GRADES-NDA 2021

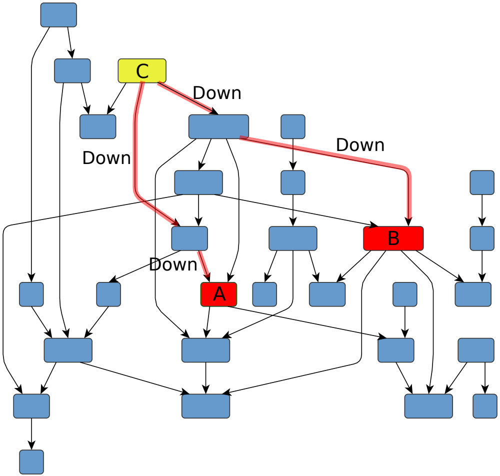
Context-Free Path Querying with All-Path Semantics by Matrix Multiplication

Rustam Azimov, Ilya Epelbaum, Semyon Grigorev

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

June 20, 2021

Context-Free Path Querying



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:
  $SameLvl \to \overline{Down}\ SameLvl\ Down \mid \varepsilon$

Context-Free Path Querying: Relational Query Semantics

- $\mathbb{G} = (\Sigma, N, P)$ — context-free grammar in normal form
  - $A \to BC$, where $A, B, C \in N$
  - $A \to x$, where $A \in N, x \in \Sigma \cup \{\varepsilon\}$
  - $L(\mathbb{G}, A) = \{\omega \mid A \Rightarrow^* \omega\}$

Context-Free Path Querying: Relational Query Semantics

- $\mathbb{G} = (\Sigma, N, P)$ — context-free grammar in normal form
  - $A \to BC$, where $A, B, C \in N$
  - $A \to x$, where $A \in N, x \in \Sigma \cup \{\varepsilon\}$
  - $L(\mathbb{G}, A) = \{\omega \mid A \Rightarrow^* \omega\}$
- $G = (V, E, L)$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$

Context-Free Path Querying: Relational Query Semantics

- $\mathbb{G} = (\Sigma, N, P)$ — context-free grammar in normal form
  - $A \to BC$, where $A, B, C \in N$
  - $A \to x$, where $A \in N, x \in \Sigma \cup \{\varepsilon\}$
  - $L(\mathbb{G}, A) = \{\omega \mid A \Rightarrow^* \omega\}$
- $G = (V, E, L)$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$

Context-Free Path Querying: Relational Query Semantics

- $\mathbb{G} = (\Sigma, N, P)$ — context-free grammar in normal form
  - $A \to BC$, where $A, B, C \in N$
  - $A \to x$, where $A \in N, x \in \Sigma \cup \{\varepsilon\}$
  - $L(\mathbb{G}, A) = \{\omega \mid A \Rightarrow^* \omega\}$
- $G = (V, E, L)$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$
- $R_A = \{(n, m) \mid \exists n \pi m$, such that $\omega(\pi) \in L(\mathbb{G}, A)\}$

Matrix-Based Algorithm: Relational Query Semantics

Algorithm Context-free path querying algorithm

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

Context-Free Path Querying: Single-Path Query Semantics

- For all $A \in N$, for all $(n, m) \in R_A$ also return some path $n\pi m$ such that $\omega(\pi) \in L(\mathbb{G}, A)$
  - usually the shortest path is returned
  - returned path can be used as a proof of existence

Context-Free Path Querying: Single-Path Query Semantics

- For all $A \in N$, for all $(n, m) \in R_A$ also return some path $n\pi m$ such that $\omega(\pi) \in L(\mathbb{G}, A)$
  - usually the shortest path is returned
  - returned path can be used as a proof of existence
- The main idea for the matrix-based algorithm is to store additional information in adjacency matrices to be able to restore one such path $n\pi m$ for all $(n, m) \in R_A$
  - the intermediate vertex
  - some additional information about path such as length

Context-Free Path Querying: All-Path Query Semantics

- For all $A \in N$, for all $(n, m) \in R_A$ also return **all** paths $n\pi m$ such that $\omega(\pi) \in L(\mathbb{G}, A)$
  - in some cases we want to find all data dependencies, for example in static code analysis when searching for vulnerabilities
  - the number of such paths can be infinite if the input graph has cycles

Context-Free Path Querying: All-Path Query Semantics

- For all $A \in N$, for all $(n, m) \in R_A$ also return **all** paths $n \pi m$ such that $\omega(\pi) \in L(\mathbb{G}, A)$
  - in some cases we want to find all data dependencies, for example in static code analysis when searching for vulnerabilities
  - the number of such paths can be infinite if the input graph has cycles
- Currently, our matrix-based algorithms cannot handle the all-path query semantics
- The only linear algebra-based algorithm that solves this problem is the Kronecker product-based CFPQ algorithm

Research Questions

- Can we extend the matrix-based CFPQ algorithm to all-path query semantics?
- What the cost of such extension?
- How does the matrix-based solution for the all-path query semantics compare to the Kronecker product-based?

All-Path Index

- We store the additional information about the paths found as the sets of the intermediate vertices
- We introduce the following matrix multiplication operation
- $T^A \odot T^B = T^C$ where $T^C_{i,j} = \bigcup_{k=1}^{n}(T^A_{i,k} \otimes T^B_{k,j})$ and

$$T^A_{i,k} \otimes T^B_{k,j} = \begin{cases} \{k\}, & \text{if } T^A_{i,k} \neq \emptyset \wedge T^B_{k,j} \neq \emptyset \\ \emptyset, & \text{otherwise} \end{cases}$$

Path extraction

- After constructing a set of matrices with sets of intermediate vertices, we can extract all required paths $i\pi j$ for every vertex pair i, j if such paths exist
- It is assumed that the sets of paths are computed lazily, to ensure the termination in case of an infinite number of paths

Implementation

- 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

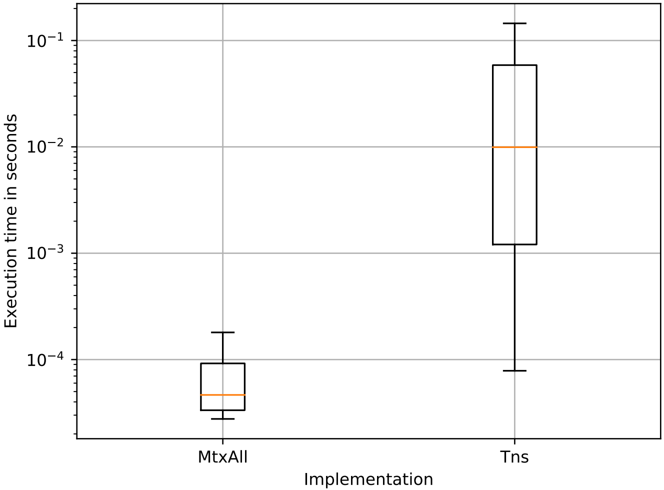
[1]Time in seconds and memory is measured in megabytes

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*



Evaluation Results

- The proposed algorithm constructs index up to 2-3 times slower and consumes more memory than the algorithm for single-path query semantics

Evaluation Results

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

Evaluation Results

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

Conclusion

- The matrix-based CFPQ algorithm can be extended for all-path query semantics

Conclusion

- The matrix-based CFPQ algorithm can be extended for all-path query semantics
- The cost of such extension is an increase in the index creation time and a large increase in memory consumption for some graphs and queries with complex structure of result

Conclusion

- The matrix-based CFPQ algorithm can be extended for all-path query semantics
- The cost of such extension is an increase in the index creation time and a large increase in memory consumption for some graphs and queries with complex structure of result
- The proposed matrix-based solution for all-path query semantics compared to the Kronecker product-based solution consumes more memory, but allows one to extract paths significantly faster

Future Research

- We compare the CPU-based implementation. In the future, we want to obtain GPU-based and distributed implementations
- Also, further improvements in index creation and path extraction for both matrix-based and Kronecker product-based algorithms are required
- We plan to provide the multiple-source modifications for all linear algebra-based CFPQ algorithms

Contact Information

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

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

Thanks!