Evaluating Complex Queries on Streaming Graphs

Anıl Paçacı 1  Angela Bonifati 2  M. Tamer Özsu 1

1 University of Waterloo
2 Lyon 1 University
Streaming Graphs

- Streams are unbounded – no global access
- High streaming rates in real-world applications
Streaming Graphs

Characteristics of Streaming Graphs

- Streams are unbounded – no global access
- High streaming rates in real-world applications
Applications of Streaming Graph Queries

- Fraud detection in e-commerce [Qiu et al., 2018]

(a) Credit-card fraud

(Taken from [Qiu et al., 2018])
Applications of Streaming Graph Queries

- Fraud detection in e-commerce [Qiu et al., 2018]
- Intrusion detection on networks [Kent et al., 2015; Choudhury et al., 2015]

(a) Credit-card fraud
(Taken from [Qiu et al., 2018])

(b) Denial-of-service (DOS) attack
(Taken from [Choudhury et al., 2015])
Applications of Streaming Graph Queries

- Fraud detection in e-commerce [Qiu et al., 2018]
- Intrusion detection on networks [Kent et al., 2015; Choudhury et al., 2015]

A common theme:

▶ Specialized algorithms targeting particular use-case & applications!

(a) Credit-card fraud
(Taken from [Qiu et al., 2018])

(b) Denial-of-service (DOS) attack
(Taken from [Choudhury et al., 2015])
Challenges

Two difficult problems: *streams* + *graphs*

1. These are graph queries:
   - Subgraph patterns
   - Recursive path navigations
   - Returning and manipulating paths

2. Continuously evaluated on fast-changing data:
   - Online evaluation with real-time results
   - Potentially unbounded streams

The objective of this research:
▶ Combine graphs + streams in a principled way
▶ Design of a general-purpose Streaming Graph Query Processor
Challenges

Two difficult problems: *streams* + *graphs*

1. These are graph queries:
   - Subgraph patterns
   - Recursive path navigations
   - Returning and manipulating paths

The objective of this research:
- Combine graphs + streams in a principled way
- Design of a general-purpose Streaming Graph Query Processor
Two difficult problems: *streams + graphs*

1. These are graph queries:
   - Subgraph patterns
   - Recursive path navigations
   - Returning and manipulating paths

2. Continuously evaluated on fast-changing data:
   - Online evaluation with real-time results
   - Potentially unbounded streams
Challenges

Two difficult problems: *streams* + *graphs*

1. These are graph queries:
   - Subgraph patterns
   - Recursive path navigations
   - Returning and manipulating paths

2. Continuously evaluated on fast-changing data:
   - Online evaluation with real-time results
   - Potentially unbounded streams
Challenges

Two difficult problems: *streams* + *graphs*

1. These are graph queries:
   - Subgraph patterns

2. Continuously evaluated on fast-changing data:
   - Online evaluation with real-time results
   - Potentially unbounded streams

---

The objective of this research:

- Combine graphs + streams in a principled way
- Design of a general-purpose *Streaming Graph Query Processor*
The focus of this paper is orthogonal to the contributions of this paper.
The focus of this paper is orthogonal to the contributions of this paper.
End-to-end Query Processor

High level user query → PARSER → Algebra expression (query tree) → PLAN GENERATION → Possible plans

Equivalence rules

Design of an SGQ optimizer is orthogonal to the contributions of this paper.
The focus of this paper is orthogonal to the contributions of this paper.
End-to-end Query Processor

1. **High level user query**

2. **PARSER**
   - Algebra expression (query tree)

3. **PLAN GENERATION**
   - Possible plans
   - Equivalence rules
   - Cost model & statistics

4. **OPTIMIZER**
   - Optimized logical plan
   - Cost model & statistics

5. **CODE GENERATION**
   - Execution plan
   - Operator impl.

The focus of this paper is orthogonal to the contributions of this paper.
The focus of this paper is on the design of an end-to-end query processor. The design of an SGQ optimizer is orthogonal to the contributions of this paper.
The focus of this paper is on the design of an end-to-end query processor, specifically the SGQ optimizer. The paper's contributions are orthogonal to other work in the field. The model includes high-level user queries, a PARSER producing algebra expressions (query trees), PLAN GENERATION with possible plans, and OPTIMIZER with equivalence rules and cost model & statistics to produce an optimized logical plan. CODE GENERATION then generates execution plans, which are executed by the EXECUTION ENGINE. This diagram illustrates the integration of these components into an end-to-end query processing system.
End-to-end Query Processor

The focus of this paper

The focus of this paper is orthogonal to the contributions of this paper.
In this talk

A principled approach for evaluating Streaming Graph Queries
A principled approach for evaluating *Streaming Graph Queries*

1. Precise semantics of *Streaming Graph Queries*
   - Data model & query model
A principled approach for evaluating *Streaming Graph Queries*

1. Precise semantics of *Streaming Graph Queries*
   - Data model & query model

2. An algebraic basis for query evaluation
   - Logical primitives to formulate *Streaming Graph Queries*
   - Space of query execution plans
A principled approach for evaluating *Streaming Graph Queries*

1. Precise semantics of *Streaming Graph Queries*
   - Data model & query model

2. An algebraic basis for query evaluation
   - Logical primitives to formulate *Streaming Graph Queries*
   - Space of query execution plans

3. A prototype implementation
   - Timely Dataflow as the execution engine
   - Incremental & non-blocking operator implementations
Streaming Graph Queries
Data Model

**Streaming graph** as a sequence of *streaming graph tuples*

- \( t = (\text{src}, \text{trg}, l, [\text{ts}, \text{exp}], D) \)
**Data Model**

**Streaming graph** as a sequence of *streaming graph tuples*

- \( t = (src, trg, l, [ts, exp], D) \)
Streaming graph as a sequence of streaming graph tuples

- $t = (src, trg, l, [ts, exp), D)$

Snapshot of a streaming graph

- a mapping $\tau$ from $T$ to finite set of sgts
- Directed, labelled multigraph
**Data Model**

**Streaming graph** as a sequence of *streaming graph tuples*

- \( t = (\text{src}, \text{trg}, l, [ts, \exp], D) \)

**Snapshot** of a streaming graph

- a mapping \( \tau \) from \( T \) to finite set of sgts
- Directed, labelled multigraph

\[ t = 20 \]

\[
\begin{align*}
\text{[7, 31]} & \quad \text{[10, 34]} & \quad \text{[13, 37]} & \quad \text{[17, 41]} & \quad \text{[17, 31]} & \quad \text{[22, 46]} & \quad \text{[28, 52]} & \quad \text{[29, 53]} & \quad \text{[30, 54]}
\end{align*}
\]
Streaming Graph Query example:\textsuperscript{2}

2 adopted from LDBC SNB Interactive Query 7
Streaming Graph Query example:

G-CORE representation

PATH RL = (x)-[:follows]->(y),
(x)-[:likes]->(m1)<-[[:posts]]-(y)
Streaming Graph Query example:

G-CORE representation

PATH RL = (x)-[:follows]->(y),
  (x)-[:likes]->(m1)<-[[:posts]]-(y)
Streaming Graph Query example:

G-CORE representation

PATH RL = (x)-[:follows]->(y),
(x)-[:likes]->(m1)<-[[:posts]]-(y)
**SGQ Model**

**Streaming Graph Query example:**

![Diagram showing a graph with nodes and edges labeled with relations and properties.]

**G-CORE representation**

\[
\text{PATH } RL = (x)-[:follows]->(y), \\
(x)-[:likes]->(m_1)<-[:posts]-(y)
\]

\[
\text{MATCH } (p_1)\sim RL^+ \rightarrow (p_2), \\
(p_2)-[:posts]->(m)
\]
SGQ Model

Streaming Graph Query example:

G-CORE representation

PATH RL = (x)-[:follows]->(y),
(x)-[:likes]->(m1)<-[[:posts]]-(y)

CONSTRUCT (p1) -[:notify]-> (m)

MATCH (p1) -/ ≈RL+ /->(p2),
(p2)-[:posts]->(m)
Streaming Graph Query example:

G-CORE representation

PATH RL = (x)-[:follows]->(y),
     (x)-[:likes]->(m1)<-[:posts]-(y)
CONSTRUCT (p1)-[:notify]-> (m)
MATCH (p1)-/ ~RL+ ->(p2),
     (p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)
A formal model for Streaming Graph Queries

- non-recursive Datalog + Kleene star
A formal model for Streaming Graph Queries

- non-recursive Datalog + Kleene star
- Streaming generalization of RPGQ [Bonifati et al., 2018]
  - Regular Queries [Reutter et al., 2007]
Streaming Graph Query example:

A formal model for Streaming Graph Queries

- non-recursive Datalog + Kleene star
- Streaming generalization of RPGQ [Bonifati et al., 2018]
  - Regular Queries [Reutter et al., 2007]
- Subsumes existing languages (PGQL, SPARQL v1.1, Cypher v9)
The principle of *snapshot reducibility* [Clifford et al., 1994]
The principle of *snapshot reducibility* [Clifford et al., 1994]

![Diagram showing the semantics of streaming queries](image-url)
The principle of *snapshot reducibility* [Clifford et al., 1994]
The principle of *snapshot reducibility* [Clifford et al., 1994]

![Diagram](Image)

---

3 Figures are adopted from [Krämer and Seeger, 2009]
Streaming Graph Algebra
## A Quick Tour of SGA Primitives

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILTER</td>
<td>( \sigma_\Phi(S) )</td>
<td>boolean condition</td>
</tr>
<tr>
<td>UNION</td>
<td>( \bigcup[d](S_1, \ldots, S_n) )</td>
<td></td>
</tr>
<tr>
<td>WSCAN</td>
<td>( \mathcal{W}_{T,\beta}(S) )</td>
<td>window length</td>
</tr>
<tr>
<td>PATTERN</td>
<td>( \Join^{}_{\Phi}^{src, trg, d} (S_1, \ldots, S_n) )</td>
<td>subgraph pattern</td>
</tr>
<tr>
<td>PATH</td>
<td>( \mathcal{P}^d_R(S_1, \ldots, S_n) )</td>
<td>path expression</td>
</tr>
</tbody>
</table>
G-CORE Query:

PATH RL = (x)-[:follows]->(y),
   (x)-[:likes]->(m1)<-[[:posts]]-(y)
CONSTRUCT (p1) -[:notify]-> (m)
MATCH (p1)-/ <~RL+> /->(p2),
   (p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)
G-CORE Query:

PATH RL = (x)-[:follows]->(y),
    (x)-[:likes]->(m1)<-[:posts]-(y)
CONSTRUCT (p1) -[:notify]-> (m)
MATCH (p1)-/ <~RL+> /->(p2),
    (p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)

SGA Expression

\[ S_l = \sigma_{l=\text{likes}}(W^{24}(S)) \]
\[ S_f = \sigma_{l=\text{follows}}(W^{24}(S)) \]
\[ S_p = \sigma_{l=\text{posts}}(W^{24}(S)) \]
G-CORE Query:

PATH RL = (x)-[:follows]->(y),
(x)-[:likes]->(m1)<-[:posts]-(y)
CONSTRUCT (p1) -[:notify]-> (m)
MATCH (p1)-/ ~RL+ /->(p2),
(p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)

SGA Expression

\[ S_l = \sigma_{l=\text{likes}} (W^{24}(S)) \]
\[ S_f = \sigma_{l=\text{follows}} (W^{24}(S)) \]
\[ S_p = \sigma_{l=\text{posts}} (W^{24}(S)) \]
\[ S_{\text{RecentLiker}} = \bigwedge_{\phi}^{src1, src3, RecentLiker} (S_{\text{likes}}, S_{\text{follows}}, S_{\text{posts}}) \]
SGA - An Example

G-CORE Query:

PATH RL = (x)-[:follows]->(y),
     (x)-[:likes]->(m1)<-[[:posts]]-(y)

CONSTRUCT (p1) -[:notify]-> (m)

MATCH (p1)-/ <~RL+> /->(p2),
     (p2)-[:posts]->(m)

ON ldbc_stream WINDOW(24 hours)

SGA Expression

\[
S_l = \sigma_{l=}\text{likes}(W^{24}(S))
\]

\[
S_f = \sigma_{l=}\text{follows}(W^{24}(S))
\]

\[
S_p = \sigma_{l=}\text{posts}(W^{24}(S))
\]

\[
S_{\text{RecentLiker}} = \forall_{\phi}^{src1,src3,\text{RecentLiker}} (S_{\text{likes}}, S_{\text{follows}}, S_{\text{posts}})
\]

\[
S_{\text{Related}} = \mathcal{P}^{\text{Notify}}_{\text{RecentLiker}^+}(S_{\text{RecentLiker}})
\]
SGA - An Example

G-CORE Query:

PATH RL = (x)-[:follows]->(y),
   (x)-[:likes]->(m1)<-[[:posts]]-(y)
CONSTRUCT (p1) -[:notify]-> (m)
MATCH (p1)-/ <~RL+> /->(p2),
   (p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)

SGA Expression

\[ S_l = \sigma_{l=\text{likes}}(W^{24}(S)) \]
\[ S_f = \sigma_{l=\text{follows}}(W^{24}(S)) \]
\[ S_p = \sigma_{l=\text{posts}}(W^{24}(S)) \]
\[ S_{\text{RecentLiker}} = \exists_{\phi}^{\text{src1,src3,RecentLiker}}(S_{\text{likes}}, S_{\text{follows}}, S_{\text{posts}}) \]
\[ S_{\text{Related}} = \exists_{\phi}^{\text{Notify}_{\text{RecentLiker}}}(S_{\text{RecentLiker}}) \]
\[ \text{Answer} = \exists_{\phi}^{\text{src1,trg2,Notify}}(S_{\text{Related}}, S_p) \]
SGA - An Example

G-CORE Query:

PATH RL = (x)-[:follows]->(y),
(x)-[:likes]->(m1)<-[:posts]-(y)
CONSTRUCT (p1) -[:notify]-> (m)
MATCH (p1)-/ <~RL+> /->(p2),
(p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)

SGA Expression

\[ S_l = \sigma_{l=\text{likes}}(W^{24}(S)) \]
\[ S_f = \sigma_{l=\text{follows}}(W^{24}(S)) \]
\[ S_p = \sigma_{l=\text{posts}}(W^{24}(S)) \]
\[ S_{\text{RecentLiker}} = \chi_{\phi}^{\text{src1,src3,RecentLiker}}(S_{\text{likes}}, S_{\text{follows}}, S_{\text{posts}}) \]
\[ S_{\text{Related}} = \mathcal{P}_{\text{Notify}}^{\text{RecentLiker}+}(S_{\text{RecentLiker}}) \]
\[ \text{Answer} = \chi_{\phi}^{\text{src1, trg2, Notify}}(S_{\text{Related}}, S_p) \]

Logical query plan

Answer ⊢◁ φ₂ (src₁, trg₂, notify)
\[ \mathcal{P}_{RL}^{RLP} \]
\[ \mathcal{W}^{24} \]
\[ \mathcal{W}^{24} \]
\[ \mathcal{W}^{24} \]

Posts

\[ \mathcal{W}^{24} \]
\[ \mathcal{W}^{24} \]
\[ \mathcal{W}^{24} \]

Likes

\[ \mathcal{W}^{24} \]
\[ \mathcal{W}^{24} \]
\[ \mathcal{W}^{24} \]

Follows
Exploring the Plan Space

Equivalent SGA expressions for RPQ $Q : (a \cdot b \cdot c)^+$:

1. **P1-($\alpha - RA$):** $P_{d^+}^l(\times^{\text{src}_1, \text{trg}_3, d}_{\text{trg}_1 = \text{src}_2 \land \text{trg}_2 = \text{src}_3} (S_a, S_b, S_c))$
Exploring the Plan Space

Equivalent SGA expressions for RPQ $Q : (a \cdot b \cdot c)^+$:

1. $P_1 - (\alpha - RA): \mathcal{P}^I_{d^+} (\bigwedge_{\text{src}_1, \text{trg}_3, d_{\text{trg}_1 = \text{src}_2 \land \text{trg}_2 = \text{src}_3}} (S_a, S_b, S_c))$

2. $P_2 - \text{FA-based}: \mathcal{P}^I_{(a \cdot b \cdot c)^+} (S_a, S_b, S_c)$
Exploring the Plan Space

Equivalent SGA expressions for RPQ $Q : (a \cdot b \cdot c)^+$:

1. **P1**-$(\alpha - RA)$: $\mathcal{P}_d^l (\nabla_{\text{src}_1, \text{trg}_3, d}^\text{src}_1, \text{trg}_3 = \text{src}_2 \land \text{trg}_2 = \text{src}_3 (S_a, S_b, S_c))$

2. **P2**-FA-based: $\mathcal{P}_{(a \cdot b \cdot c)}^l (S_a, S_b, S_c)$

3. Hybrid Plans
   - **P3**-$\mathcal{P}_{(a \cdot d)}^l (S_a, \nabla_{\text{trg}_1 = \text{src}_2}^\text{src}_1, \text{trg}_2 = \text{src}_3 (S_b, S_c))$
   - **P4**-$\mathcal{P}_{(d \cdot c)}^l (S_c, \nabla_{\text{trg}_1 = \text{src}_2}^\text{src}_1, \text{trg}_2 = \text{src}_3 (S_a, S_b))$
SGA Transformation Rules

Involving W-SCAN

1. Commute with Union & filter, i.e., operators that do not alter intervals
2. $\sigma^\Phi(W^T(S)) = W^T(\sigma^\Phi(S))$
SGA Transformation Rules

Involving W-SCAN

1. Commute with Union & filter, i.e., operators that do not alter intervals
2. \( \sigma^\Phi(W^T(S)) = W^T(\sigma^\Phi(S)) \)

Involving PATH:

1. Alternation: \( P^a|b_d(S_a, S_b) = \bigcup_d(S_a, S_b) \)
2. Concatenation: \( P^{a \cdot b}_d(S_a, S_b) = \bowtie^{\text{trg}_1=\text{src}_2}_{\text{src}_1, \text{trg}_2, d}(S_a, S_b) \)
SGA Transformation Rules

Involving W-SCAN

1. Commute with Union & filter, i.e., operators that do not alter intervals
2. $\sigma^\Phi(\mathcal{W}^T(S)) = \mathcal{W}^T(\sigma^\Phi(S))$

Involving PATH:

1. Alternation: $\mathcal{P}_d^{a|b}(S_a, S_b) = \bigcup_d (S_a, S_b)$
2. Concatenation: $\mathcal{P}_d^{a:b}(S_a, S_b) = \mathcal{K}^{trg_1=src_2}_{src_1,trg_2,d} (S_a, S_b)$

There are other rules for other SGA operators adopted temporal relational algebra
Logical primitives to evaluate SGQ
Logical primitives to evaluate SGQ

1. Search space for SGQ evaluation plans
Logical primitives to evaluate SGQ

1. Search space for SGQ evaluation plans
2. There exists an SGA expression for any SGQ
Logical primitives to evaluate SGQ

1. Search space for SGQ evaluation plans
2. There exists an SGA expression for any SGQ
3. Decouples logical design & physical implementation
Logical primitives to evaluate SGQ

1. Search space for SGQ evaluation plans
2. There exists an SGA expression for any SGQ
3. Decouples logical design & physical implementation
4. Algebraic closure & composability
Streaming Graph Algebra

Logical primitives to evaluate SGQ

1. Search space for SGQ evaluation plans
2. There exists an SGA expression for any SGQ
3. Decouples logical design & physical implementation
4. Algebraic closure & composability
5. Querying graph structure (PGM is a future work!)
Implementation of an SGQ Processor
Can we execute it on a dataflow engine?

- Subgraph patterns → joins
- Path navigation queries → iteration/recursion
- Streaming support
  - Incremental + iterative
  - Differential Dataflow [McSherry et al., 2013]

Can we do better?

- Query structure
  - Recursion is limited to transitive closure
  - Streaming RPQ [Pacaci et al., 2020]

- Sliding windows have temporal patterns
  - Eliminate negative tuples for expirations
  - Simplified state maintenance for stateful operators
Can we execute it on a dataflow engine?

Answer

\[ \phi_2^{(\text{src1}, \text{trg2}, \text{notify})} \]

\[ \mathcal{P}_{RLP}^{RL^+} \]

\[ \mathcal{W}^{24} \]

\[ \phi_1^{\text{src1,src2,RL}} \]

\[ \mathcal{W}^{24} \]

posts

\[ \mathcal{W}^{24} \]

\[ \mathcal{W}^{24} \]

\[ \mathcal{W}^{24} \]

likes posts follows
Can we execute it on a dataflow engine?

- Subgraph patterns $\rightarrow$ joins

---

Answer

$\diamondsuit_{(src1,trg2,notify)}$

$\mathcal{P}_{RLP}^{RL+}$

$\mathcal{W}^{24}$

$\diamondsuit_{src1,src2,RL}^{\phi_1}$

$\mathcal{W}^{24}$

likes

$\mathcal{W}^{24}$

posts

$\mathcal{W}^{24}$

follows
Can we execute it on a dataflow engine?

- Subgraph patterns $\rightarrow$ joins
- Path navigation queries $\rightarrow$ iteration/recursion
Can we execute it on a dataflow engine?

- Subgraph patterns $\rightarrow$ joins
- Path navigation queries $\rightarrow$ iteration/recursion
- Streaming support

DIagram:

- Answer
  - $\Join^{\phi_2}_{(src1, trg2, notify)}$
  - $\mathcal{P}_{RLP}^{RL+}$
  - $\mathcal{W}^{24}$
    - $\Join^{\phi_1}_{src1, src2, RL}$
    - $\mathcal{W}^{24}$
      - $\mathcal{W}^{24}$
      - $\mathcal{W}^{24}$
        - $\mathcal{W}^{24}$
          - $\mathcal{W}^{24}$
            - $\mathcal{W}^{24}$
              - $\mathcal{W}^{24}$
                - likes
                - posts
                - follows
Can we execute it on a dataflow engine?

- Subgraph patterns $\rightarrow$ joins
- Path navigation queries $\rightarrow$ iteration/recursion
- Streaming support
- Incremental + iterative
Can we execute it on a dataflow engine?

- Subgraph patterns $\rightarrow$ joins
- Path navigation queries $\rightarrow$ iteration/recursion
- Streaming support
- Incremental + iterative
- Differential Dataflow [McSherry et al., 2013]
Can we execute it on a dataflow engine?

- Subgraph patterns $\rightarrow$ joins
- Path navigation queries $\rightarrow$ iteration/recursion
- Streaming support
  - Incremental + iterative
  - Differential Dataflow [McSherry et al., 2013]

Can we do better?

- Query structure
  - Recursion is limited to transitive closure
  - Streaming RPQ [Pacaci et al., 2020]
Can we execute it on a dataflow engine?

- Subgraph patterns → joins
- Path navigation queries → iteration/recursion
- Streaming support
- Incremental + iterative
- Differential Dataflow [McSherry et al., 2013]

Can we do better?

- Query structure
  - Recursion is limited to transitive closure
  - Streaming RPQ [Pacaci et al., 2020]

- Sliding windows have temporal patterns
  - Eliminate negative tuples for expirations
  - Simplified state maintenance for stateful operators
Experimental Evaluation

Comparison against baseline (DD)

• Stackoverflow: up to 4 × higher tput
• LDBC SNB: 30% lower – up to 6 × higher tput

Automata-based RPQ vs fixed-point iteration
• Better over cyclic graphs

Direct Approach
• Reduced cost of expirations
• Higher memory footprint

Please see the paper for more details!
Experimental Evaluation

Comparison against baseline (DD)

- Stackoverflow: up to 4× higher tput
- LDBC SNB: 30% lower – up to 6× higher tput

Please see the paper for more details!
Experimental Evaluation

Comparison against baseline (DD)

- Stackoverflow: up to 4× higher tput
- LDBC SNB: 30% lower – up to 6× higher tput

*Automata-based RPQ vs fixed-point iteration*

- Better over cyclic graphs
Experimental Evaluation

Comparison against baseline (DD)

- Stackoverflow: up to 4× higher tput
- LDBC SNB: 30% lower – up to 6× higher tput

Automata-based RPQ vs fixed-point iteration

- Better over cyclic graphs

Direct Approach

- Reduced cost of expirations
- Higher memory footprint

Please see the paper for more details!
Experimental Evaluation

Comparison against baseline (DD)

- Stackoverflow: up to $4 \times$ higher tput
- LDBC SNB: 30% lower – up to $6 \times$ higher tput

*Automata-based* RPQ vs *fixed-point* iteration

- Better over cyclic graphs

*Direct Approach*

- Reduced cost of expirations
- Higher memory footprint

*Please see the paper for more details!*
Experimental Evaluation

Comparison against baseline (DD)

- Stackoverflow: up to 4× higher tput
- LDBC SNB: 30% lower – up to 6× higher tput

Automata-based RPQ vs fixed-point iteration

- Better over cyclic graphs

Direct Approach

- Reduced cost of expirations
- Higher memory footprint

A well-known problem

1. Choose the “right” evaluation plan

Please see the paper for more details!
Experimental Evaluation

Comparison against baseline (DD)

- Stackoverflow: up to 4× higher tput
- LDBC SNB: 30% lower – up to 6× higher tput

Automata-based

- RPQ vs fixed-point iteration
  - Better over cyclic graphs

Direct

- Reduced cost of expirations
- Higher memory footprint

A well-known problem

1. Choose the “right” evaluation plan
2. Alternative physical operators?
  - WCO optimal joins [Ammar et al., 2018]

Please see the paper for more details!
Designing a *Streaming Graph Query* processor
Wrapping Up!

Designing a *Streaming Graph Query* processor

- Streaming Graph Queries & Algebra
  - Logical primitives for query evaluation
Designing a *Streaming Graph Query* processor

- Streaming Graph Queries & Algebra
  - Logical primitives for query evaluation
- A prototype query processor
  - Physical implementations of SGA operators
Designing a *Streaming Graph Query* processor

- Streaming Graph Queries & Algebra
  - Logical primitives for query evaluation
- A prototype query processor
  - Physical implementations of SGA operators
- Exciting directions
  - Cost-based query planning (ongoing!!)
  - Attribute-based predicates & PGM support
Designing a *Streaming Graph Query* processor

- Streaming Graph Queries & Algebra
  - Logical primitives for query evaluation
- A prototype query processor
  - Physical implementations of SGA operators
- Exciting directions
  - Cost-based query planning (ongoing!!!)
  - Attribute-based predicates & PGM support
- More in our project page

https://dsg-uwaterloo.github.io/s-graffito/


http://dx.doi.org/10.1145/3318464.3389733.