ADBIS 2020

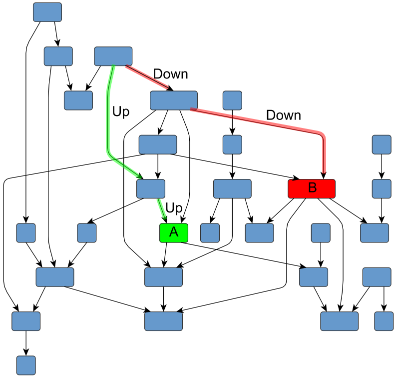
Context-Free Path Querying by Kronecker Product

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

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

August 26, 2020

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 $\textbf{Up}^n\,\textbf{Down}^n$?
- Find all paths of form $\textbf{Up}^n\,\textbf{Down}^n$ 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: Original Matrix-Based Algorithm

Algorithm Context-free path querying algorithm

function $\textsc{evalCFPQ}(D = (V, E, L), G = (\Sigma, N, P))$
    $n \leftarrow |V|$
    $T \leftarrow \{T^{A_i} \mid A_i \in N, T^{A_i}$ is a matrix $n \times n$, $T^{A_i}_{k,l} \leftarrow$ `false`$\}$
    for all $(i, x, j) \in E$, $A_k \mid A_k \to x \in P$ **do** $T^{A_k}_{i,j} \leftarrow$ `true`
    for all $A_k \mid A_k \to \varepsilon \in P$ **do**
        for all $i \in \{0, \ldots, n-1\}$ **do** $T^{A_k}_{i,i} \leftarrow$ `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

CFPQ: Grammar Transformation

- $\mathbb{G} = (\Sigma, N, P)$ — context-free grammar in general form
  - $A \to \alpha$, where $A \in N, \alpha \in (N \cup \Sigma \cup \{\varepsilon\})^*$

CFPQ: Grammar Transformation

- $\mathbb{G} = (\Sigma, N, P)$ — context-free grammar in general form
  - $A \to \alpha$, where $A \in N, \alpha \in (N \cup \Sigma \cup \{\varepsilon\})^*$
- Every context-free grammar can be transformed to binary normal form

CFPQ: Grammar Transformation

- $\mathbb{G} = (\Sigma, N, P)$ — context-free grammar in general form
  - $A \to \alpha$, where $A \in N, \alpha \in (N \cup \Sigma \cup \{\varepsilon\})^*$
- Every context-free grammar can be transformed to binary normal form
- The transformation takes time and can lead to a significant grammar size increase

Research Questions

- Can we create the matrix-based CFPQ algorithm that does not require grammar transformation?
- What matrix operations should be used?
- Is the obtained algorithm comparable with the original matrix-based algorithm?
- Can we still use existing high-performance libraries for matrix operations?

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

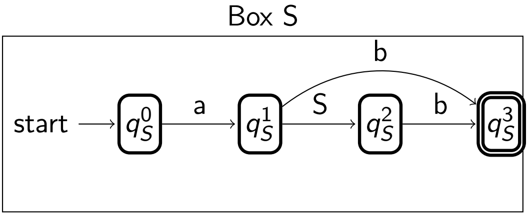
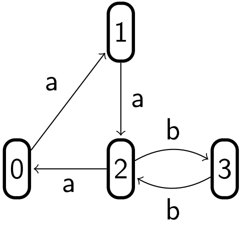
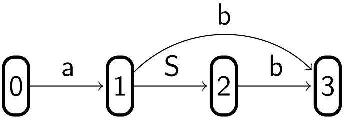


Figure: The RSM for grammar with rules $S \to aSb \mid ab$

CFPQ Algorithm Iteration



(a) The input graph $\mathcal{G}$



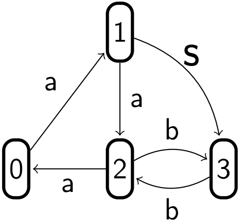
(b) The input RSM graph $\mathcal{G}_{RSM}$

- We need to intersect these two graphs by constructing the product automaton

CFPQ Algorithm Iteration

$$
\begin{array}{lllll}
0,0 & \xrightarrow{a} & 1,1 & & \\
\underline{\mathbf{1}},0 & \xrightarrow{\mathbf{a}} & 2,1 & \xrightarrow{\mathbf{b}} & \underline{\mathbf{3}},3 \\
2,0 & \xrightarrow{a} & 0,1 & & \\
2,2 & \xrightarrow{b} & 3,3 & & \\
3,2 & \xrightarrow{b} & 2,3 & & \\
3,1 & \xrightarrow{b} & 2,3 & &
\end{array}
$$

(a) Constructing the product automaton



(b) The updated input graph $\mathcal{G}$ using rule $S \to ab$

CFPQ Algorithm: Kronecker Product

- We repeat this iteration 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
- We still can use existing high-performance math libraries if they provide satisfying operations

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 implementations of the original matrix-based algorithm using M4RI library with sparse matrix representation

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*		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

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*
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

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

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

- Dataset is published: both graphs and queries
  - Link: `https://github.com/JetBrains-Research/CFPQ_Data`
- Implementations are available on GitHub
  - Link: `https://github.com/YaccConstructor/RedisGraph`

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!