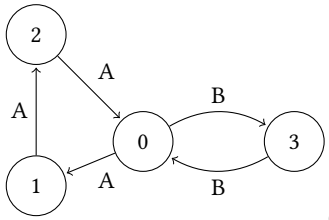
Graphs are represented as a set of triples (edges) and are stored in the files with `txt` extension. Example of graph is presented in figure **??**.

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.

[1] 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

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



(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

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

[Worst] Theoretical worst case for CFPQ time complexity which is proposed by Hellings [**?**]: 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 [**?**] which emulates realistic sparse data. Query is a same generation query.

5 EVALUATION

We evaluate all described implementattions on all data and queries from presented dataset.

For evaluation we use PC with the next characteristics.

- OS
- CPU
- RAM
- GPU
  - Name
  - GHz
  - RAM
  - Threads
  - ...
- Libs versions
- Pyton runtime

Compiler options, Python runtime, etc.

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

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

We can see, that in this case !!!!

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

Table 2: Worst case evaluation results

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

In this case !!!!! In this case !!!!!

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

Table 3: Sparse graphs querying results

Graph	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
G5k	10.352	!!!	0.113	0.041	0.216	!!!
G10k	37.286	!!!	0.435	0.215	1.331	!!!
G10k-0.01	97.607	!!!	0.273	0.138	0.763	!!!
G10k-0.1	!!!	!!!	0.223	0.114	0.859	!!!
G20k	150.774	!!!	1.842	1.274	6.180	!!!
G40k	-	!!!	11.663	8.393	37.821	!!!
G80k	-	!!!	88.366	65.886	-	!!!

For such type of graphs !!!!

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

Finally, we can cocnlude that

- 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 shouws that!!!

log cfg. datalog -> cfg or to boolean/conjunctive

First direction for future research is a more detailed CFPQ algorithms investigation. We should do more evaluation on sparse matrices on GPGPUs.

Also it is nesessary to implement and evaluate solutions for graphs which is not fit in RAM. There is a set of technics for huge matrices multiplication. Is it possible to dopt it for CFPQ

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 !!! data format to be able to evaluate different algorithms. 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.

Table 1: RDFs querying results

RDF	Query 1						Query 2					
	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
atom-primitive	0.003	!!!	0.002	0.001	0.005	!!!	0.001	!!!	0.001	0	0.002	!!!
biomedical-mesure-primitive	0.003	!!!	0.002	0.001	0.005	!!!	0.004	!!!	0.001	0	0.005	!!!
foaf	0.002	!!!	0.002	0	0.005	!!!	0.001	!!!	0.001	0	0.002	!!!
funding	0.004	!!!	0.004	0.001	0.005	!!!	0.002	!!!	0.003	0	0.004	!!!
generations	0.003	!!!	0.002	0	0.005	!!!	0.001	!!!	0.001	0	0.002	!!!
people_pets	0.003	!!!	0.003	0.001	0.007	!!!	0.001	!!!	0.001	0	0.003	!!!
pizza	0.006	!!!	0.003	0.001	0.006	!!!	0.002	!!!	0.002	0	0.005	!!!
skos	0.002	!!!	0.002	0	0.005	!!!	0	!!!	0.001	0	0.002	!!!
travel	0.003	!!!	0.002	0	0.006	!!!	0.001	!!!	0.001	0	0.003	!!!
univ-bench	0.002	!!!	0.002	0	0.005	!!!	0.001	!!!	0.001	0	0.003	!!!
wine	0.007	!!!	0.004	0.001	0.007	!!!	0.001	!!!	0.003	0	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
10	0.001	!!!	0	0	0.002	!!!	0.002	!!!	0.001	0.001	0.004	!!!
50	0.002	!!!	0.001	0	0.003	!!!	0.005	!!!	0.002	0.001	!!!	!!!
100	0.007	!!!	0.002	0	0.003	!!!	0.023	!!!	0.005	0.001	0.007	!!!
200	0.040	!!!	0.002	0.001	0.004	!!!	0.105	!!!	0.004	0.001	0.007	!!!
500	0.480	!!!	0.003	0.001	0.004	!!!	1.636	!!!	0.007	0.001	0.010	!!!
1000	3.741	!!!	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	!!!	0.125	0.038	0.150	!!!	!!!	!!!	0.195	0.075	0.239	!!!
10000	-	!!!	0.552	0.315	0.840	!!!	!!!	!!!	1.055	0.648	1.838	!!!
25000	-	!!!	7.252	5.314	15.521	!!!	-	!!!	15.240	10.961	36.495	!!!
50000	-	!!!	58.751	44.611	129.641	!!!	-	!!!	130.203	91.579	!!!	!!!
80000	-	!!!	256.579	190.343	641.260	!!!	-	!!!	531.694	376.691	!!!	!!!

REFERENCES

[] Martin Albrecht and Gregory Bard. 2019. *The M4RI Library.* The M4RI Team. https://bitbucket.org/malb/m4ri

[] MR Albrecht, GV Bard, and W Hart. 2008. Efficient multiplication of dense matrices over GF (2). *arXiv preprint arXiv:0811.1714* (2008).

[] 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.

[] 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

[] 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).

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

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

[] 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].

[] 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.