GRADES-NDA 2020

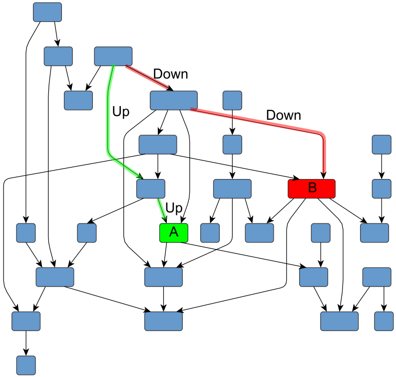
Context-Free Path Querying with Single-Path Semantics by Matrix Multiplication

Arseniy Terekhov, Artyom Khoroshev,
Semyon Grigorev, **Rustam Azimov**

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

June 14, 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

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 $\textsc{evalCFPQ}(D = (V, E, L), G = (\Sigma, N, P))$
    $n \leftarrow |V|$
    $T \leftarrow \{T^{A_i} \mid A_i \in N, T^{A_i}$ is a matrix $n \times n$, $T^{A_i}_{k,l} \leftarrow \texttt{false}\}$
    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

Context-Free Path Querying: Single-Path Query Semantics

- $R_A = \{(n, m) \mid \exists n\pi m$, such that $\omega(\pi) \in L(\mathbb{G}, A)\}$ — answers for the relational query semantics

Context-Free Path Querying: Single-Path Query Semantics

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

Research Questions

- Can we extend the matrix-based CFPQ algorithm to single-path query semantics?
- What the cost of such extension?
- Can we achieve high performance of CFPQ integrated with existing graph database?
- Does using GPGPU still improve performance over CPU versions?

Path Index

- $PathIndex = (left, right, middle, height, length)$

Path Index

- $PathIndex = (left, right, middle, height, length)$
- $\perp = (0, 0, 0, 0, 0)$

Path Index

- $PathIndex = (left, right, middle, height, length)$
- $\perp = (0, 0, 0, 0, 0)$

$$PI_1 \otimes PI_2 = (PI_1.left, PI_2.right, PI_1.right, max(PI_1.height, PI_2.height) + 1,\\ PI_1.length + PI_2.length).$$

Path Index

- $PathIndex = (left, right, middle, height, length)$
- $\perp = (0, 0, 0, 0, 0)$

$$PI_1 \otimes PI_2 = (PI_1.left, PI_2.right, PI_1.right, max(PI_1.height, PI_2.height) + 1, PI_1.length + PI_2.length).$$

$$PI_1 \oplus PI_2 = \begin{cases} PI_1, & \text{if } PI_1.height \leq PI_2.height \\ PI_2, & \text{otherwise} \end{cases}$$

Matrix-Based Algorithm: Single-Path Query Semantics

Algorithm CFPQ algorithm w.r.t. single-path query semantics

function $\text{EVALCFPQ}(D = (V, E), G = (N, \Sigma, P))$

  $n \leftarrow |V|$

  $T \leftarrow \{ T^{A_i} \mid A_i \in N, T^{A_i}$ is a matrix $n \times n$, $T^{A_i}_{k,l} \leftarrow \bot \}$

  for all $(i, x, j) \in E$, $A_k \mid A_k \to x \in P$ **do** $T^{A_k}_{i,j} \leftarrow (i, j, i, 1, 1)$

  for $A_k \mid A_k \to \varepsilon \in P$ **do** $T^{A_k}_{i,i} \leftarrow (i, i, i, 1, 0)$

  while any matrix in T is changing **do**

    for $A_i \to A_j A_k \in P$ **do** $T^{A_i} \leftarrow T^{A_i} + (T^{A_j} \odot T^{A_k})$

  return T

Matrix-Based Algorithm: Technical Details

- We can remove *length* or *height* to reduce memory consumption
- The PathIndex operations can be represented as bitwise atomic operations
- We still can use existing high-performance libraries for matrix operations if they support the creation of custom operations

Path extraction

- After constructing a set of matrices with PathIndexes, we can extract the required path $i\pi j$ for every node pair i, j and non-terminal A if such path exists

Path extraction

- After constructing a set of matrices with PathIndexes, we can extract the required path $i\pi j$ for every node pair i, j and non-terminal A if such path exists
  - The path which forms a string with minimal height of derivation tree

Path extraction

- After constructing a set of matrices with PathIndexes, we can extract the required path $i\pi j$ for every node pair i, j and non-terminal A if such path exists
  - The path which forms a string with minimal height of derivation tree
  - The shortest path

Path extraction

- After constructing a set of matrices with PathIndexes, we can extract the required path $i\pi j$ for every node pair i, j and non-terminal A if such path exists
  - The path which forms a string with minimal height of derivation tree
  - The shortest path
- Linear complexity in the length of the extracted path

Implementations

- We use **RedisGraph** graph database as storage

Implementations

- We use **RedisGraph** graph database as storage
- **CPU-based** implementations use **SuiteSparse** implementation of **GraphBLAS**, which provides a set of sparse matrix operations
  - **RG_CPU**$_{rel}$ — for the relation query semantics

Implementations

- We use **RedisGraph** graph database as storage
- **CPU-based** implementations use **SuiteSparse** implementation of **GraphBLAS**, which provides a set of sparse matrix operations
  - **RG_CPU**$_{rel}$ — for the relation query semantics
  - **RG_CPU**$_{path}$ — for the single-path query semantics

Implementations

- We use **RedisGraph** graph database as storage
- **CPU-based** implementations use **SuiteSparse** implementation of **GraphBLAS**, which provides a set of sparse matrix operations
  - $\mathbf{RG\_CPU}_{rel}$ — for the relation query semantics
  - $\mathbf{RG\_CPU}_{path}$ — for the single-path query semantics
- **GPGPU-based** implementations with sparse matrix representation
  - $\mathbf{RG\_CUSP}_{rel}$ — relational query semantics, utilizes a **CUSP** library for matrix operations

Implementations

- We use **RedisGraph** graph database as storage
- **CPU-based** implementations use **SuiteSparse** implementation of **GraphBLAS**, which provides a set of sparse matrix operations
    - **RG_CPU**$_{rel}$ — for the relation query semantics
    - **RG_CPU**$_{path}$ — for the single-path query semantics
- **GPGPU-based** implementations with sparse matrix representation
    - **RG_CUSP**$_{rel}$ — relational query semantics, utilizes a **CUSP** library for matrix operations
    - **RG_SPARSE**$_{rel}$ — relational query semantics, uses low-latency on-chip shared memory for the hash table of each row of the result matrix

Implementations

- We use **RedisGraph** graph database as storage
- **CPU-based** implementations use **SuiteSparse** implementation of **GraphBLAS**, which provides a set of sparse matrix operations
  - **RG_CPU**$_{rel}$ — for the relation query semantics
  - **RG_CPU**$_{path}$ — for the single-path query semantics
- **GPGPU-based** implementations with sparse matrix representation
  - **RG_CUSP**$_{rel}$ — relational query semantics, utilizes a **CUSP** library for matrix operations
  - **RG_SPARSE**$_{rel}$ — relational query semantics, uses low-latency on-chip shared memory for the hash table of each row of the result matrix
  - **RG_SPARSE**$_{path}$ — single-path query semantics, operating over PathIndex semiring

Dataset[1]

RDF Name	#V	#E
univ-bench	179	413
pizza	671	2,604
wine	733	2,450
core	1,323	8,684
pathways	6,238	37,196
go-hierarchy	45,007	1,960,436
enzyme	48,815	219,390
eclass_514en	239,111	1,047,454
go	272,770	1,068,622
geospecies	450,609	4,622,922

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

Evaluation

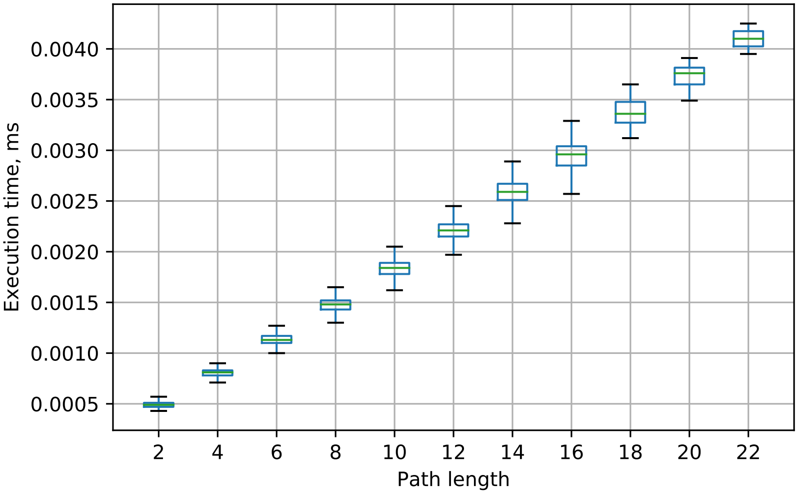
- OS: Ubuntu 18.04
- CPU: Intel core i7 6700 3,4GHz
- RAM: DDR4 64 Gb
- GPGPU: NVIDIA GeForce 1070 (8Gb RAM)

Evaluation: CFPQ[2]

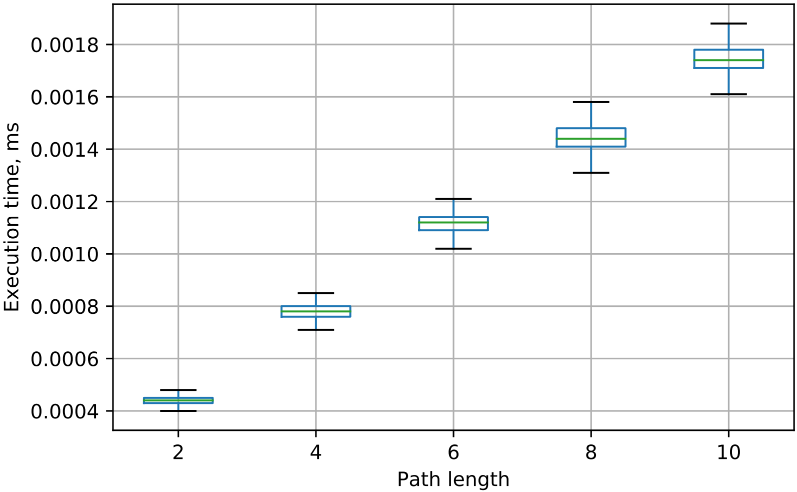
Name	Relational semantics index						Single path semantics index			
	RG_CPU$_{rel}$		RG_CUSP$_{rel}$		RG_SPARSE$_{rel}$		RG_CPU$_{path}$		RG_SPARSE$_{path}$	
	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem
core	0.004	0.3	0.022	2.0	0.010	0.1	0.002	0.3	0.016	0.1
eclass_514en	0.067	13.8	0.075	14.0	0.166	16.0	0.195	31.2	0.496	26.0
enzyme	0.018	5.9	0.021	0.1	0.018	4.0	0.029	8.1	0.043	6.0
go-hierarchy	0.091	16.3	0.433	650.0	0.108	121.2	0.976	92.0	0.336	125.0
go	0.604	28.8	0.590	70.0	0.365	30.2	1.286	75.7	0.739	45.4
pathways	0.011	0.1	0.019	0.1	0.007	0.1	0.021	0.5	0.021	2.0
univ-bench	0.002	0.3	0.010	0.1	0.005	0.1	0.013	0.3	0.007	0.1
pizza	0.030	1.8	0.021	4.0	0.006	0.1	0.075	5.5	0.009	0.1
wine	0.017	3.5	0.032	6.0	0.009	0.1	0.117	7.1	0.015	0.2
geospecies	7.146	16934.2	—	—	0.856	5274	15.134	35803.6	1.935	5282

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

Evaluation: Path Extraction Time For *go*



Evaluation: Path Extraction Time For *geospecies*



Conclusion

- GPGPUs utilization significantly increases the performance of CFPQ for both query semantics

Conclusion

- GPGPUs utilization significantly increases the performance of CFPQ for both query semantics
- Implementations with sparse matrix representation are significantly faster than others

Conclusion

- GPGPUs utilization significantly increases the performance of CFPQ for both query semantics
- Implementations with sparse matrix representation are significantly faster than others
- The cost of computing matrices with PathIndexes for single-path query semantics is not high

Conclusion

- GPGPUs utilization significantly increases the performance of CFPQ for both query semantics
- Implementations with sparse matrix representation are significantly faster than others
- The cost of computing matrices with PathIndexes for single-path query semantics is not high
- The additional running time of the path extraction is small and linear in the length of the path

Conclusion

- GPGPUs utilization significantly increases the performance of CFPQ for both query semantics
- Implementations with sparse matrix representation are significantly faster than others
- The cost of computing matrices with PathIndexes for single-path query semantics is not high
- The additional running time of the path extraction is small and linear in the length of the path
- The matrix-based algorithm paired with a suitable database is a promising way to make CFPQ applicable for real-world data analysis

Conclusion

- GPGPUs utilization significantly increases the performance of CFPQ for both query semantics
- Implementations with sparse matrix representation are significantly faster than others
- The cost of computing matrices with PathIndexes for single-path query semantics is not high
- The additional running time of the path extraction is small and linear in the length of the path
- The matrix-based algorithm paired with a suitable database is a promising way to make CFPQ applicable for real-world data analysis

- 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

- Extend the matrix-based CFPQ algorithm to all-path query semantics

Future Research

- Extend the matrix-based CFPQ algorithm to all-path query semantics
- Update the query results dynamically when data changes

Future Research

- Extend the matrix-based CFPQ algorithm to all-path query semantics
- Update the query results dynamically when data changes
- Improve the dataset
  - Include real-world cases from the area of static code analysis
  - Find new applications that required CFPQ, such as graph segmentation

Acknowledgments

- We thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions
- Special thanks to
  - Габору
  - Флетчеру
  - Рою

Contact Information

- Semyon Grigorev:
  - s.v.grigoriev@spbu.ru
  - Semen.Grigorev@jetbrains.com
- Rustam Azimov:
  - rustam.azimov19021995@gmail.com
  - Rustam.Azimov@jetbrains.com
- Arseniy Terekhov: simpletondl@yandex.ru
- Artyom Khoroshev: arthoroshev@gmail.com

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

Thanks!