

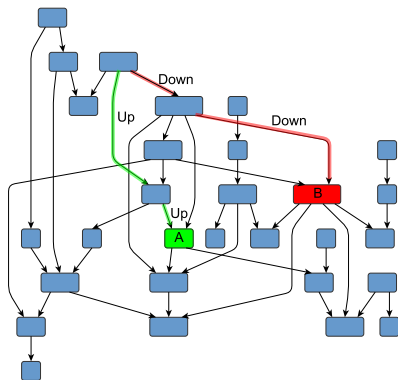
Context-Free Path Querying by Kronecker Product

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

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

August 26, 2020

Context-Free Path Querying



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

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

CFPQ: Query Semantics

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

CFPQ: Query Semantics

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

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 works only with context-free grammar in normal form

CFPQ: Existing solutions

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

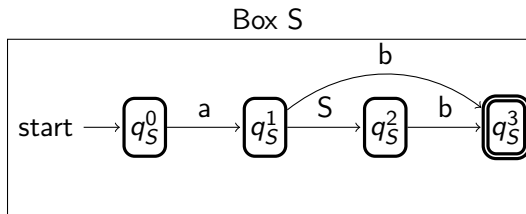
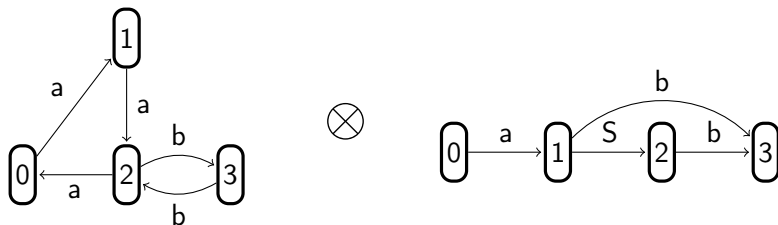


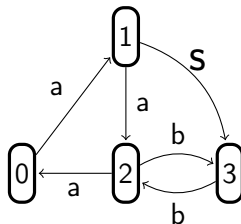
Figure: The RSM for grammar with rules $S \rightarrow aSb \mid ab$

CFPQ Algorithm Iteration 1

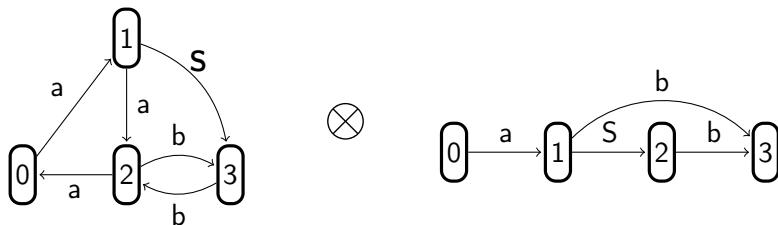


CFPQ Algorithm Iteration 1: Product Automaton

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



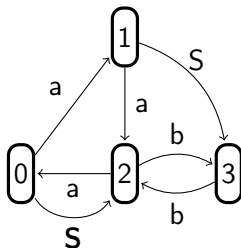
CFPQ Algorithm Iteration 2



CFPQ Algorithm Iteration 2: Product Automaton

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

\longrightarrow



CFPQ Algorithm: Kronecker Product

- We repeat these iterations while input graph \mathcal{G} is changing
- Constructing of the product automaton can be done using the **Kronecker product** of adjacency matrices for \mathcal{G} and \mathcal{G}_{RSM}
- We can use the sparse and block nature of the obtained matrices to apply wide class of optimizations

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

- **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 implementations of the original matrix-based algorithm using M4RI library

- 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		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

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

²Time 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

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

²Time 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

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

²Time 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

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

²Time is measured in seconds

Conclusion

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

Conclusion

- We show that the matrix-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 matrix-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

Conclusion

- We show that the matrix-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
- We still can use existing high-performance libraries for matrix operations

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

- 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)

- 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

- 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!