VLDB 2021 PhD Workshop

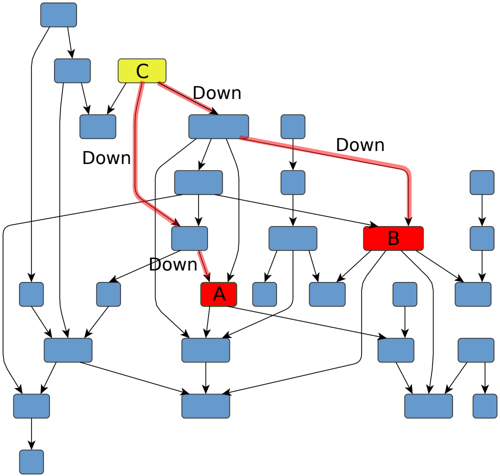
Context-Free Path Querying In Terms of Linear Algebra

Rustam Azimov
supervised by Semyon Grigorev

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

August 16, 2021

Context-Free Path Querying (CFPQ)



Context-free languages as constraints on the paths

- Are nodes A and B on the same level of hierarchy?
- Is there a path of form $\overline{\textbf{Down}}^n\,\textbf{Down}^n$ between A and B?
- Find all paths of form $\overline{\textbf{Down}}^n\,\textbf{Down}^n$ between A and B
- Context-free grammar:
  $SameLvl \to \overline{Down}\ SameLvl\ Down \mid \varepsilon$

CFPQ

- L — context-free language over the alphabet Σ

CFPQ

- L — context-free language over the alphabet Σ
- $G = (V, E, \Sigma')$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $\Sigma' \subseteq \Sigma$

CFPQ

- L — context-free language over the alphabet Σ
- $G = (V, E, \Sigma')$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $\Sigma' \subseteq \Sigma$
- $\pi = v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n$ — path in G
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$

CFPQ

- L — context-free language over the alphabet Σ
- $G = (V, E, \Sigma')$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $\Sigma' \subseteq \Sigma$
- $\pi = v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n$ — path in G
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$
- It is necessary to find information about paths π, such that $\omega(\pi) \in L$

Formulations of the CFPQ Problem

- type of the required information about paths in the graph
  - solution of the reachability problem (relational query semantics)
  - single path search (single-path query semantics)
  - finding all paths in the graph (all-path query semantics)
- fixing a set of source and destination vertices in the graph
  - finding paths between all pairs of vertices (All-Pairs problem)
  - a fixed set of source vertices (Multiple Source problem)
  - single source single destination vertices
- additional path constraints
  - finding the shortest paths
  - finding the simple paths

Applications

- Static code analysis
  - interprocedural points-to analysis
  - interprocedural alias analysis
- Graph database querying
- RDF analysis
- Bioinformatics

Existing Solutions

- Based on various parsing algorithms
  - (G)LL and (G)LR-based algorithms
  - CYK-based algorithm
  - Combinators-based approach to CFPQ
- Yet recent research by Jochem Kuijpers et al. shows that existing solutions are not applicable for real-world graph analysis because of significant
  - running time
  - memory consumption
- Like with solutions of other graph analysis problems
  - irregular access patterns leads to poor locality
  - caching and parallelization are difficult
  - limited portability after applying optimizations

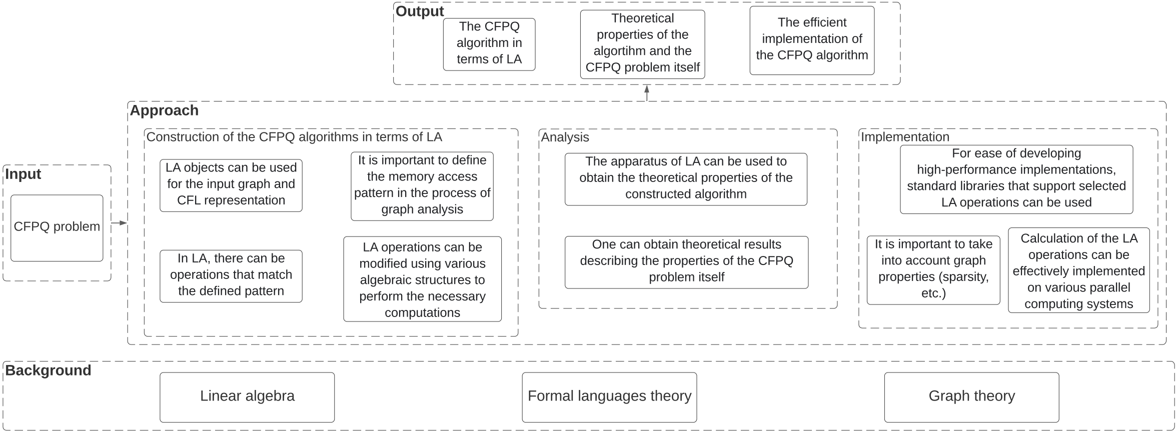
Linear Algebra (LA) For Graph Analysis Problems

- The idea of using a sparse adjacency matrix as a graph representation in graph analysis problems is well-known
- Recently, became very popular the **GraphBLAS API** specification that defines standard building blocks for graph algorithms in the language of LA
- Using efficient libraries that implement it is a good recipe for making a high-performance CFPQ solution if we can reduce the CFPQ problem to LA operations
  - although such reduction was found for a number of graph algorithms (BFS, PageRank, Graph coloring, Connected components, ...)
  - there are many graph algorithms for which it has not been done (DFS, **CFPQ**, ...)

Research Statement

- Aim: to explore the applicability of linear algebra methods to the CFPQ problem for obtaining the efficient implementations using parallel computations
- Objectives:
  - An approach to solving the CFPQ problem using LA methods will be provided
  - Using provided approach, the CFPQ algorithms based on LA operations for relational, single-path, and all-path query semantics will be devised
  - The efficient implementations of the devised algorithms for CFPQ evaluation using parallel computations will be provided, their experimental study on synthetic and real data will be conducted

Proposed Approach



Devised Algorithms: Matrix-Based

- The all-pairs CFPQ algorithms for relational, single-path and all-path query semantics were devised
- Matrix-based CFPQ algorithms
  - Represent the input graph using adjacency matrix
    - boolean matrices for the relational query semantics
    - the additional information are stored for single-path and all-path query semantics
  - Use CF-grammars in normal form for describing the input CFL
  - Construct the semirings for modifying the matrix multiplication operations
  - Explore the graph by computing the transitive closure
  - Resulting matrices are used for restoring found paths

Matrix-Based Algorithm: Relational Query Semantics

Algorithm Context-free path querying algorithm

function EVALCFPQ($D = (V, E, L), G = (\Sigma, N, P)$)
  $n \leftarrow |V|$
  $T \leftarrow \{T^{A_i} \mid A_i \in N, T^{A_i}$ is a matrix $n \times n$, $T^{A_i}_{k,l} \leftarrow \texttt{false}\}$
  for all $(i, x, j) \in E$, $A_k \mid A_k \rightarrow x \in P$ **do** $T^{A_k}_{i,j} \leftarrow \texttt{true}$
  for all $A_k \mid A_k \rightarrow \varepsilon \in P$ **do**
    for all $i \in \{0, \ldots, n-1\}$ **do** $T^{A_k}_{i,i} \leftarrow \texttt{true}$
  while any matrix in T is changing **do**
    for $A_i \rightarrow A_j A_k \in P$ **do** $T^{A_i} \leftarrow T^{A_i} + (T^{A_j} \times T^{A_k})$
  return T

Devised Algorithms: Kronecker Product-Based

- On the contrary, the algorithms in the second group are based on the Kronecker product operation
  - Do not require the transformation of the CF-grammar
    - The transformation to Chomsky normal form leads to at least a quadratic blow-up in grammar size
  - Use Recursive State Machines (RSMs) for describing the input CFL
    - The RSMs can be represented as graphs
  - Evaluate query using the Kronecker product of the corresponding adjacency matrices of the input graph and the RSM

Implementations

- For evaluation we use the following CPU-based implementations of CFPQ algorithms with sparse matrix representation
  - *MtxRel* — for relational query semantics that uses **pygraphblas** — a Python wrapper around the GraphBLAS API
  - *MtxSingle* — for single-path query semantics that also uses **pygraphblas**
  - *MtxAll* — the implementation of the proposed matrix-based algorithm for all-path query semantics which utilizes **SuiteSparse** and our own Python wrapper
  - *Tns* — the implementation of the Kronecker product-based algorithm for all-path query semantics that uses **pygraphblas**

Evaluation Setup

- Ubuntu 18.04, Intel Core i7-6700 CPU, 3.4GHz, DDR4 64Gb RAM
- We use graphs corresponding to real RDFs
- We use same-generation query

Evaluation: CFPQ[1]

Graph	#V	#E	MtxRel		MtxSingle		MtxAll		Tns	
			Time	Mem	Time	Mem	Time	Mem	Time	Mem
pathways	6 238	18 598	0.01	140	0.01	671	0.01	49	0.01	122
go-hierarchy	45 007	980 218	0.09	255	0.84	671	0.35	195	0.24	252
enzyme	48 815	109 695	0.01	181	0.01	217	0.02	61	0.02	132
eclass_514en	239 111	523 727	0.06	181	0.16	216	0.22	126	0.27	193
go	272 770	534 311	0.94	246	0.93	217	1.13	990	1.27	243
geospecies	450 609	2 311 461	7.48	7645	15.54	22941	32.06	44235	26.32	19537
taxonomy	5 728 398	14 922 125	0.72	1175	1.15	2250	3.84	1507	3.56	1776

[1] Time in seconds and memory is measured in megabytes

Evaluation: CFPQ[1]

Graph	#V	#E	MtxRel		MtxSingle		MtxAll		Tns	
			Time	Mem	Time	Mem	Time	Mem	Time	Mem
pathways	6 238	18 598	0.01	140	0.01	671	0.01	49	0.01	122
go-hierarchy	45 007	980 218	0.09	255	0.84	671	0.35	195	0.24	252
enzyme	48 815	109 695	0.01	181	0.01	217	0.02	61	0.02	132
eclass_514en	239 111	523 727	0.06	181	0.16	216	0.22	126	0.27	193
go	272 770	534 311	0.94	246	0.93	217	1.13	990	1.27	243
geospecies	450 609	2 311 461	7.48	7645	15.54	22941	32.06	44235	26.32	19537
taxonomy	5 728 398	14 922 125	0.72	1175	1.15	2250	3.84	1507	3.56	1776

[1] Time in seconds and memory is measured in megabytes

Evaluation: CFPQ[1]

Graph	#V	#E	MtxRel		MtxSingle		MtxAll		Tns	
			Time	Mem	Time	Mem	Time	Mem	Time	Mem
pathways	6 238	18 598	0.01	140	0.01	671	0.01	49	0.01	122
go-hierarchy	45 007	980 218	0.09	255	0.84	671	0.35	195	0.24	252
enzyme	48 815	109 695	0.01	181	0.01	217	0.02	61	0.02	132
eclass_514en	239 111	523 727	0.06	181	0.16	216	0.22	126	0.27	193
go	272 770	534 311	0.94	246	0.93	217	1.13	990	1.27	243
geospecies	450 609	2 311 461	7.48	7645	15.54	22941	32.06	44235	26.32	19537
taxonomy	5 728 398	14 922 125	0.72	1175	1.15	2250	3.84	1507	3.56	1776

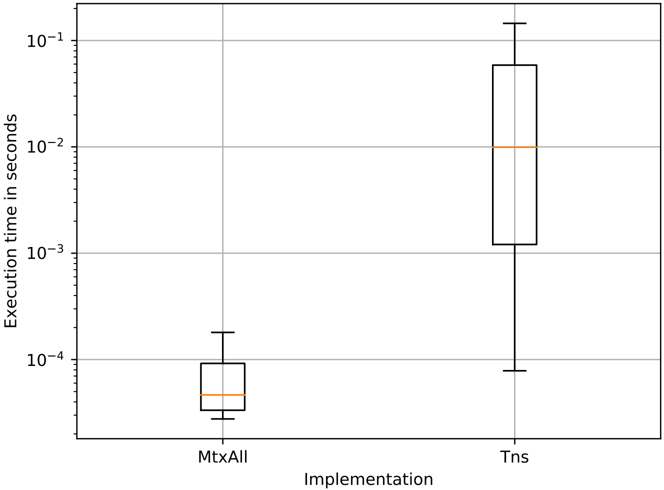
[1] Time in seconds and memory is measured in megabytes

Evaluation: CFPQ[1]

Graph	#V	#E	MtxRel		MtxSingle		MtxAll		Tns	
			Time	Mem	Time	Mem	Time	Mem	Time	Mem
pathways	6 238	18 598	0.01	140	0.01	671	0.01	49	0.01	122
go-hierarchy	45 007	980 218	0.09	255	0.84	671	0.35	195	0.24	252
enzyme	48 815	109 695	0.01	181	0.01	217	0.02	61	0.02	132
eclass_514en	239 111	523 727	0.06	181	0.16	216	0.22	126	0.27	193
go	272 770	534 311	0.94	246	0.93	217	1.13	990	1.27	243
geospecies	450 609	2 311 461	7.48	7645	15.54	22941	32.06	44235	26.32	19537
taxonomy	5 728 398	14 922 125	0.72	1175	1.15	2250	3.84	1507	3.56	1776

[1] Time in seconds and memory is measured in megabytes

Evaluation: Average Path Extraction Time For *go*



Preliminary Results

- The single-path matrix-based algorithm constructs matrices up to 2 times slower than the relational matrix-based algorithm
- The all-path matrix-based algorithm constructs matrices up to 2-3 times slower than the single-path matrix-based algorithm

Preliminary Results

- The single-path matrix-based algorithm constructs matrices up to 2 times slower than the relational matrix-based algorithm
- The all-path matrix-based algorithm constructs matrices up to 2-3 times slower than the single-path matrix-based algorithm
- If it is necessary to frequently recalculate the matrices for a changing graph or a path query then the best choice is the Kronecker product-based algorithm with faster and less memory consuming matrices construction

Preliminary Results

- The single-path matrix-based algorithm constructs matrices up to 2 times slower than the relational matrix-based algorithm
- The all-path matrix-based algorithm constructs matrices up to 2-3 times slower than the single-path matrix-based algorithm
- If it is necessary to frequently recalculate the matrices for a changing graph or a path query then the best choice is the Kronecker product-based algorithm with faster and less memory consuming matrices construction
- If it is necessary to extract paths many times for a once constructed matrices or matrix changes can be efficiently computed dynamically then the proposed matrix-based CFPQ algorithm is preferable

Conclusion

- We provide an approach to solving the CFPQ problem using LA methods that allows one to get interesting practical and theoretical results
- Using provided approach we devise the CFPQ algorithms for all three query semantics based on such LA operations as matrix multiplication and Kronecker product

Conclusion

- We provide an approach to solving the CFPQ problem using LA methods that allows one to get interesting practical and theoretical results
- Using provided approach we devise the CFPQ algorithms for all three query semantics based on such LA operations as matrix multiplication and Kronecker product
- We provide the efficient implementations of the devised algorithms using the implementations of the GraphBLAS API with parallel computations and sparse matrix representations

Conclusion

- We provide an approach to solving the CFPQ problem using LA methods that allows one to get interesting practical and theoretical results
- Using provided approach we devise the CFPQ algorithms for all three query semantics based on such LA operations as matrix multiplication and Kronecker product
- We provide the efficient implementations of the devised algorithms using the implementations of the GraphBLAS API with parallel computations and sparse matrix representations
- We can conclude that the devised LA-based CFPQ solutions are applicable for real-world graph analysis

Future Research

- As a next step, we plan to provide a full comparison of our LA-based implementations with all state-of-the-art CFPQ solutions on the same benchmark and experimental setup
- We compare the CPU-based implementation. In the future, we want to obtain GPU-based and distributed implementations for all devised algorithms
- Also, we plan to provide the multiple-source modifications for all LA-based CFPQ algorithms

Contact Information

- Rustam Azimov:
  - rustam.azimov19021995@gmail.com
  - Rustam.Azimov@jetbrains.com
- Semyon Grigorev:
  - s.v.grigoriev@spbu.ru
  - Semen.Grigorev@jetbrains.com

- Dataset: https://github.com/JetBrains-Research/CFPQ_Data
- Algorithm implementations: https://github.com/JetBrains-Research/CFPQ_PyAlgo

Thanks!