



# Implementation and experimental study of GLL algorithm with Neo4j graph database

Pogozhelskaya Vlada Vladimirovna, 18.Б11-ММ

**Supervisor:** Ph.D. of Physico-mathematical Sciences, Associate Professor of the Department of Informatics of St Petersburg State University S.V. Grigoriev

St Petersburg State University

April 30, 2022

- Graph data model
  - ▶ Basic entities — graph vertices
  - ▶ Relationships between entities are graph edges
- Graph databases
  - ▶ The most popular is Neo4j
  - ▶ Only regular queries are partially supported
- Context-free constraints
  - ▶ Strictly more expressive than the regular one
  - ▶ Widely used in bioinformatics, RDF file analysis, static code analysis

# Context-free path querying problems

## All-paths CFPQ problem and reachability CFPQ problem

Let be:

- Context-free grammar  $\mathbb{G} = \langle N, \Sigma, P, S \rangle$
- Directed graph  $\mathbb{D} = \langle V, E, T \rangle$
- Set of start vertices  $V_S \subseteq V$  and set of final vertices  $V_F \subseteq V$

### All-paths problem:

- Find all paths  $\pi = (e_1, \dots, e_{n-1}, e_n)$ ,  $e_k = (v_{k-1}, t_k, v_k)$  in graph  $\mathbb{D}$ , such as  $I(\pi) = t_1 t_2 \dots t_n \in L(\mathbb{G})$  and  $v_0 \in V_S$ ,  $v_n \in V_F$

### Reachability problem:

- Find all pairs  $\{(v_0, v_n) \mid \text{exists a path } \pi = (e_1, \dots, e_{n-1}, e_n), e_k = (v_{k-1}, t_k, v_k) \text{ in } \mathbb{D}, v_0 \in V_S, v_n \in V_F, I(\pi) = t_1 t_2 \dots t_n \in L(\mathbb{G})\}$

- The problem of poor performance of CFPQ algorithms was formulated by Jochem Kuijpers as a result of an attempt to extend Neo4j<sup>1</sup>
- Later, the matrix-based CFPQ algorithm showed high performance on real-world data

---

<sup>1</sup>An Experimental Study of Context-Free Path Query Evaluation Methods / Jochem Kuijpers, George Fletcher, Nikolay Yakovets, Tobias Lindaaker / SSDBM '19

# Goal and tasks

**The aim** of this work is to improve existing CFPQ algorithm for the Neo4j graph database<sup>2</sup> and evaluate it

## Tasks:

- To make initial experiments and analysis of existing algorithm to identify performance problems
- To refactor the code of the current implementation of the GLL-based CFPQ algorithm in order to identifying and eliminate performance problems of the current implementation of the algorithm
- To provide an ability to obtain information about both reachability CFPQ problem in a graph and all paths CFPQ problem
- To evaluate the resulting algorithm on real-world graphs and to compare it with an existing one

---

<sup>2</sup>Algorithm implementation: <https://github.com/JetBrains-Research/GLL4Graph/tree/8be59e6b314a1bfa646b119f751b3f28ad34ac64>

## Generalized LL algorithm (GLL)

- Supports the entire class of context-free languages
- To reconstruct the paths, the Shared Packed Parse Forest (SPPF) is used

## Proposed solution

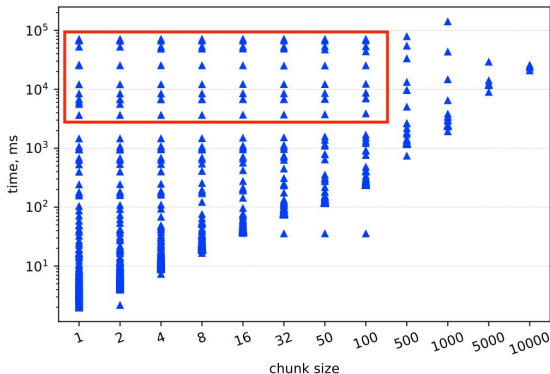
- Based on GLL algorithm implementation in Iguana<sup>3</sup> project made at CWI Amsterdam in 2016
- Neo4j graph database is used as a graph storage
- The solution was integrated with Neo4j using Native Java API

---

<sup>3</sup>Repository of Iguana project: <https://github.com/iguana-parser/iguana>

# Initial experiments

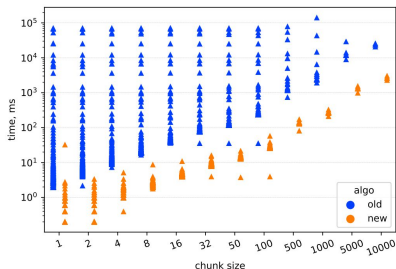
An unexpected deterioration in the behavior of the resulting solution was revealed in the multiple-source scenario



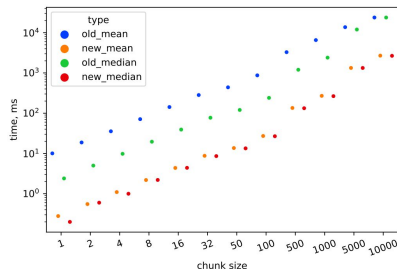
Grammar  $G_2$  and Enzyme

# Modifications

- The modification of the way to get vertices from Neo4j graph database
- The optimization of transition between vertices while graph traversal
- The optimization of procedure for getting edge labels
- The change of result data representation



Query time



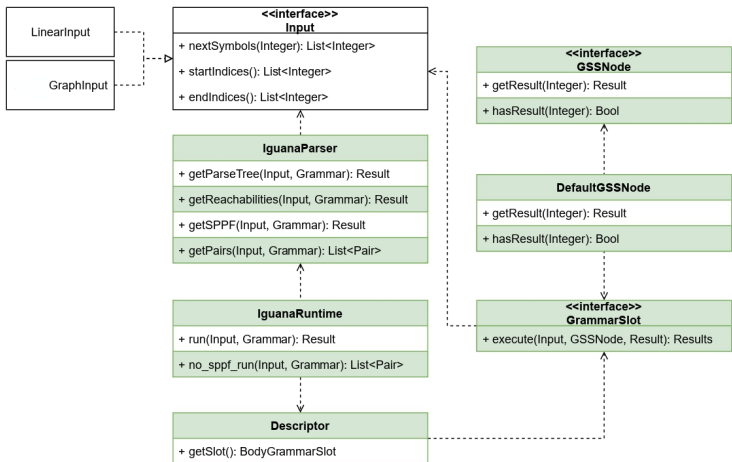
Median and mean time

Grammar  $G_2$  and Enzyme



# Extension to solve the reachability problem

The ability to switch between the SPPF construction and reachability facts calculation was provided



Architecture of the proposed solution

# Experimental study setup

## Data

- **RDF Graphs**

- ▶ *Grammars*

$$S \rightarrow \overline{subClassOf} \ S \ subClassOf \mid \overline{type} \ S \ type \\ \mid \overline{subClassOf} \ subClassOf \mid \overline{type} \ type \quad (G_1)$$

$$S \rightarrow \overline{subClassOf} \ S \ subClassOf \mid subClassOf \quad (G_2)$$

$$S \rightarrow broaderTransitive \ S \ \overline{broaderTransitive} \\ \mid broaderTransitive \ \overline{broaderTransitive} \quad (Geo)$$

- **Program analysis graphs**

- ▶ *Grammar*

$$M \rightarrow \overline{d} \ V \ d \\ V \rightarrow (M? \ \overline{a})^* \ M? \ (a \ M?)^* \quad (PointsTo)$$

# All pairs results for graphs related to **RDF analysis**

## Graphs considered

Graphs	V	E	#subClassOf	#type	#broaderTransitive
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

## Results

Graphs	$G_1$		$G_2$		$Geo$	
	time (sec)	#answer	time (sec)	#answer	time (sec)	#answer
Core	0,02	204	0,01	214	–	–
Pathways	0,07	884	0,04	3117	–	–
Go hierarchy	3,68	588 976	5,4	738 937	–	–
Enzyme	0,22	396	0,17	8163	5,7	14 267 542
Eclass	1,5	90 994	0,98	96 163	–	–
Geospecies	2,87	85	2,65	0	145,8	226 669 749
Go	5,56	640 316	4,2	659 501	–	–
Taxonomy	45,47	151 706	36,07	2 112 637	–	–

# All pairs results for graphs related to **RDF analysis**

## Graphs considered

Graphs	V	E	#subClassOf	#type	#broaderTransitive
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

## Results

Graphs	$G_1$		$G_2$		$Geo$	
	time (sec)	#answer	time (sec)	#answer	time (sec)	#answer
Core	0,02	204	0,01	214	–	–
Pathways	0,07	884	0,04	3117	–	–
Go hierarchy	3,68	588 976	5,4	738 937	–	–
Enzyme	0,22	396	0,17	8163	5,7	14 267 542
Eclass	1,5	90 994	0,98	96 163	–	–
Geospecies	2,87	85	2,65	0	145,8	226 669 749
Go	5,56	640 316	4,2	659 501	–	–
Taxonomy	45,47	151 706	36,07	2 112 637	–	–

# All pairs results for graphs related to **static code analysis**

## Graphs considered

Graphs	$ V $	$ E $	#a	#d
Apache	1 721 418	1 510 411	362 799	1 147 612
Block	3 423 234	2 951 393	669 238	2 282 155
Net	4 039 470	3 500 141	807 162	2 692 979
Postgre	5 203 419	4 678 543	1 209 597	3 468 946
Init	2 446 224	2 112 809	481 994	1 630 815
Drivers	4 273 803	3 707 769	858 568	2 849 201
Kernel	11 254 434	9 484 213	1 981 258	7 502 955

## Results

Graphs	<i>PointsTo</i>		
	GLL time (sec)	#answer	Matrix time (sec)
Apache	—	92 806 768	536,703
Block	113,01	53 514 095	123,88
Net	160,64	8 833 403	206,29
Postgre	—	90 661 446	969,89
Init	87,25	3 783 769	45,84
Drivers	371,18	18 825 025	279,39
Kernel	614,05	16 747 731	378,05

# All pairs results for graphs related to **static code analysis**

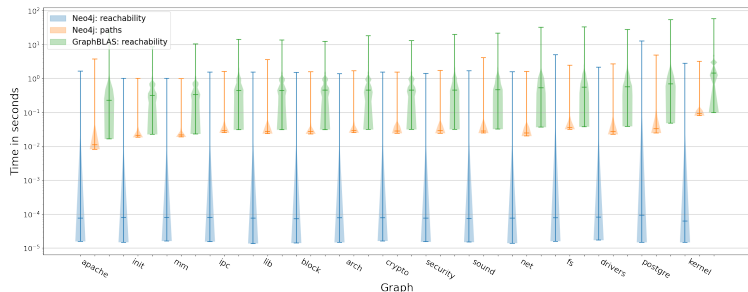
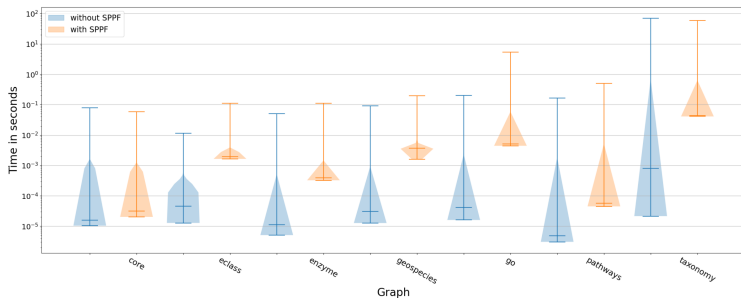
## Graphs considered

Graphs	$ V $	$ E $	#a	#d
Apache	1 721 418	1 510 411	362 799	1 147 612
Block	3 423 234	2 951 393	669 238	2 282 155
Net	4 039 470	3 500 141	807 162	2 692 979
Postgre	5 203 419	4 678 543	1 209 597	3 468 946
Init	2 446 224	2 112 809	481 994	1 630 815
Drivers	4 273 803	3 707 769	858 568	2 849 201
Kernel	11 254 434	9 484 213	1 981 258	7 502 955

## Results

Graphs	<i>PointsTo</i>		
	GLL time (sec)	#answer	Matrix time (sec)
Apache	–	92 806 768	536,703
Block	113,01	53 514 095	123,88
Net	160,64	8 833 403	206,29
Postgre	–	90 661 446	969,89
Init	87,25	3 783 769	45,84
Drivers	371,18	18 825 025	279,39
Kernel	614,05	16 747 731	378,05

# Single-source results



- There were made initial experiments and analysis which confirmed performance problems
- The performance problems in the implementation of the GLL-based CFPQ algorithm were eliminated
- The implementation of GLL-based CFPQ algorithm was extended with ability to solve the reachability CFPQ problem
- The resulting algorithm implementation was evaluated on two sets of real-world graphs: a number of graphs related to RDF analysis and a number of graph related to static code analysis problem for both the *all pairs* and the *multiple sources* scenarios. The evaluation shows that the proposed algorithm is more than 45 times faster than the previous solution for Neo4j