GLL-based Context-Free Path Querying for Neo4j

Vadim Abzalov,[1,*] Vlada Pogozhelskaya,[2,] Vladimir Kutuev,[1,***] Olga Bachishche,[3,****] and Semyon Grigorev[1,*****]**

(Submitted by TODO: ADD SUBMITTER NAME)

[1] *Saint Petersburg State University, St. Petersburg, Russia*
[2] *National Research University Higher School of Economics, St. Petersburg, Russia*
[3] *ITMO University, St. Petersburg, Russia*

Received TODO: ADD DATES

Abstract—We propose a GLL-based *context-free path querying* algorithm that handles queries in Extended Backus-Naur Form (EBNF) using Recursive State Machines (RSM). Utilization of EBNF allows one to combine traditional regular expressions and mutually recursive patterns in constraints natively. The proposed algorithm solves both the *reachability-only* and the *all-paths* problems for the *all-pairs* and the *multiple sources* cases. The evaluation on real-world graphs demonstrates that the utilization of RSMs increases the performance of the query evaluation. Being implemented as a stored procedure for Neo4j, our solution demonstrates better performance than a similar solution for RedisGraph. The performance of our solution on the regular path queries is comparable to the performance of the native Neo4j solution, and in some cases, our solution requires significantly less memory.

2010 Mathematical Subject Classification: `68Q45, 68Q42, 68P15`

Keywords and phrases: *Graph database, graph querying, context-free path querying, CFPQ, reachability problem, all-paths problem, generalized LL, GLL*

1. INTRODUCTION

Context-free path querying (CFPQ) allows one to use context-free grammars to specify constraints on paths in edge-labeled graphs. Compared to regular path queries (RPQ), CFPQ is strictly more expressive: for instance, a context-free grammar can be built to find siblings or same-generation categories in a taxonomy [1, 2]. This expressiveness allows CFPQ to be used for graph analysis in such areas as bioinformatics (hierarchy analysis [3], similarity queries [4]), data provenance analysis [5], static code analysis [6, 7]. Although a lot of algorithms for CFPQ have been proposed, poor performance on real-world graphs and bad integration with real-world graph databases and graph analysis systems are still problems that hinder the adoption of CFPQ.

The problem with the performance of the CFPQ algorithms in real-world scenarios was pointed out by Jochem Kuijpers [2] as a result of an attempt to extend the Neo4j graph database with CFPQ. Several algorithms, based on such techniques as LR parsing algorithm [8], dynamic programming [9], LL parsing algorithm [10], linear algebra [1], were implemented, using Neo4j as a graph storage, and evaluated. None of them were performant enough to be used in real-world applications.

Since Jochem Kuijpers pointed out the performance problem, it was shown that linear-algebra-based CFPQ algorithms, which operate over the adjacency matrix of the input graph and utilize parallel algorithms for linear algebraic operations, demonstrate good performance [11]. Moreover, the matrix-based CFPQ algorithm is a base for the first full-stack support of CFPQ by extending the RedisGraph graph database [11].

* E-mail: vadim.i.abzalov@gmail.com
** E-mail: vvpogozhelskaia@edu.hse.ru
*** E-mail: v.kutuev@spbu.ru
**** E-mail: bachisheo@gmail.com
***** E-mail: s.v.grigoriev@spbu.ru

However, adjacency matrix is not the only possible format for graph representation, and data format conversion may take a lot of time; thus, it is not applicable in some cases. As a result, the development of the performant CFPQ algorithm for graph representations not based on matrices and its integration with real-world graph databases is still an open problem. Moreover, while the *all pairs context-free constrained reachability* is widely discussed in the community, such practical cases as the *all-paths* queries and the *multiple sources* queries are not studied enough.

Additionally, almost all existing solutions are for *reachability-only* problem. Recently, Jelle Hellings in [12] and Rustam Azimov in [13] have proposed algorithms that allow one to extract paths that satisfy the specified context-free constraints. The ability to extract paths of interest is important for some applications where the user wants to know not only the fact that one vertex is reachable from another one, but also to get a detailed explanation of why this vertex is reachable. One of such applications is a program code analysis where the fact is a potential problem in code, and paths can help to analyze and fix this problem. While the utilization of the general-purpose graph databases for code analysis is gaining popularity [14], CFPQ algorithms that provide a structural representation of paths are not studied enough.

Generalized LL (GLL) is a parsing algorithm that can handle any context-free grammar, including left-recursive and ambiguous ones. Similar to GLR (e.g. Tomita algorithm), it achieves this by using a graph-structured stack (GSS) to efficiently share and manage the parsing state, allowing it to explore all possible derivations in parallel without an exponential time cost. The output of the parsing is a compact representation of all possible parse trees for the input, known as a *parse forest*. It was shown that the GLL algorithm can be naturally generalized to the CFPQ algorithm [15]. Moreover, this provides a natural solution not only for the *reachability-only* problem but also for the *all-paths* problem. At the same time, there exists a high-performance GLL parsing algorithm [16] and its implementation in the Iguana project [17].

Additionally, pure context-free grammars in Backus-Naur Form (BNF) are too verbose to express complex constraints. However, almost all algorithms require a query to be in such form. At the same time, Extended Backus-Naur Form (EBNF) can be used to specify context-free languages. EBNF allows combining typical regular expressions with mutually recursive rules which are required to specify context-free languages. That makes EBNF more user-friendly. But there are no CFPQ algorithms that utilize EBNF for query specification.

In this paper, we generalize the GLL parsing algorithm to handle queries in EBNF without its transformation. We show that it allows us to increase the performance of the query evaluation. We also integrate our solution with the Neo4j graph database and evaluate it. Thus, we make the following contributions in this paper.

- We propose the GLL-based CFPQ algorithm that can handle queries in Extended Backus-Naur Form without transformation. Our solution utilizes Recursive State Machines (RSM) [18] for it. The proposed algorithm can be used to solve both the *reachability-only* and the *all-paths* problems.

- We provide an implementation of the proposed CFPQ algorithm. By experimental study on real-world graphs we show that utilization of RSMs allows one to increase performance of the query evaluation.

- We integrate the implemented algorithm with Neo4j by providing a respective stored procedure that can be called from Cypher. Currently, we use Neo4j as a graph storage and do not extend Cypher to express context-free path patterns. Thus, expressive power of our solution is limited: we cannot use the full power of Cypher within our constraints. Implementing a query language extension amounts to a lot of additional effort and is a part of future work.

- We evaluate the proposed solution on several real-world graphs. Our evaluation shows that the proposed solution in order of magnitude is faster than a similar linear algebra-based solution for RedisGraph. Moreover, we show that our solution is compatible with native Neo4j solution for RPQs, and in some cases requires significantly less memory. Note that while Cypher's expressively is limited, our solution can handle arbitrary RPQs.

The paper has the following structure. In section **??** we introduce basic notions and definitions from graph theory and formal language theory. Then, in section 2 we introduce *recursive state machines* and provide an example of the CFPQ evaluation using naïve strategy which may lead to infinite computations. Section **??** contains a description of the GLL-based CFPQ evaluation algorithm which uses RSM and solves problems of the naïve strategy from the previous section: the algorithm always terminates and can build a finite representation of all paths of interest. Section 3 introduces a data set (both graphs and queries) which will be used further for experiments. Further, in section 4 we use this data set to compare different versions of the GLL-based CFPQ algorithm. After that, in section 5 we provide details on the integration of the best version into the Neo4j graph database, and evaluate our solution on the data set introduced before. Related work is discussed in section 6. Final remarks and conclusion are provided in section 7.

2. RSM-BASED CFPQ ALGORITHM

In this section, we provide background on Recursive State Machines (RSMs) and describe informally how they can specify context-free constraints for path querying in graphs. Also we describe main idea behind adoption GLL for RSM-based CFPQ algorithm.

Recursive state machine or RSM [18] is a way to represent context-free languages in a way resembling finite automata. It allows us to use a graph-based representation of the context-free languages specification and unify the processing of regular and context-free languages. Moreover, RSMs provide a natural representation for grammars in Extended BackusвЂ“Naur Form (EBNF), which are often more compact and user-friendly than BNF.

In our work, we modify the GLL-based CFPQ algorithm to handle RSMs instead of grammars in BNF as a query specification. It enables performance improvement and native handling of both context-free and regular languages. A detailed description of the algorithm, including formal definitions of descriptors, GSS, and correctness proofs, is beyond the scope of this paper and can be found in extended version [19].

Our solution is based on the generalized LL parsing algorithm [20] which was shown to generalize well to graph processing [15]. To generalize GLL to handle RSMs as query specification we introduce *matched ranges*: each range is defined by two graph vertices (e.g., u and v) and two RSM states (e.g., p and q). A matched range of the form $(p, u), (q, v)$ indicates that there exist at least one path from u to v such that it forms a word is accepted by RSM when starting in state p and ending in state q.

As a result of parsing, GLL can construct a *Shared Packed Parse Forest* (SPPF) [21] — a special data structure which represents all possible derivations of the input in the compressed form. A Shared Packed Parse Forest (SPPF) was proposed by Jan Rekers in [22] to represent parse forest without duplication of subtrees. Later, other versions of SPPF were introduced in different generalized parsing algorithms. For example, GLL can provide the result in the form of the *binary subtree sets* [23]. As shown in [15], the SPPF can finitely represent the result of an all-paths query, enabling the recovery of any concrete path. In our work, we employ SPPF with the following types of nodes.

- A *Terminal node* to represent a matched edge label.
- A *Nonterminal node* to represent the root of the subtree which corresponds to paths that can be derived from the respective nonterminal.
- An *Intermediate node* which corresponds to the intermediate point used to connect two matched ranges. It has exactly two children (both range nodes) and is denoted $I_{q,v}$, indicating that ranges are joined at point (q, v).
- A *Range node* corresponds to a matched range $(p, u), (q, v)$ and is denoted $R^{p,u}_{q,v}$. It encodes all possible derivations of that range and can have arbitrarily many children of any type except range. Range nodes are reusable within the SPPF.
- An *Epsilon node* to represent the empty subtree in the case when nonterminal produces the empty string.

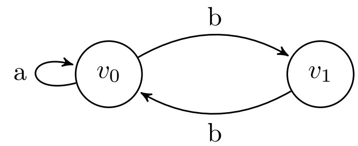


Fig. 1. Input graph.

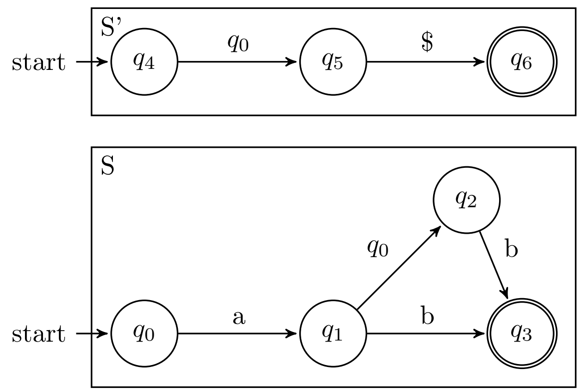


Fig. 2. Extended RSM for grammar $S \to a\ b \mid a\ S\ b$.

2.1. Example

Let us introduce a brief example. Suppose we have a graph as shown in Fig. 1 and we wish to find all paths from v_0 to v_0 that satisfy the constraint specified by the RSM in Fig. 2. The RSM describes paths of the form $a^n b^n$, where $n > 0$. It is easy to see that there are infinitely many such paths. Therefore, the resulting SPPF will contain cycles.

The resulting SPPF is shown in Fig. 3. Its root is the node $R^{q_4,v_0}_{q_5,v_0}$, which indicates that it represents paths from v_0 to v_0 that can be recognized by the RSM transitioning from state q_4 to q_5. This transition corresponds to recognizing a word derived from the non-terminal S. The node $R^{q_0,v_0}_{q_3,v_0}$ is the entry point for a cycle. The node $R^{q_0,v_0}_{q_3,v_1}$ has two siblings, indicating two ways to construct the corresponding subpath. The first corresponds to the shortest possible path:

$$v_0 \xrightarrow{a} v_0 \xrightarrow{a} v_0 \xrightarrow{b} v_1 \xrightarrow{b} v_0.$$

The second contains the cycle mentioned above and therefore represents an infinite family of paths formed by repeatedly wrapping the shortest path with additional pairs of as and bs. For example, traversing the cycle once yields the path:

$$v_0 \xrightarrow{a} v_0 \xrightarrow{a} v_0 \xrightarrow{a} v_0 \xrightarrow{a} v_0 \xrightarrow{b} v_1 \xrightarrow{b} v_0 \xrightarrow{b} v_1 \xrightarrow{b} v_0.$$

3. EXPERIMENT DESIGN

In this section, we provide a description of the graphs and queries used for the evaluation of the implemented algorithms. Also, we describe common evaluation scenarios and evaluation environment.

We evaluated our solution on both classical regular and context-free path queries to estimate the ability to use the proposed algorithm as a universal algorithm for the wide range of queries.

3.1. Graphs

For the evaluation, we use a number of graphs from the CFPQ_Data [24] data set. We selected a number of graphs related to RDF analysis. A detailed description of the graphs, namely the number of the vertices and edges and the number of the edges labeled by tags used in the queries, is in Table 1. Here "bt" is an abbreviation for *broaderTransitive* relationship.

To evaluate regular path queries, we used only RDF graphs, because code analysis graphs contain only two types of labels. Regular queries over such graphs are meaningless.

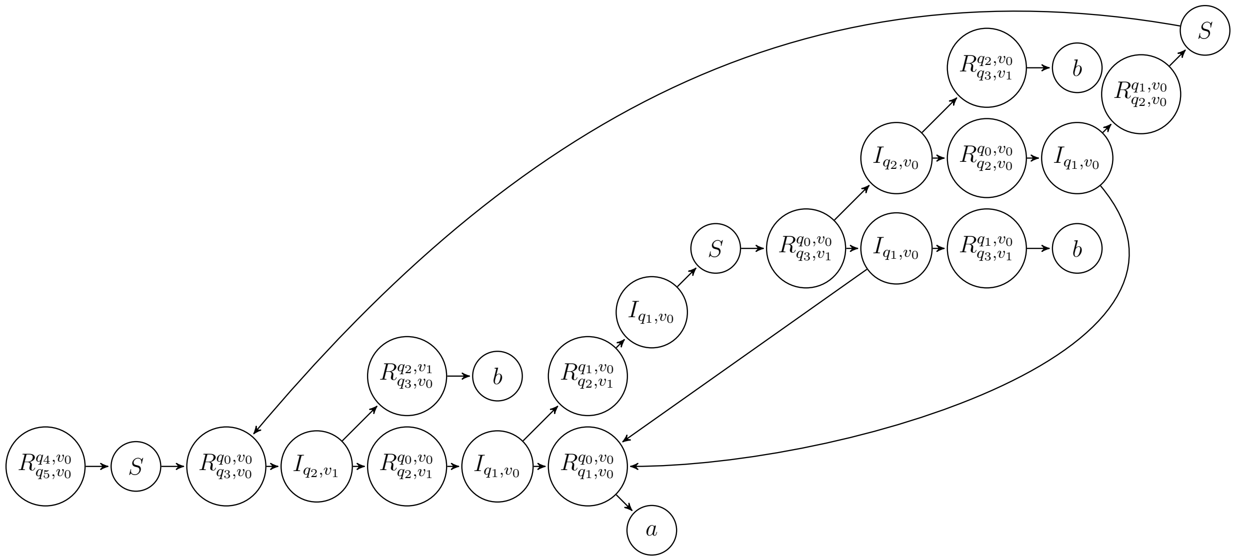


Fig. 3. The SPPF produced for input graph 1 and RSM 2 for all paths from v_0 to v_0.

Table 1. RDF graphs for evaluation: number of the vertices and edges, and number of the edges with specific label.

Graph name	$\|V\|$	$\|E\|$	#subClassOf	#type	#bt
Core	1 323	2 752	178	0	0
Pathways	6 238	12 363	3 117	3 118	0
Go_hierarchy	45 007	490 109	490 109	0	0
Enzyme	48 815	86 543	8 163	14 989	8 156
Eclass	239 111	360 248	90 962	72 517	0
Geospecies	450 609	2 201 532	0	89 065	20 867
Go	582 929	1 437 437	94 514	226 481	0
Taxonomy	5 728 398	14 922 125	2 112 637	2 508 635	0

3.2. Regular Queries

Regular queries were generated using a well-established set of templates for RPQ algorithms evaluation. Namely, we use templates presented in Table 2 in [25] and in Table 5 in [26]. We select four non-trivial templates (that contain compositions of Kleene star and union) that are expressible in Cypher syntax to be able to compare the native Neo4j querying algorithm with our solution. Used templates are presented in equations 1, 2, 3, and 4. Respective path patterns expressed in Cypher are presented in equations 5, 6, 7, and 8. Note that while Cypher's power is limited, our solution can handle arbitrary RPQs. We generate one query for each template and each graph. The most frequent relations from the given graph were used as symbols in the query template.

$$reg_1 = (a \mid b)^* \quad (1) \qquad reg_3 = (a \mid b \mid c)^+ \quad (3)$$

$$reg_2 = a^* \cdot b^* \quad (2) \qquad reg_4 = (a \mid b)^+ \cdot (c \mid d)^+ \quad (4)$$

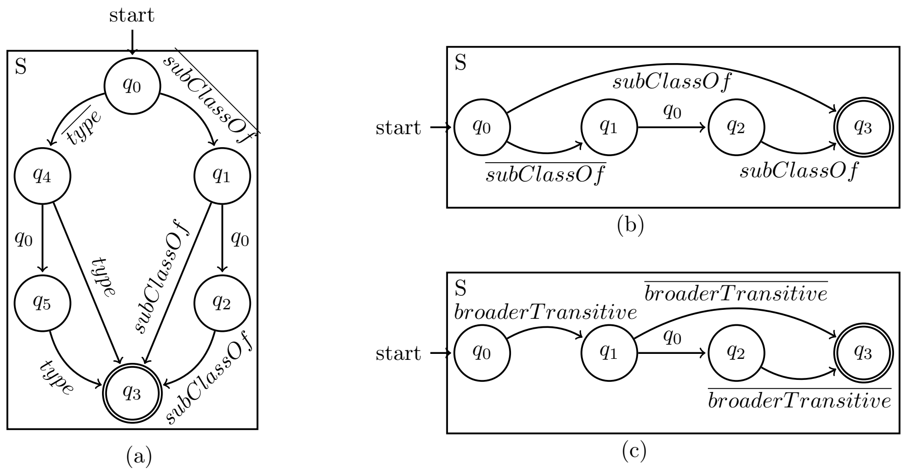


Fig. 4. RSMs for queries: (a) G_1 (10), (b) G_2 (9), and (c) *Geo* (11).

$$reg_1^{\text{N4j}} = \texttt{()-[:a | :b]->\{0,\}()} \tag{5}$$
$$reg_2^{\text{N4j}} = \texttt{()-[:a]->\{0,\}()-[:b]->\{0,\}()} \tag{6}$$
$$reg_3^{\text{N4j}} = \texttt{()-[:a | :b | :c]->\{1,\}()} \tag{7}$$
$$reg_4^{\text{N4j}} = \texttt{()-[:a | :b]->\{1,\}()-[:c | :d]->\{1,\}()} \tag{8}$$

Also note that *go_hierarchy* graph is excluded from evaluation because it contains only one type of edge, so it is impossible to express meaningful queries over it.

3.3. Context-Free Queries

All queries used in our evaluation are variants of the *same-generation query*. For the *RDF* graphs we use the same queries that were used for CFPQ algorithms evaluation in other works [1, 2]: G_1 (10), G_2 (9), and *Geo* (11). The queries are expressed as context-free grammars where S is a nonterminal, *subClassOf, type, broaderTransitive,* $\overline{subClassOf}$, $\overline{type}$, $\overline{broaderTransitive}$ are terminals or the labels of the edges. We denote the inverse of an x relation and the respective edge as $\overline{x}$.

$$S \to \overline{subClassOf}\ \ S\ \ subClasOf \mid subClassOf \tag{9}$$

$$\begin{aligned} S \to & \overline{subClassOf}\ \ S\ \ subClasOf \mid \overline{type}\ \ S\ \ type \\ & \mid \overline{subClassOf}\ \ subClassOf \mid \overline{type}\ type \end{aligned} \tag{10}$$

$$\begin{aligned} S \to & broaderTransitive\ \ S\ \ \overline{broaderTransitive} \\ & \mid broaderTransitive\ \ \overline{broaderTransitive} \end{aligned} \tag{11}$$

Respective RSMs are presented in Fig. 4 a for G_1, Fig. 4 b for G_2, and Fig. 4 c for *Geo*.

3.4. Scenarios Description

We evaluate the proposed solution on the *multiple sources reachability* scenario. We assume that the size of the starting set is significantly less than the number of the input graph vertices. This limitation looks reasonable in practical cases.

The starting sets for the multiple sources querying are generated from all vertices of a graph with a random permutation. We use chunks of size 1, 10, 100. For graphs with less than 10 000 vertices, all vertices were used for querying. For graphs with from 10 000 to 100 000 vertices, 10% of vertices were considered starting ones. For the graphs with more than 100 000 vertices, only 1% of vertices were considered. We use the same sets for all cases in all experiments to be able to compare results.

To check the correctness of our solution and to force the result stream, we compute the number of reachable pairs for each query.

3.5. *Evaluation Environment*

We ran all experiments on a PC with Ubuntu 20.04 installed. It has an Intel Core i7-6700 CPU, 3.4GHz, 4 threads (hyper-threading is turned off), and DDR4 64Gb RAM. We use OpenJDK 64-Bit Server VM Corretto-17.0.8.8.1 (build 17.0.8.1+8-LTS, mixed mode, sharing). JVM was configured to use 55Gb of heap memory: both `xms` and `xmx` are set to 55Gb.

We use Neo4j 5.12.0. Almost all configurations of Neo4j are default. We only set `memory_transaction_global_max_size` to `0`, which means unlimited memory usage per transaction.

As a competitor for our implementation, we use a linear algebra-based solution, integrated to RedisGraph by Arseniy Terekhov et al. and described in [27] and we use the configuration described in it for RedisGraph evaluation in our work.

4. PERFORMANCE OF GLL ON QUERIES IN BNF AND EBNF

((TODO is it relevant for neo4j paper? If no, maybe remove?))

As discussed above, different ways to specify context-free grammars can be used to specify the query. The basic one is BNF (see definition **??**), the more expressive (but not more powerful) is EBNF (see definition **??**). EBNF is not only more expressive, but potentially allows one to improve the performance of the query evaluation because it avoids stack usage by replacing some recursive rules with the Kleene star. RSMs allow one to natively represent grammars in EBNF and can be handled by GLL as described in section **??**.

We implemented and evaluated two versions of the GLL-based CFPQ algorithm [28]: one operates with grammar in BNF, and another one operates with grammar in EBNF and utilizes RSM to represent it. At this step, we use simple data structures for graph and query representations, tuned for our algorithm. Both versions were evaluated in reachability-only mode to estimate performance differences and to choose the best one to integrate with Neo4j.

4.1. *Evaluation*

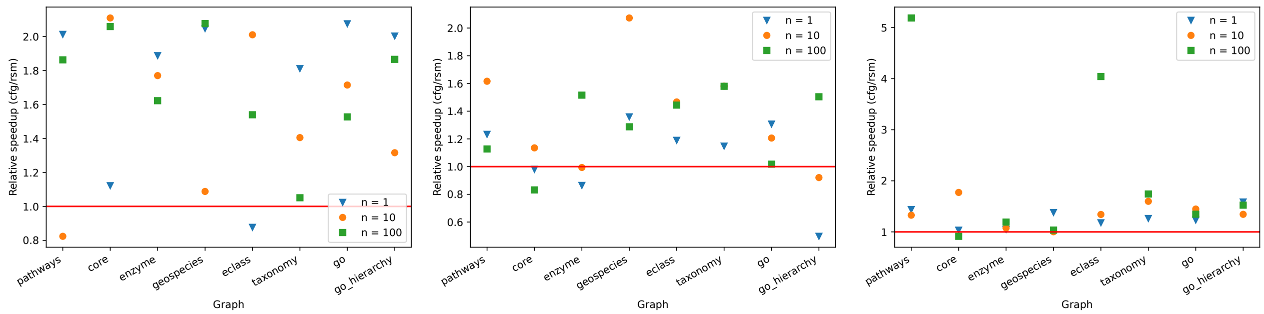
To assess the applicability of the proposed solution, we evaluate it on a number of real-world graphs and queries described in section 3.

We compare the performance of the evaluation of queries in *reachability-only* mode with different sizes of the start vertex set to estimate the speedup of the RSM-based version relative to the BNF-based one.

The experimental study was conducted as follows.

- For all graphs, queries, and start vertex sets, described in section 3, we measure evaluation time for both versions.
- Average time for each start vertex set size was calculated. Thus, for each graph, query, and start vertex set size, we have an average time of respective query evaluation.
- Speedup as a ratio of BNF-based version evaluation time to RSM-based version evaluation time was calculated.

Results presented in Fig. 5 a, Fig. 5 b, and Fig. 5c for queries G_1, G_2 and Geo respectively. We can see that in almost all cases the RSM-based version is faster than the BNF-based one. While in most cases speedup is not greater than 2, in some cases it can be more than 5 (see Fig. 5 c: graph *pathways*, grammar Geo). Average speedup over all graphs and grammars is 1.5. So we can conclude that RSM-based GLL demonstrates better performance than the BNF-based one on average.



(a) (b) (c)

Fig. 5. Multiple-source CFPQ reachability speedup (RSM over CFG) on RDF graphs for (a) G_1, (b) G_2, and (c) *Geo* queries.

5. CFPQ FOR NEO4J

In this section we provide details on the integration of the GLL-based CFPQ to Neo4j graph database. We choose the RSM-based version because our comparison (see section 4.4.1) shows that it is faster than the BNF-based one.

Also, we provide results of the implemented solution evaluation which show that, first of all, the provided solution is faster than a similar linear algebra-based solution for RedisGraph. Also, we show that on RPQs our solution is compatible with the Neo4j native one and in some cases evaluates queries successfully while the native solution fails with an *OutOfMemory* exception.

5.1. Implementation Details

Neo4j stored procedure is a mechanism through which query language can be extended by writing custom code in Java in such a way that it can be called directly via Cypher.

We implemented a Neo4j stored procedure which solves the reachability problem for the given set of the start vertices and the given query. The procedure can be called as follows: `CALL cfpq.gll.getReachabilities(nodes, q)` where `nodes` is a collection of start nodes, and `q` is a string representation (or description) of RSM specified over relation types. The result of the given procedure is a stream of reachable pairs of nodes. Note that the expressive power of our solution is limited: we cannot use the full power of Cypher inside our constraints. For example, we cannot specify constraints on the vertices inside our constraints.

We implemented a wrapper for Neo4j. Communication with the database is done using the Neo4j Native Java API. So, we used an embedded database, which means it is run inside of the application and is not used as an external service.

Along with the existing modifications of GLL, we made a Neo4j-specific one. Neo4j return result should be represented as a `Stream` and it is important to prevent early stream forcing, thus we changed all GLL internals to ensure that. This also has an added benefit that the query result is a stream, and thus it is possible to get the results on demand.

5.2. Evaluation

To assess the applicability of the proposed solution, we evaluate it on a number of real-world graphs and queries. To estimate relative performance, we compare our solution with the matrix-based CFPQ algorithm implemented in RedisGraph by Arseniy Terekhov et al in [27]. Also, we compare the performance of the query evaluation in *reachability-only* mode on regular path queries with the native Neo4j solution.

The results of the context-free path query evaluations are presented in Fig. 6 a for G_1, Fig. 6 b for G_2, and Fig. 6 c for *Geo*.

The results show that query evaluation time depends not only on a graph size or its sparsity, but also on the inner structure of the graph. For example, a relatively small graph *go_hierarchy* fully consists of edges used in queries G_1 and G_2, thus evaluation time for these queries is significantly

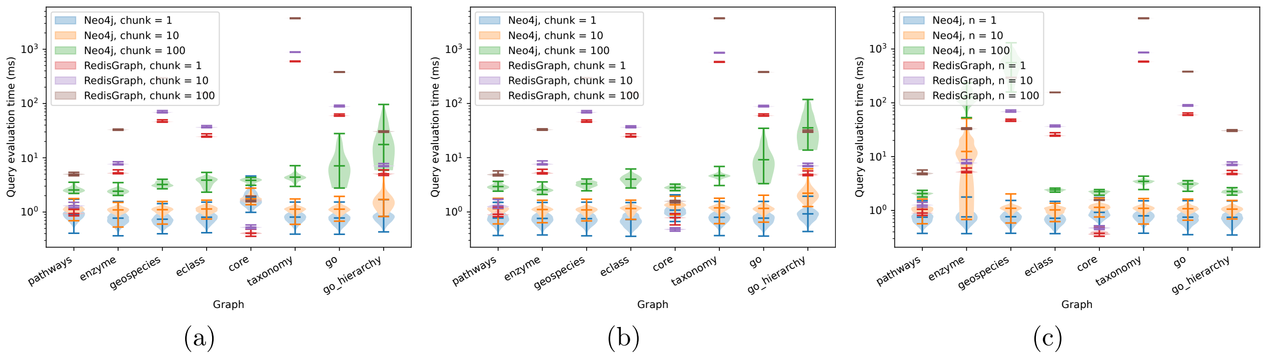


Fig. 6. Multiple source CFPQ reachability results for the queries: (a) G_1, (b) G_2, (c) *Geo* related to RDF analysis.

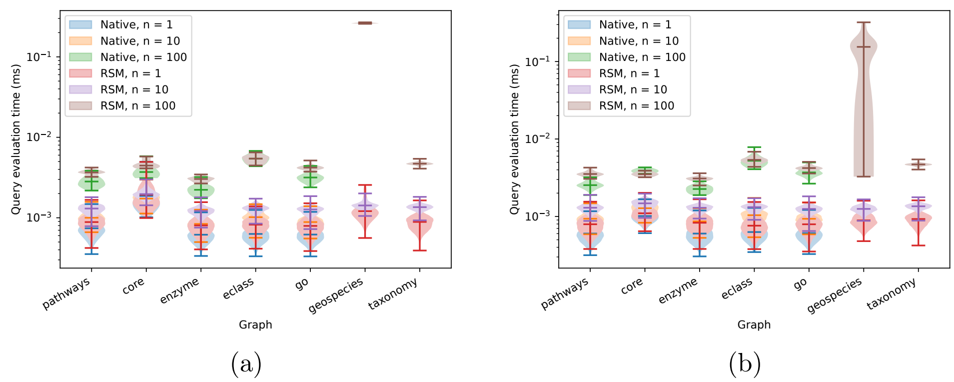


Fig. 7. Multiple source RPQ reachability results for queries related to RDF analysis: (a) reg_1 query, (b) reg_2 query (native solution failed with OOM on the last two graphs).

bigger than for some bigger but more sparse graphs, for example, for the *eclass* graph. Especially for a relatively large starting vertex set. Note that the creation of the relevant metrics for the CFPQ query evaluation time prediction is a challenging problem by itself and should be tackled in the future.

Also, we can see that in almost all cases the proposed solution is significantly faster than RedisGraph (in orders of magnitude in some cases). At the same time, in some cases (see results for the graph *core* and all queries) RedisGraph outperforms our solution. Moreover, it can be seen that evaluation time for RedisGraph is more predictable. For our solution, in some cases, execution time highly depends on the start set. For example, see Fig. 6 c, graph *enzyme*.

The particularly important scenario is the case when the start set is a single vertex. The results of the `single-source reachability` show that such queries are reasonably fast: median time is about few milliseconds for all graphs and all queries. Note that even for single source queries, evaluation time highly depends on the graph structure: evaluation time on *core* graph is significantly bigger than for all other graphs for all queries. Note that *core* is the smallest graph in terms of the number of the nodes and edges. Again, to provide a reliable metric to predict query execution time is a non-trivial task. Moreover, time grows with the size of the chunk, as expected.

Partial results for RPQ evaluation are presented in Fig. 7 a and Fig. 7 b for reg_1 (defined in 1) and reg_2 (defined in 2) respectively. For queries reg_3 (defined in 3) and reg_4 (defined in 4) we get similar results. Note that on *geospecies* and *taxonomy* graphs, the native solution failed with the *OutOfMemory* exception, while our solution evaluates queries successfully.

We can see that the proposed solution is slightly slower than the native Neo4j algorithm, but not dramatically: typically less than two times. Moreover, in some cases, our solution is comparable with the native one (see Fig. 7 a and Fig. 7 b, graph *eclass*), and in some cases, our solution is faster than the native one (see Fig. 7 b, graph *core*).

Finally, we conclude that the proposed GLL-based solution can serve as an alternative to linear algebra-based CFPQ algorithms and is suitable for real-world graph analysis systems: our evaluation shows that the proposed solution outperforms the linear algebra-based one. Furthermore, we show that the proposed solution can be used as a universal algorithm for both RPQ and CFPQ.

6. RELATED WORK

The idea to use context-free grammars as constraints in the path finding problem in the graph databases was introduced and explored by Mihalis Yannakakis in [29]. Later, Thomas Reps et al. developed the same idea into a general framework for static code analysis [30]. This framework, called Context-Free Language Reachability (CFL-r), is widely used and actively studied. The landscape analysis of the area was recently provided by Andreas Pavlogiannis in [31]. In the context of the graph databases, the most recent systematic comparison of different CFPQ algorithms was done by Jochem Kuijpers et al. A set of CFPQ algorithms was implemented and evaluated using Neo4j as a graph storage. Results were presented in [2]. It was concluded that the existing algorithms are not performant enough to be used for the real-world data analysis.

Regarding graph databases, CFPQ was applied in such graph analysis related tasks as biological data analysis [4], data provenance [5], hierarchy analysis in RDF data [3, 32].

Multiple CFPQ algorithms are based on different parsing algorithms and techniques. For example, an approach based on parsing combinators was proposed by Ekaterina Verbitskaia et al. in [33]. Several algorithms based on LL-like and LR-like techniques were developed by Ciro M. Medeiros et al. in [32, 34, 35]. Also, CFPQ algorithms were investigated by Phillip Bradford in [36, 37] and Charles B. Ward et al. in [38]. An algorithm based on matrix equations was proposed by Yuliya Susanina in [39]. Paths extraction problem was studied by Jelle Hellings in [12].

A set of linear-algebra-based CFPQ algorithms was developed by Rustam Azimov et al., including all-path and single-path variants, proposed in [13] and [11], respectively. The Kronecker product-based algorithm [40] was proposed by Egor Orachev et al., and a multiple-source CFPQ algorithm for RedisGraph was proposed by Arseniy Terekhov et al. in [27].

Recursive state machines were studied in the context of CFPQ in several papers, including [40] where Egor Orachev et al. use RSM to specify context-free constraints, Yuxiang Lei et al. [41] propose to use RSM to specify path constraints, and [42] where Swarat Chaudhuri proposes a slightly subcubic algorithm for the reachability problem for recursive state machines — the equivalent to CFPQ problem.

GLL was introduced by Elizabeth Scott and Adrian Johnstone in [20]. A number of modifications of the GLL algorithm were further proposed, including the version which supports EBNF without its transformation [43] and the version which uses binary subtree sets [23] instead of SPPF. The latest version simplifies the algorithm and avoids the overhead of the explicit graph construction. Within it, the optimized and simplified OOP-friendly version of GLL was proposed by Ali Afroozeh and Anastasia Izmaylova in [16].

7. CONCLUSION AND FUTURE WORK

In this work, we presented the GLL-based context-free path querying algorithm for the Neo4j graph database. The implementation is available on GitHub: `https://github.com/vadyushkins/cfpq-gll-neo4j-procedure`.

Our solution uses Neo4j for graph storage, but the query language should be extended to support context-free constraints to make it useful. Both the extension of Cypher and the integration of our algorithm with the query engine are non-trivial challenges left for future work.

While GLL-based CFPQ can potentially be used to solve the *all-paths* problem, currently we have implemented the procedure only for the reachability. The choice of effective strategies to enumerate paths and implementation of them is a direction for future research.

The most important direction for future work is to find a way to provide an incremental version of the GLL-based CFPQ algorithm to avoid full query reevaluation when the graph is only slightly changed. While our solution is based on the well-known parsing algorithm and there are solutions for incremental parsing, development of the efficient incremental version of the GLL-based CFPQ algorithm is a challenging problem left for future research.

Another direction is to create a parallel version of the GLL-based CFPQ algorithm to improve its performance on huge graphs. Although it seems natural to handle descriptors in parallel, the algorithm operates over global structures, and the naïve implementation of this idea leads to a significant overhead in synchronization.

ACKNOWLEDGMENTS

We thank Adrian Johnstone for pointing out the Generalized LL algorithm in our discussion at Parsing@SLE–2013, which motivated the development of the presented solution. We thank George Fletcher for the discussion of the evaluation of different CFPQ algorithms for Neo4j.

FUNDING

The study was funded by the St. Petersburg State University (Grant No. 116636233).

CONFLICT OF INTEREST

The authors of this work declare that they have no conflicts of interest.

REFERENCES

[1] R. Azimov and S. Grigorev, 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, 2018) pp. 5:1–5:10.

[2] J. Kuijpers, G. Fletcher, N. Yakovets, and T. Lindaaker, 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, 2019) pp. 121–132.

[3] X. Zhang, Z. Feng, X. Wang, G. Rao, and W. Wu, Context-free path queries on rdf graphs, in *The Semantic Web – ISWC 2016*, edited by P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, and Y. Gil (Springer International Publishing, Cham, 2016) pp. 632–648.

[4] P. Sevon and L. Eronen, Subgraph queries by context-free grammars, Journal of Integrative Bioinformatics **5**, 157 (2008).

[5] H. Miao and A. Deshpande, Understanding data science lifecycle provenance via graph segmentation and summarization, in *2019 IEEE 35th International Conference on Data Engineering (ICDE)* (2019) pp. 1710–1713.

[6] X. Zheng and R. Rugina, Demand-driven alias analysis for c, in *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08 (ACM, New York, NY, USA, 2008) pp. 197–208.

[7] J. Rehof and M. Fähndrich, Type-based flow analysis: From polymorphic subtyping to cfl-reachability, SIGPLAN Not. **36**, 54 (2001).

[8] F. C. Santos, U. S. Costa, and M. A. Musicante, A bottom-up algorithm for answering context-free path queries in graph databases, in *Web Engineering*, edited by T. Mikkonen, R. Klamma, and J. Hernández (Springer International Publishing, Cham, 2018) pp. 225–233.

[9] J. Hellings, Conjunctive context-free path queries, in *Proceedings of ICDT'14* (2014) pp. 119–130.

[10] C. M. Medeiros, M. A. Musicante, and U. S. Costa, Efficient evaluation of context-free path queries for graph databases, in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18 (Association for Computing Machinery, New York, NY, USA, 2018) pp. 1230–1237.

[11] A. Terekhov, A. Khoroshev, R. Azimov, and S. Grigorev, Context-free path querying with single-path semantics by matrix multiplication, in *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, GRADES-NDA'20 (Association for Computing Machinery, New York, NY, USA, 2020).

[12] J. Hellings, Explaining results of path queries on graphs, in *Software Foundations for Data Interoperability and Large Scale Graph Data Analytics*, edited by L. Qin, W. Zhang, Y. Zhang, Y. Peng, H. Kato, W. Wang, and C. Xiao (Springer International Publishing, Cham, 2020) pp. 84–98.

[13] R. Azimov, I. Epelbaum, and S. Grigorev, Context-free path querying with all-path semantics by matrix multiplication, in *Proceedings of the 4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, GRADES-NDA '21 (Association for Computing Machinery, New York, NY, USA, 2021).

[14] O. Rodriguez-Prieto, A. Mycroft, and F. Ortin, An efficient and scalable platform for java source code analysis using overlaid graph representations, IEEE Access **8**, 72239 (2020).

[15] S. Grigorev and A. Ragozina, 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, 2017) pp. 10:1–10:7.

[16] A. Afroozeh and A. Izmaylova, Faster, practical gll parsing, in *Compiler Construction*, edited by B. Franke (Springer Berlin Heidelberg, Berlin, Heidelberg, 2015) pp. 89–108.

[17] Iguana parsing framework, `https://iguana-parser.github.io/`, accessed: 12.11.2024.

[18] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis, Analysis of recursive state machines, ACM Trans. Program. Lang. Syst. **27**, 786 (2005).

[19] V. Abzalov, V. Pogozhelskaya, V. Kutuev, and S. Grigorev, Gll-based context-free path querying for neo4j (2023).

[20] E. Scott and A. Johnstone, Gll parsing, Electronic Notes in Theoretical Computer Science **253**, 177 (2010), proceedings of the Ninth Workshop on Language Descriptions Tools and Applications (LDTA 2009).

[21] E. Scott and A. Johnstone, Gll parse-tree generation, Science of Computer Programming **78**, 1828 (2013), special section on Language Descriptions Tools and Applications (LDTA'08 & '09) & Special section on Software Engineering Aspects of Ubiquitous Computing and Ambient Intelligence (UCAmI 2011).

[22] J. G. Rekers, *Parser generation for interactive environments*, Ph.D. thesis, Citeseer (1992).

[23] E. Scott, A. Johnstone, and L. T. van Binsbergen, Derivation representation using binary subtree sets, Science of Computer Programming **175**, 63 (2019).

[24] Cfpq_data: A public set of graphs and grammars for cfpq algorithms evaluation, `https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_Data`, accessed: 27.09.2024.

[25] A. Pacaci, A. Bonifati, and M. T. Özsu, Regular path query evaluation on streaming graphs, in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20 (Association for Computing Machinery, New York, NY, USA, 2020) pp. 1415–1430.

[26] X. Wang, S. Wang, Y. Xin, Y. Yang, J. Li, and X. Wang, Distributed pregel-based provenance-aware regular path query processing on RDF knowledge graphs, World Wide Web **23**, 1465 (2019).

[27] A. Terekhov, V. Pogozhelskaya, V. Abzalov, T. Zinnatulin, and S. V. Grigorev, Multiple-source context-free path querying in terms of linear algebra, in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, edited by Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis, and F. Guerra (OpenProceedings.org, 2021) pp. 487–492.

[28] Gll-based cfpq algorithm implementation, `https://github.com/vadyushkins/kotgll`, accessed: 12.11.2024.

[29] M. Yannakakis, Graph-theoretic methods in database theory, in *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '90 (Association for Computing Machinery, New York, NY, USA, 1990) pp. 230–242.

[30] T. Reps, S. Horwitz, and M. Sagiv, Precise interprocedural dataflow analysis via graph reachability, in *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95 (Association for Computing Machinery, New York, NY, USA, 1995) pp. 49–61.

[31] A. Pavlogiannis, Cfl/dyck reachability: An algorithmic perspective, ACM SIGLOG News **9**, 5 (2023).

[32] C. M. Medeiros, M. A. Musicante, and U. S. Costa, Querying graph databases using context-free grammars, Journal of Computer Languages **68**, 101089 (2022).

[33] E. Verbitskaia, I. Kirillov, I. Nozkin, and S. Grigorev, Parser combinators for context-free path querying, in *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala*, Scala 2018 (Association for Computing Machinery, New York, NY, USA, 2018) pp. 13–23.

[34] C. M. Medeiros, M. A. Musicante, and U. S. Costa, An algorithm for context-free path queries over graph databases, in *Proceedings of the 24th Brazilian Symposium on Context-Oriented Programming and Advanced Modularity*, SBLP '20 (Association for Computing Machinery, New York, NY, USA, 2020) pp. 40–47.

[35] C. M. Medeiros, M. A. Musicante, and U. S. Costa, Ll-based query answering over rdf databases, Journal of Computer Languages **51**, 75 (2019).

[36] P. G. Bradford and D. A. Thomas, Labeled shortest paths in digraphs with negative and positive edge weights, RAIRO - Theoretical Informatics and Applications **43**, 567 (2009).

[37] P. G. Bradford, Efficient exact paths for dyck and semi-dyck labeled path reachability (extended abstract), in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)* (2017) pp. 247–253.

[38] C. B. Ward, N. M. Wiegand, and P. G. Bradford, A distributed context-free language constrained shortest path algorithm, in *2008 37th International Conference on Parallel Processing* (2008) pp. 373–380.

[39] Y. Susanina, Context-free path querying via matrix equations, in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20 (Association for Computing Machinery, New York, NY, USA, 2020) pp. 2821–2823.

[40] E. Orachev, I. Epelbaum, R. Azimov, and S. Grigorev, Context-free path querying by kronecker product, in *Advances in Databases and Information Systems*, edited by J. Darmont, B. Novikov, and R. Wrembel (Springer International Publishing, Cham, 2020) pp. 49–59.

[41] Y. Lei, Y. Sui, S. H. Tan, and Q. Zhang, Recursive state machine guided graph folding for context-free language reachability, Proc. ACM Program. Lang. **7**, 10.1145/3591233 (2023).

[42] S. Chaudhuri, Subcubic algorithms for recursive state machines, in *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08 (Association for Computing Machinery, New York, NY, USA, 2008) pp. 159–169.

[43] E. Scott and A. Johnstone, Gll syntax analysers for ebnf grammars, Science of Computer Programming **166**, 120 (2018).