

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

Nikita Mishin
Iaroslav Sokolov
Egor Spirin
mishinnikitam@gmail.com
sokolov.yas@gmail.com
egor@spirin.tech
Saint Petersburg State University
St. Petersburg, Russia

Vladimir Kutuev
Egor Nemchinov
Sergey Gorbatyuk
vladimir.kutuev@gmail.com
nemchegor@gmail.com
sergeygorbatyuk171@gmail.com
Saint Petersburg State University
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) evaluation offloads the most performance-critical parts onto boolean matrices multiplication (BMM). 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

Transitive closure, context-free path querying, graph databases, context-free grammar, GPGPU, 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 Matrix Multiplication Based Context-Free Path Querying Algorithm. In *Proceedings of Woodstock '18: ACM Symposium on Neural Gaze Detection (Woodstock '18)*. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/1122445.1122456>

1 INTRODUCTION

Language-constrained path querying [?], and particularly 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

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

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9999-9/18/06...\$15.00
<https://doi.org/10.1145/1122445.1122456>

evaluation is important for practical tasks. Recently active research !!! [?]

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

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

Research question: comparison of different implementations of matrix-based CFPQ. We implement and compare performance.

We make the following contributions in this paper.

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

2 MATRIX-BASED ALGORITHM FOR CFPQ

Matrix-based algorithm 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 queries 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.

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 $N_k \xrightarrow[G]{*} \omega(p)$, 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 a set of boolean matrices: one for each nonterminal. In this case the matrix update 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 reduce CFPQ to boolean matrices multiplication. After such transformation we can apply the next optimization: we can

Algorithm 1 Context-free path querying algorithm

```

1: function CONTEXTFREEPATHQUERYING( $D, G$ )
2:    $n \leftarrow$  the number of nodes in  $D$ 
3:    $E \leftarrow$  the directed edge-relation from  $D$ 
4:    $P \leftarrow$  the set of production rules in  $G$ 
5:    $T \leftarrow$  the matrix  $n \times n$  in which each element is  $\emptyset$ 
6:   for all  $(i, x, j) \in E$  do ▷ Matrix initialization
7:      $T_{i,j} \leftarrow T_{i,j} \cup \{A \mid (A \rightarrow x) \in P\}$ 
8:   end for
9:   while matrix  $T$  is changing do
10:     $T \leftarrow T \cup (T \times T)$  ▷ Transitive closure calculation
11:  end while
12:  return  $T$ 
13: end function

```

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 implementation.

- **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. 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 achaiive 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] Saprsre matrices multiplication by using Scipy [8] in Python programming language.

[M4RI] Dense matrices multiplication by using m4ri¹ [1] library which implements 4 russian method [3] in C language.

¹Actually 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

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 Python by using numba compiler².

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 Chomsky 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

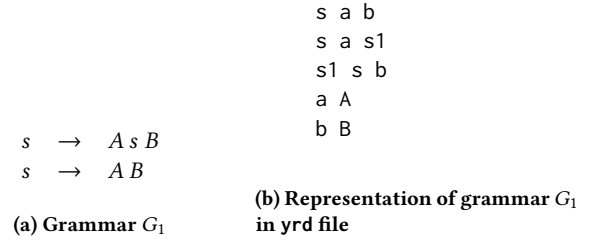


Figure 1: Example of grammar representation in the yrd file

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.

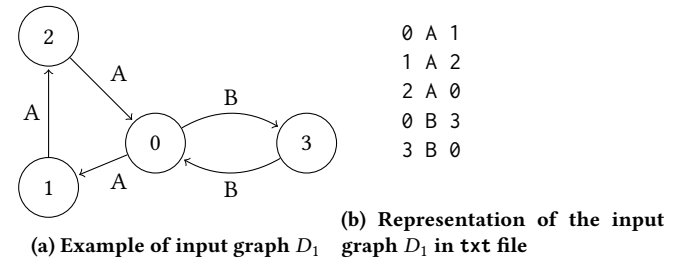


Figure 2: Example of graph representation in txt file

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

implements operations over $GF(2)$, and this pull request conteains operations over boolean semiring

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

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

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

[RDF] The set of real RDF files (ontologies) from [9] and two variants of the same generation query (figures ??).

[Worst] Theoretical worst case for CFPQ time complexity which is proposed by Hellings [7]: graph is a two cycles of co-prime lengths with single common vertex. First cycle labelled by open bracket and the second cycle is labelled by close bracketed. Query is a grammar for $A^n B^n$ language (grammar G_1 , figure 1).

[Full] The case when input graph is sparse, 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 queries we use two grammars which describe arbitrary repetition of a token: unambiguous and highly ambiguous grammar (figure ??).

[Sparse] Sparse graphs from [5] which emulates realistic sparse data. Query is a same generation query.

5 EVALUATION

We evaluate all described implementations 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
- Python runtime

Compiler options, Python runtime, etc.

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

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]** dataset) is presented in table 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

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

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

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

For such type of graphs !!!!

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

Finally, we can conclude that

- On GPU utilization
- On Existing 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!!!

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 necessary to implement and evaluate solutions for graphs which is not fit in RAM. There is a set of techniques for huge matrices multiplication. Is it possible to adopt it for CFPQ

Another direction 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 queries. Secondly, it is necessary to !!! data format to be able to evaluate different algorithms. Collaboration with community is required.

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 Igor Aleksandrovich 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.

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 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

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

[8] 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].

[9] 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.