Evaluation of Matrix-Based Context-Free Path Quering Algorithm

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

Egor Spirin
Vladimir Kutuev
egor@spirin.tech
vladimir.kutuev@gmail.com
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, Russia 199034

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

Semyon Grigorev
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

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference format:
Nikita Mishin, Iaroslav Sokolov, Egor Spirin, Vladimir Kutuev, Egor Nemchinov, Sergey Gorbatyuk, and Semyon Grigorev. 2018. Evaluation of Matrix-Based Context-Free Path Quering Algorithm. In *Proceedings of Woodstock '18: ACM Symposium on Neural Gaze Detection, Woodstock, NY, June 03–05, 2018 (Woodstock '18),* 5 pages.
https://doi.org/10.1145/1122445.1122456

Woodstock '18, June 03–05, 2018, Woodstock, NY

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

1 INTRODUCTION

Graph querying, Context-Free Path Querying (CFPQ), applications in different areas. Performance is important for practical tasks.

Matrix-based algorithm. Pretty simple. Performance problems. 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.

Contribution

- Implementation. Source code is available on GitHub:!!!!!!
- Evaluation
- Dataset for evaluation. Available. Data format. Reference values.

This paper is organized as follows. !!!!

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 ← the number of nodes in D
    E ← the directed edge-relation from D
    P ← the set of production rules in G
    T ← the matrix n × n in which each element is ∅
    for all (i, x, j) ∈ E do                    ▷ Matrix initialization
        T_{i,j} ← T_{i,j} ∪ {A | (A → x) ∈ P}
    end for
    while matrix T is changing do
        T ← T ∪ (T × T)                         ▷ Transitive closure calculation
    end while
    return T
end function
```

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 $N_k \in T[i, j] \iff \exists p = v_i \ldots v_j$—path in D, such that $N_k \xRightarrow[G]{*} \omega(p)$, where $\omega(p)$ is a word formed by labels along path p. Thus, this algorithm solves reachability problem, or, according Hellings [6], 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 $N_i \to N_j\ N_k$ 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?
- **Sparse 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 [7] in Python programming language.

[M4RI] Dense matrices multiplication by using m4ri[1] [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[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 wirst cases.

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

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

(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 2.

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

[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

```
0 A 1
1 A 2
2 A 0
0 B 4
4 B 0
```

Figure 2: Example of graph representation in `txt` file

in folders with case-specific name. Grammars and graph are placed in subfolders with names `Grammars` and `Matrices` respectively.

The datatset includes data for next cases.

[RDF] The set of RDF files from [8] and two variants of the same generation query (figures **??**) which is classical queries for CFPQ [**?**].

[Worst] Theoretical worst case for CFPQ which is proposed by Hellings [**?**]. Grammar is G_1.

[Full] Cycle to full. Two grammars: unambiguous and highly ambiguous. Grammars are presented in figure **??**

[Sparse] Sparse graphs from [5]. 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
- Libs versions
- Pyton runtime

Compiler options, Python runtime, etc.

Results of evaluation are presented in tables below.

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

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

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

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

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

For such type of graphs !!!!

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

Finally, we can cocnlude that

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

6 CONCLUSION AND FUTURE WORK

We present !!!

Our evaluation shows that !!!

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. More data. More grammars/queries.

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] 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).

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

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

[8] 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.

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
rdf1	1	2	2	2	2	2	2	2	2	2	2	2
rdf1	1	2	2	2	2	2	2	2	2	2	2	2
rdf1	1	2	2	2	2	2	2	2	2	2	2	2

Table 2: Worst case evaluation results

Graph	Query 1					
	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
rdf1	1	2	2	2	2	2
rdf1	1	2	2	2	2	2
rdf1	1	2	2	2	2	2

Table 3: Sparse graphs querying results

Graph	Query 1					
	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
rdf1	1	2	2	2	2	2
rdf1	1	2	2	2	2	2
rdf1	1	2	2	2	2	2

Table 4: Full querying results

Graph size	Query 1						Query 2					
	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs	Scipy	M4RI	GPU4R	GPU_N	GPU_Py	CuSprs
rdf1	1	2	2	2	2	2	2	2	2	2	2	2
rdf1	1	2	2	2	2	2	2	2	2	2	2	2
rdf1	1	2	2	2	2	2	2	2	2	2	2	2