Regular Path Query Evaluation on Streaming Graphs

Anil Pacaci $^1$  Angela Bonifati $^2$  M. Tamer Özsu $^1$

$^1$University of Waterloo

$^2$Lyon 1 University
Streaming Graphs

Characteristics of 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
Characteristics of 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
Streaming Graphs

Characteristics of 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
Streaming Graphs

Characteristics of Streaming Graphs

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

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

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

(a) Credit-card fraud

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

- 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 Graphs

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

**Our setting:** Query processing over streaming graphs

- Persistent graph queries on streaming data
- Real-time results that are continuously updated

(a) Credit-card fraud
(b) Denial-of-service (DOS) attack

(Taken from [Qiu et al., 2018])
(Taken from [Choudhury et al., 2015])
What is a graph query?

Graph queries feature:

Subgraph Pattern

\[
\begin{array}{c}
\text{u}_1 \\
\text{c} \\
\text{u}_2
\end{array}
\]

\(\text{worksAt} \quad \text{worksAt} \quad \text{follows}\)
What is a graph query?

Graph queries feature:

**Subgraph Pattern**

```
\[ u_1 \xrightarrow{\text{worksAt}} c \xrightarrow{\text{worksAt}} u_2 \]
```

**Path Navigation**

```
(P_1 \xrightarrow{(\text{follows} \cdot \text{mentions})^+} P_2)
```
What is a graph query?

Graph queries feature:

- **Subgraph Pattern**

![Subgraph Pattern Diagram]

- **Path Navigation**

  $(\text{follows} \cdot \text{mentions})^+$

- **Path Navigation Queries**
  - Property paths in SPARQL v1.1
  - Single-label reachability in Cypher
  - Path expressions in G-CORE & PGQL
What is a graph query?

Graph queries feature:

**Subgraph Pattern**

**Path Navigation**

- Path Navigation Queries
  - Property paths in SPARQL v1.1
  - Single-label reachability in Cypher
  - Path expressions in G-CORE & PGQL

- Regular Path Queries
  - Generalized reachability & traversal
  - Directed paths that match regular expressions
- RPQs are heavily used in practice
  - SPARQL v1.1, Cypher, PGQL, G-CORE
  - 1 in 4 queries in Wikidata query logs [Bonifati et al., 2019]
Regular Path Queries

- RPQs are heavily used in practice
  - SPARQL v1.1, Cypher, PGQL, G-CORE
  - 1 in 4 queries in Wikidata query logs [Bonifati et al., 2019]

- RPQ evaluation on static graphs
  - Tractability results [Mendelzon and Wood, 1995; Bagan et al., 2013]
  - Cost-based planning for SPARQL property paths [Yakovets et al., 2016]
Regular Path Queries

- RPQs are heavily used in practice
  - SPARQL v1.1, Cypher, PGQL, G-CORE
  - 1 in 4 queries in Wikidata query logs [Bonifati et al., 2019]

- RPQ evaluation on static graphs
  - Tractability results [Mendelzon and Wood, 1995; Bagan et al., 2013]
  - Cost-based planning for SPARQL property paths [Yakovets et al., 2016]

- Unboundedness & high streaming rates
  - Windowed processing
  - Incremental & non-blocking operators
Regular Path Queries

- RPQs are heavily used in practice
  - SPARQL v1.1, Cypher, PGQL, G-CORE
  - 1 in 4 queries in Wikidata query logs [Bonifati et al., 2019]

- RPQ evaluation on static graphs
  - Tractability results [Mendelzon and Wood, 1995; Bagan et al., 2013]
  - Cost-based planning for SPARQL property paths [Yakovets et al., 2016]

- Unboundedness & high streaming rates
  - Windowed processing
  - Incremental & non-blocking operators

- Continuous processing of streaming graphs
  - Largely focus on pattern matching
Regular Path Queries

- RPQs are heavily used in practice
  - SPARQL v1.1, Cypher, PGQL, G-CORE
  - 1 in 4 queries in Wikidata query logs [Bonifati et al., 2019]
- RPQ evaluation on static graphs
  - Tractability results [Mendelzon and Wood, 1995; Bagan et al., 2013]

The objective of this paper

**Persistent** RPQ evaluation over **streaming graphs**

- Incremental & non-blocking operators
- Continuous processing of streaming graphs
  - Largely focus on pattern matching
Our Contributions

Persistent Regular Path Query Evaluation on Streaming Graphs

1. The design space of persistent RPQ algorithms
   - Path semantics
   - Result semantics & stream types
Our Contributions

Persistent Regular Path Query Evaluation on Streaming Graphs

1. The design space of persistent RPQ algorithms
   - Path semantics
   - Result semantics & stream types

2. Path semantics used in practice
   - Simple paths (no repeating vertex): navigation on road networks
   - Arbitrary paths: reachability on communication networks
Our Contributions

Persistent Regular Path Query Evaluation on Streaming Graphs

1. The design space of persistent RPQ algorithms
   - Path semantics
   - Result semantics & stream types

2. Path semantics used in practice
   - Simple paths (no repeating vertex): navigation on road networks
   - Arbitrary paths: reachability on communication networks

3. Result semantics & stream types
   - Append-only streams with fast insertions
   - Support for explicit deletions
Our Contributions

Persistent Regular Path Query Evaluation on Streaming Graphs

1. The design space of persistent RPQ algorithms
   - Path semantics
   - Result semantics & stream types

2. Path semantics used in practice
   - Simple paths (*no repeating vertex*): navigation on road networks
   - Arbitrary paths: reachability on communication networks

3. Result semantics & stream types
   - Append-only streams with fast insertions
   - Support for explicit deletions

4. Physical operators for path navigation queries
   - Compatible with existing languages: Cypher, G-CORE, PGQL, SPARQL v1.1
   - Efficiency & efficacy in real-world workloads
Our Solution
An automata-based algorithm

- Automata transition graph to guide traversals
Design Space for Streaming RPQ

An automata-based algorithm

- Automata transition graph to guide traversals
- Spanning-tree index to encode partial results
An *automata-based algorithm*

- Automata transition graph to guide traversals
- Spanning-tree index to encode partial results

<table>
<thead>
<tr>
<th>Path Semantics</th>
<th>Result Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Append-only</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>$\mathcal{O}(n \cdot k^2)$</td>
</tr>
<tr>
<td>Simple$^1$</td>
<td>$\mathcal{O}(n \cdot k^2)$</td>
</tr>
</tbody>
</table>

$^1$These results hold in the absence of conflicts, a condition on cyclic structure of the query and graph.
Design Space for Streaming RPQ

An automata-based algorithm
- Automata transition graph to guide traversals
- Spanning-tree index to encode partial results

<table>
<thead>
<tr>
<th>Path Semantics</th>
<th>Result Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Append-only</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>$O(n \cdot k^2)$</td>
</tr>
<tr>
<td>Simple$^1$</td>
<td>$O(n \cdot k^2)$</td>
</tr>
</tbody>
</table>

$^1$These results hold in the absence of conflicts, a condition on cyclic structure of the query and graph.
Incremental Maintenance of Spanning Trees

Cross-edge Append-only Algorithm for Arbitrary Path Semantics

$Q_1 = (a \cdot b)^+$
Incremental Maintenance of Spanning Trees

Cross-edge Append-only Algorithm for Arbitrary Path Semantics

\[ Q_1 = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q_1 = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]

\[ Q = (a \cdot b)^+ \]
Incremental Maintenance of Spanning Trees

\[ Q_1 = (a \cdot b)^+ \]

Cross-edge

\[ \begin{align*}
(\text{start}, 0) & \rightarrow (s_0, a) \\
& \rightarrow (s_1, a) \rightarrow (s_2, b)
\end{align*} \]
Incremental Maintenance of Spanning Trees

$Q_1 = (a \cdot b)^+$

(start $s_0$) $a \rightarrow s_1$ $b \rightarrow s_2$
Incremental Maintenance of Spanning Trees

Cross-edge

Append-only Algorithm for Arbitrary Path Semantics

$Q_1 = (a \cdot b)^+$

(start $\rightarrow s_0 \rightarrow s_1 \rightarrow s_2$

$\rightarrow a \rightarrow b$

$\rightarrow \rightarrow \rightarrow$)

$\rightarrow (x, 0)$

$\rightarrow (y, 1)$

$\rightarrow (z, 1)$

$\rightarrow (u, 2)$

$\rightarrow (w, 2)$

$\rightarrow (v, 1)$

$\rightarrow$
Incremental Maintenance of Spanning Trees

\[ Q_1 = (a \cdot b)^+ \]

\[ \text{start} \rightarrow s_0 \rightarrow a \rightarrow s_1 \rightarrow b \rightarrow s_2 \]
Incremental Maintenance of Spanning Trees

Append-only Algorithm for Arbitrary Path Semantics

- $O(n)$ amortized insertion cost ($n =$ # of vertices in the window $W$)
- Expired tuples are removed in batches
- Explicit windows are supported via negative tuples
- A variation of *Counting* and *DRed* [Gupta et al., 1993]
Incremental Maintenance of Spanning Trees

\[ Q_1 = (a \cdot b)^+ \]
Incremental Maintenance of Spanning Trees

\[ Q_1 = (a \cdot b)^+ \]

Cross-edge

Append-only Algorithm for Arbitrary Path Semantics

▶ $O(n)$ amortized insertion cost ($n = \# \text{of vertices in the window } W$)

▶ Expired tuples are removed in batches

▶ Explicit windows are supported via negative tuples

▶ A variation of Counting and DRed [Gupta et al., 1993]
Incremental Maintenance of Spanning Trees

\[ Q_1 = (a \cdot b)^+ \]

Cross-edge

Cross-edge
Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]
Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]

- **Conflict-freeness**: A sufficient condition for efficient evaluation
  - A condition on the cyclic structure of the graph and the automata
  - An arbitrary path $p_a : u \rightarrow v \implies$ a simple path $p_s : u \rightarrow v$
How to Find Simple Paths?

Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]

- **Conflict-freeness**: A sufficient condition for efficient evaluation
  - A condition on the cyclic structure of the graph and the automata
  - An arbitrary path $p_a : u \rightarrow v \Rightarrow$ a simple path $p_s : u \rightarrow v$

- Requires DFS ordering
How to Find Simple Paths?

Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]

- **Conflict-freeness**: A sufficient condition for efficient evaluation
  - A condition on the cyclic structure of the graph and the automata
  - An arbitrary path $p_a : u \rightarrow v \implies$ a simple path $p_s : u \rightarrow v$

- Requires DFS ordering

Our contribution: A *streaming algorithm for conflict detection*
Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]

- **Conflict-freeness**: A sufficient condition for efficient evaluation
  - A condition on the cyclic structure of the graph and the automata
  - An arbitrary path $p_a : u \rightarrow v \implies$ a simple path $p_s : u \rightarrow v$

- Requires DFS ordering

Our contribution: *A streaming algorithm for conflict detection*

- Optimized for append-only streams
How to Find Simple Paths?

Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]

- **Conflict-freeness**: A sufficient condition for efficient evaluation
  - A condition on the cyclic structure of the graph and the automata
  - An arbitrary path $p_a : u \rightarrow v \implies$ a simple path $p_s : u \rightarrow v$

- Requires DFS ordering

Our contribution: A *streaming algorithm for conflict detection*

- Optimized for append-only streams
- An optimistic conflict-detection mechanism
How to Find Simple Paths?

Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]

- **Conflict-freeness**: A sufficient condition for efficient evaluation
  - A condition on the cyclic structure of the graph and the automata
  - An arbitrary path $p_a : u \rightarrow v \implies$ a simple path $p_s : u \rightarrow v$

- Requires DFS ordering

Our contribution: *A streaming algorithm for conflict detection*

- Optimized for append-only streams
- An optimistic conflict-detection mechanism
- Aggressive pruning – backtracking only if absolutely necessary
Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]

- **Conflict-freeness**: A sufficient condition for efficient evaluation
  - A condition on the cyclic structure of the graph and the automata
  - An arbitrary path \( p_a : u \rightarrow v \implies \) a simple path \( p_s : u \rightarrow v \)

- Requires DFS ordering

Our contribution: A *streaming algorithm for conflict detection*

- Optimized for append-only streams
- An optimistic conflict-detection mechanism
- Aggressive pruning – backtracking only if absolutely necessary
- Correctness & complexity proofs are in the paper
Performance Results

- Most common RPQs in Wikidata query logs [Bonifati et al., 2019]
Performance Results

- Most common RPQs in Wikidata query logs [Bonifati et al., 2019]
  - Update stream of LDBC SNB  [Erling et al., 2015]
  - Stackoverflow temporal graph
  - Yago2s RDF graph

Sub-millisecond tail latency (99th percentile)
Performance matching the complexity analysis
Finding simple paths
Over 70% of the workloads
Impact on the tail latency
Scalability analysis using synthetic RPQ workloads (gMark [Bagan et al., 2016])
Performance Results

- Most common RPQs in Wikidata query logs [Bonifati et al., 2019]
  - Update stream of LDBC SNB [Erling et al., 2015]
  - Stackoverflow temporal graph
  - Yago2s RDF graph
- Sub-millisecond tail latency (99th percentile)
Performance Results

- Most common RPQs in Wikidata query logs [Bonifati et al., 2019]
  - Update stream of LDBC SNB [Erling et al., 2015]
  - Stackoverflow temporal graph
  - Yago2s RDF graph

- Sub-millisecond tail latency (99th percentile)

- Performance matching the complexity analysis
Performance Results

- Most common RPQs in Wikidata query logs [Bonifati et al., 2019]
  - Update stream of LDBC SNB [Erling et al., 2015]
  - Stackoverflow temporal graph
  - Yago2s RDF graph

- Sub-millisecond tail latency (99th percentile)
- Performance matching the complexity analysis
- Finding simple paths
  - Over 70% of the workloads
  - $2 - 5\times$ impact on the tail latency
Performance Results

- Most common RPQs in Wikidata query logs [Bonifati et al., 2019]
  - Update stream of LDBC SNB [Erling et al., 2015]
  - Stackoverflow temporal graph
  - Yago2s RDF graph

- Sub-millisecond tail latency (99\textsuperscript{th} percentile)

- Performance matching the complexity analysis

- Finding simple paths
  - Over 70\% of the workloads
  - 2 \texttimes 5\% impact on the tail latency

- Scalability analysis using synthetic RPQ workloads (gMark [Bagan et al., 2016])
Wrapping-up

1. Design space of non-blocking RPQ operators
   - Path semantics
   - Result semantics

2. Persistent RPQs under arbitrary path semantics
   - Support for explicit edge deletions
   - Implicit & explicit window semantics

3. Streaming conflict-detection algorithm
   - Safely prune the search space to avoid exhaustive search
   - Tractable evaluation on conflict-free graphs (matching tractability results [Mendelzon and Wood, 1995])

4. Extensive of experiments with real-world workloads [Bonifati et al., 2019]
   - Scalability w.r.t. the window & query size
   - Feasibility of simple path semantics
Wrapping-up

1. Design space of non-blocking RPQ operators
   - Path semantics
   - Result semantics

2. Persistent RPQs under arbitrary path semantics
   - Support for explicit edge deletions
   - Implicit & explicit window semantics
Wrapping-up

1. Design space of non-blocking RPQ operators
   - Path semantics
   - Result semantics

2. Persistent RPQs under arbitrary path semantics
   - Support for explicit edge deletions
   - Implicit & explicit window semantics

3. Streaming conflict-detection algorithm
   - Safely prune the search space to avoid exhaustive search
   - Tractable evaluation on conflict-free graphs (matching tractability results [Mendelzon and Wood, 1995])

4. Extensive of experiments with real-world workloads [Bonifati et al., 2019]

Scalability w.r.t. the window & query size
Feasibility of simple path semantics
Wrapping-up

1. Design space of non-blocking RPQ operators
   - Path semantics
   - Result semantics

2. Persistent RPQs under arbitrary path semantics
   - Support for explicit edge deletions
   - Implicit & explicit window semantics

3. Streaming conflict-detection algorithm
   - Safely prune the search space to avoid exhaustive search
   - Tractable evaluation on conflict-free graphs (matching tractability results [Mendelzon and Wood, 1995])

4. Extensive of experiments with real-world workloads [Bonifati et al., 2019]
   - Scalability w.r.t. the window & query size
   - Feasibility of simple path semantics


Additional Slides
A streaming RDF engine

- C-SPARQL [Barbieri et al., 2009], CQELS [Le-Phuoc et al., 2011], SPARQL\textsubscript{stream} [Calbimonte et al., 2010], W3C RSP-QL [Dell’Aglio et al., 2015]
- Based on SPARQL v1.0 - no path navigation
- Our algorithms can be incorporated to provide incremental RPQ evaluation

A graph analytics engine for dynamic graphs

- GraphOne [Kumar and Huang, 2020], GraphBolt [Mariappan and Vora, 2019], GraphTau [Iyer et al., 2016]
- Efficient maintenance of graph snapshots
- Iterative graph analytics in the vertex-centric model
- We focus on persistent evaluation of path queries that are specified declaratively
**Workloads**

**Table:** Most common RPQs in real workloads [Bonifati et al., 2019].

<table>
<thead>
<tr>
<th>Name</th>
<th>Query</th>
<th>Name</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>$a^*$</td>
<td>$Q_7$</td>
<td>$a \circ b \circ c^*$</td>
</tr>
<tr>
<td>$Q_2$</td>
<td>$a \circ b^*$</td>
<td>$Q_8$</td>
<td>$a? \circ b^*$</td>
</tr>
<tr>
<td>$Q_3$</td>
<td>$a \circ b^* \circ c^*$</td>
<td>$Q_9$</td>
<td>$(a_1 + a_2 + \cdots + a_k)^+$</td>
</tr>
<tr>
<td>$Q_4$</td>
<td>$(a_1 + a_2 + \cdots + a_k)^*$</td>
<td>$Q_{10}$</td>
<td>$(a_1 + a_2 + \cdots + a_k) \circ b^*$</td>
</tr>
<tr>
<td>$Q_5$</td>
<td>$a \circ b^* \circ c$</td>
<td>$Q_{11}$</td>
<td>$a_1 \circ a_2 \circ \cdots \circ a_k$</td>
</tr>
<tr>
<td>$Q_6$</td>
<td>$a^* \circ b^*$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Throughput & Tail Latency

(a) Yago2s

(b) LDBC SF10

(c) Stackoverflow
Figure: Size of the tree index $\Delta$ on the SO graph with $|W| = 30$ days.
Scalability - Tail Latency (99\textsuperscript{th} Percentile)

Figure: Tail latency with various $|W|$ and $\beta$
Figure: The average window maintenance cost with various $|W|$ and $\beta$
Figure: Impact of the ratio of explicit deletions on tail latency for all queries on Yago2s RDF graph with $|W| \approx 10M$ tuples.
Impact of DFA Construction

Figure: The impact of the query length $|Q_R|$ on the automata size, $k$, and the throughput. RPQs are generated using gMark [Bagan et al., 2016] where the query size ranges from 2 to 20. Each RPQ is formulated by grouping labels into concatenations and alternations of size up to 3, and each group has a 50% probability of having $*$ and $+$. 