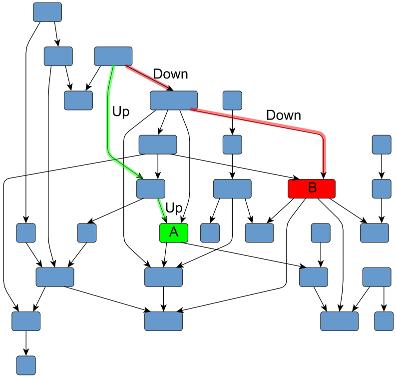
ADBIS 2020

Context-Free Path Querying by Kronecker Product

Egor Orachev, Ilya Epelbaum,
Semyon Grigorev, **Rustam Azimov**

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

August 26, 2020



Navigation through a graph

- Are nodes A and B on the same level of hierarchy?
- Is there a path of form **Up**^{n} **Down**^{n}?
- Find all paths of form **Up**^{n} **Down**^{n} which start from the node A

CFPQ: 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 $A \in N, x \in \Sigma \cup \{\varepsilon\}$
  - $L(\mathbb{G}, A) = \{\omega \mid A \Rightarrow^* \omega\}$

CFPQ: 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
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$

CFPQ: 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
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$

CFPQ: 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 $\omega(\pi) \in L(\mathbb{G}, A)\}$

CFPQ: Existing solutions

- Solutions based on different parsing techniques (CYK, LL, LR, etc.)

CFPQ: Existing solutions

- Solutions based on different parsing techniques (CYK, LL, LR, etc.)
- Matrix-based solutions

CFPQ: Existing solutions

- Solutions based on different parsing techniques (CYK, LL, LR, etc.)
- Matrix-based solutions
- All existing solutions work only with context-free grammar in normal form (CNF, BNF)

CFPQ: Existing solutions

- Solutions based on different parsing techniques (CYK, LL, LR, etc.)
- Matrix-based solutions
- All existing solutions work only with context-free grammar in normal form (CNF, BNF)
- The transformation takes time and can lead to a significant grammar size increase

Recursive State Machines (RSM)

- RSM behaves as a set of finite state machines (FSM) with additional recursive calls
- Any CFG can be easily encoded by an RSM with one box per nonterminal

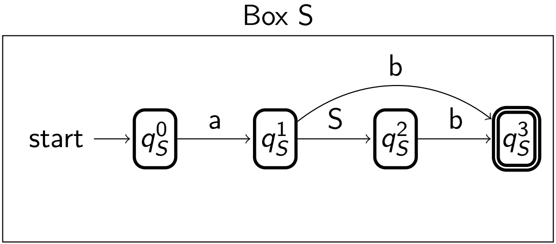
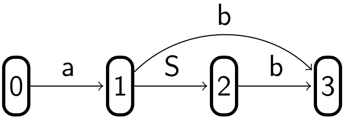
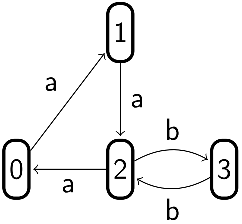


Figure: The RSM for grammar with rules $S \to aSb \mid ab$

CFPQ Algorithm Iteration 1

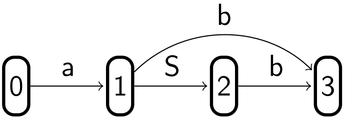


$\otimes$

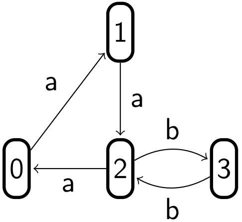


$=$

CFPQ Algorithm Iteration 1



$\otimes$



$=$

$$0,0 \xrightarrow{a} 1,1$$

$$0,\underline{\mathbf{1}} \xrightarrow{\mathbf{a}} 1,2 \xrightarrow{\mathbf{b}} 3,\underline{\mathbf{3}}$$

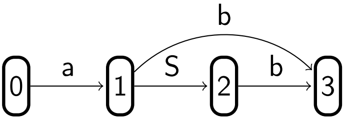
$$0,2 \xrightarrow{a} 1,0$$

$$2,2 \xrightarrow{b} 3,3$$

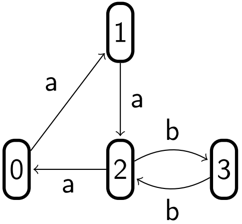
$$2,3 \xrightarrow{b} 3,2$$

$$1,3 \xrightarrow{b} 3,2$$

CFPQ Algorithm Iteration 1



$\otimes$



$=$

$0,0 \xrightarrow{a} 1,1$

$0,\underline{\mathbf{1}} \xrightarrow{\mathbf{a}} 1,2 \xrightarrow{\mathbf{b}} 3,\underline{\mathbf{3}}$

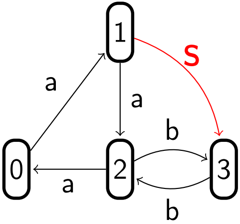
$0,2 \xrightarrow{a} 1,0$

$2,2 \xrightarrow{b} 3,3$

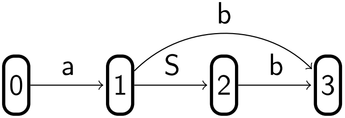
$2,3 \xrightarrow{b} 3,2$

$1,3 \xrightarrow{b} 3,2$

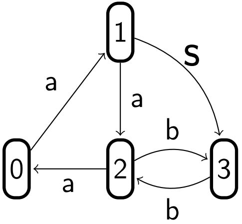
$\longrightarrow$



CFPQ Algorithm Iteration 2

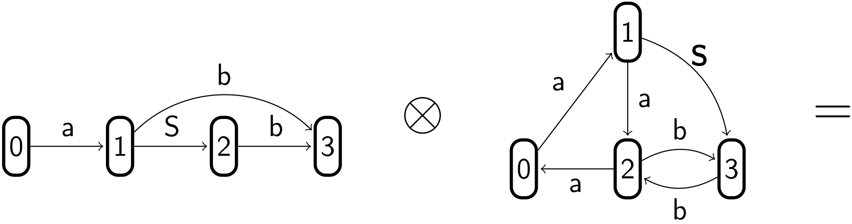


$\otimes$



$=$

CFPQ Algorithm Iteration 2



$$0,\underline{\mathbf{0}} \xrightarrow{\mathbf{a}} 1,1 \xrightarrow{\mathbf{S}} 2,3 \xrightarrow{\mathbf{b}} 3,\underline{\mathbf{2}}$$

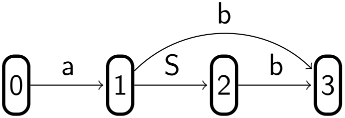
$$0,1 \xrightarrow{a} 1,2 \xrightarrow{b} 3,3$$

$$0,2 \xrightarrow{a} 1,0$$

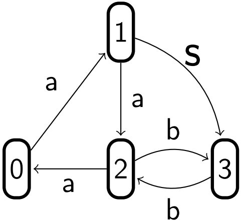
$$2,2 \xrightarrow{b} 3,3$$

$$1,3 \xrightarrow{b} 3,2$$

CFPQ Algorithm Iteration 2



$\otimes$



$=$

$$0, \underline{\mathbf{0}} \xrightarrow{\mathbf{a}} 1, 1 \xrightarrow{\mathbf{S}} 2, 3 \xrightarrow{\mathbf{b}} 3, \underline{\mathbf{2}}$$

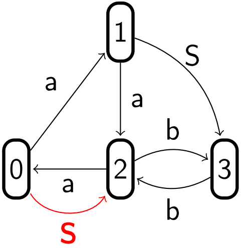
$$0, 1 \xrightarrow{a} 1, 2 \xrightarrow{b} 3, 3$$

$$0, 2 \xrightarrow{a} 1, 0$$

$$2, 2 \xrightarrow{b} 3, 3$$

$$1, 3 \xrightarrow{b} 3, 2$$

$\longrightarrow$



Automaton intersection is a **Kronecker product** of adjacency matrices for $\mathcal{G}$ and $\mathcal{G}_{RSM}$

$$\begin{pmatrix} . & \{a\} & . & . \\ . & . & \{S\} & \{b\} \\ . & . & . & \{b\} \\ . & . & . & . \end{pmatrix} \otimes \begin{pmatrix} . & \{a\} & . & . \\ . & . & \{a\} & . \\ \{a\} & . & . & \{b\} \\ . & . & \{b\} & . \end{pmatrix} =$$

	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(1,3)	(2,0)	(2,1)	(2,2)	(2,3)	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	.	.	.	.	.	$\{a\}$	.	.	.	.	.	.	.	.	.	.
(0,1)	.	.	.	.	.	.	**{a}**	.	.	.	.	.	.	.	.	.
(0,2)	.	.	.	.	$\{a\}$	.	.	.	.	.	.	.	.	.	.	.
(0,3)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(1,0)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(1,1)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(1,2)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	**{b}**
(1,3)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	$\{b\}$	.
(2,0)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(2,1)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(2,2)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	$\{b\}$
(2,3)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	$\{b\}$	.
(2,0)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(2,1)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(2,2)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(2,3)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Implementations

- **Kron** — implementation of the proposed algorithm using **SuiteSparse** C implementation of **GraphBLAS** API, which provides a set of sparse matrix operations

Implementations

- **Kron** — implementation of the proposed algorithm using **SuiteSparse** C implementation of **GraphBLAS** API, which provides a set of sparse matrix operations
- We compare our implementation with **Orig** — the best CPU implementation of the original matrix-based algorithm using M4RI library

Evaluation

- OS: Ubuntu 18.04
- CPU: Intel(R) Core(TM) i7-4790 CPU 3.60GHz
- RAM: DDR4 32 Gb

Evaluation results[1] [2]

	Graph	#V	#E	*Kron*	*Orig*
RDF	generations	129	351	0.04	0.03
RDF	travel	131	397	0.05	0.05
RDF	skos	144	323	0.02	0.04
RDF	unv-bnch	179	413	0.05	0.04
RDF	foaf	256	815	0.07	0.02
RDF	atm-prim	291	685	0.24	0.02
RDF	ppl_pets	337	834	0.18	0.03
RDF	biomed	341	711	0.24	0.05
RDF	pizza	671	2604	1.14	0.08
RDF	wine	733	2450	1.71	0.06
RDF	funding	778	1480	0.43	0.07

	Graph	#V	#E	*Kron*	*Orig*
RDF	core	1323	8684	0.28	0.12
RDF	pways	6238	37196	4.88	0.18
Worst case	WC_1	64	65	0.03	0.04
Worst case	WC_2	128	129	0.16	0.23
Worst case	WC_3	256	257	0.96	1.99
Worst case	WC_4	512	513	7.14	23.21
Worst case	WC_5	1024	1025	121.99	528.52
Full	F_1	100	100	0.17	0.02
Full	F_2	200	200	1.04	0.03
Full	F_3	500	500	18.86	0.03
Full	F_4	1000	1000	554.22	0.07

[1] Queries are based on the context-free grammars for nested parentheses

[2] Time is measured in seconds

Evaluation results[1] [2]

	Graph	#V	#E	*Kron*	*Orig*
RDF	generations	129	351	0.04	0.03
	travel	131	397	0.05	0.05
	skos	144	323	0.02	0.04
	unv-bnch	179	413	0.05	0.04
	foaf	256	815	0.07	0.02
	atm-prim	291	685	0.24	0.02
	ppl_pets	337	834	0.18	0.03
	biomed	341	711	0.24	0.05
	pizza	671	2604	1.14	0.08
	wine	733	2450	1.71	0.06
	funding	778	1480	0.43	0.07

	Graph	#V	#E	*Kron*	*Orig*
RDF	core	1323	8684	0.28	0.12
	pways	6238	37196	4.88	0.18
Worst case	WC_1	64	65	0.03	0.04
	WC_2	128	129	0.16	0.23
	WC_3	256	257	0.96	1.99
	WC_4	512	513	7.14	23.21
	WC_5	1024	1025	121.99	528.52
Full	F_1	100	100	0.17	0.02
	F_2	200	200	1.04	0.03
	F_3	500	500	18.86	0.03
	F_4	1000	1000	554.22	0.07

[1] Queries are based on the context-free grammars for nested parentheses

[2] Time is measured in seconds

Evaluation results[1] [2]

	Graph	#V	#E	*Kron*	*Orig*		Graph	#V	#E	*Kron*	*Orig*
RDF	generations	129	351	0.04	0.03	RDF	core	1323	8684	0.28	0.12
	travel	131	397	0.05	0.05		pways	6238	37196	4.88	0.18
	skos	144	323	0.02	0.04	Worst case	WC_1	64	65	0.03	0.04
	unv-bnch	179	413	0.05	0.04		WC_2	128	129	0.16	0.23
	foaf	256	815	0.07	0.02		WC_3	256	257	0.96	1.99
	atm-prim	291	685	0.24	0.02		WC_4	512	513	7.14	23.21
	ppl_pets	337	834	0.18	0.03		WC_5	1024	1025	121.99	528.52
	biomed	341	711	0.24	0.05	Full	F_1	100	100	0.17	0.02
	pizza	671	2604	1.14	0.08		F_2	200	200	1.04	0.03
	wine	733	2450	1.71	0.06		F_3	500	500	18.86	0.03
	funding	778	1480	0.43	0.07		F_4	1000	1000	554.22	0.07

[1] Queries are based on the context-free grammars for nested parentheses

[2] Time is measured in seconds

Evaluation results[1] [2]

	Graph	#V	#E	*Kron*	*Orig*		Graph	#V	#E	*Kron*	*Orig*
RDF	generations	129	351	0.04	0.03	RDF	core	1323	8684	0.28	0.12
	travel	131	397	0.05	0.05		pways	6238	37196	4.88	0.18
	skos	144	323	0.02	0.04	Worst case	WC_1	64	65	0.03	0.04
	unv-bnch	179	413	0.05	0.04		WC_2	128	129	0.16	0.23
	foaf	256	815	0.07	0.02		WC_3	256	257	0.96	1.99
	atm-prim	291	685	0.24	0.02		WC_4	512	513	7.14	23.21
	ppl_pets	337	834	0.18	0.03		WC_5	1024	1025	121.99	528.52
	biomed	341	711	0.24	0.05	Full	F_1	100	100	0.17	0.02
	pizza	671	2604	1.14	0.08		F_2	200	200	1.04	0.03
	wine	733	2450	1.71	0.06		F_3	500	500	18.86	0.03
	funding	778	1480	0.43	0.07		F_4	1000	1000	554.22	0.07

[1] Queries are based on the context-free grammars for nested parentheses

[2] Time is measured in seconds

Conclusion

- We show that the linear algebra based CFPQ can be done without grammar transformation

Conclusion

- We show that the linear algebra based CFPQ can be done without grammar transformation
- The Kronecker product can be used as the main matrix operation in such algorithm

Conclusion

- We show that the linear algebra based CFPQ can be done without grammar transformation
- The Kronecker product can be used as the main matrix operation in such algorithm
- We show that in some cases our algorithm outperforms the original matrix-based algorithm

Future Research

- Improve our implementation to make it applicable for real-world graphs analysis

Future Research

- Improve our implementation to make it applicable for real-world graphs analysis
- Analyze how the behavior depends on the query type and its form
  - Analyze regular path queries evaluation and context-free path queries in the form of extended context-free grammars (ECFG)

Future Research

- Improve our implementation to make it applicable for real-world graphs analysis
- Analyze how the behavior depends on the query type and its form
  - Analyze regular path queries evaluation and context-free path queries in the form of extended context-free grammars (ECFG)
- Compare our algorithm with the matrix-based one in cases when the size difference between Chomsky Normal Form and ECFG representation of the query is significant

Future Research

- Improve our implementation to make it applicable for real-world graphs analysis
- Analyze how the behavior depends on the query type and its form
  - Analyze regular path queries evaluation and context-free path queries in the form of extended context-free grammars (ECFG)
- Compare our algorithm with the matrix-based one in cases when the size difference between Chomsky Normal Form and ECFG representation of the query is significant
- Extend our algorithm to single-path and all-path query semantics

Contact Information

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

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

Thanks!