Regular Path Query Evaluation on Streaming Graphs

Anil Pacaci ¹  Angela Bonifati ²  M. Tamer Özsu ¹

¹University of Waterloo
²Lyon 1 University
Streaming Graphs

Characteristics of Streaming Graphs

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

$t_{10}$
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
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

$t_{10}$ $t_{11}$ $t_{13}$ $t_{14}$ $t_{15}$ $t_{16}$
Streaming Graphs

Characteristics of Streaming Graphs

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

| \( t_{10} \) | \( t_{11} \) | \( t_{13} \) | \( t_{14} \) | \( t_{15} \) | \( t_{16} \) |
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
   (Taken from [Qiu et al., 2018])

(b) Denial-of-service (DOS) attack
   (Taken from [Choudhury et al., 2015])
What is a graph query?

Graph queries feature:

*Subgraph Pattern*

- $u_1$ worksAt $c$
- $c$ worksAt $u_2$
- $u_1$ follows $u_2$
What is a graph query?

Graph queries feature:

Subgraph Pattern

Path Navigation

\[(\text{follows} \cdot \text{mentions})^+\]
What is a graph query?

Graph queries feature:

- **Subgraph Pattern**
  - \( u_1 \) (worksAt) \( c \) (worksAt) \( u_2 \) (follows)

- **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**
  
  
  \[
  \begin{align*}
  u_1 \xrightarrow{\text{worksAt}} c \xrightarrow{\text{worksAt}} u_2 \quad \text{follows}
  \end{align*}
  \]

- **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

- **Regular Path Queries**
  - Generalized reachability & traversal
  - Directed paths that match regular expressions
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]
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
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
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
Design Space for Streaming RPQ

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
**Design Space for Streaming RPQ**

*An automata-based algorithm*

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

| Path Semantics | Result Semantics |
|----------------|------------------|
|                | Append-only      | Explicit Deletions |
| Arbitrary      | $\mathcal{O}(n \cdot k^2)$ | $\mathcal{O}(n^2 \cdot k)$ |
| Simple$^1$     | $\mathcal{O}(n \cdot k^2)$ | $\mathcal{O}(n^2 \cdot k)$ |

---

$^1$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

| Path Semantics | Result Semantics |
|----------------|------------------|
|                | Append-only      | Explicit Deletions |
| Arbitrary      | $O(n \cdot k^2)$ | $O(n^2 \cdot k)$   |
| Simple$^1$     | $O(n \cdot k^2)$ | $O(n^2 \cdot k)$   |

$^1$ 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

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

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

Cross-edge

\( (x, 0) \)
\( (y, 1) \)
\( (z, 1) \)
\( (u, 2) \)
\( (v, 1) \)
Incremental Maintenance of Spanning Trees

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

\[ \begin{align*}
Q_1 &= (a \cdot b)^+ \\
\text{start} &\rightarrow s_0 \rightarrow a \rightarrow s_1 \rightarrow b \rightarrow s_2
\end{align*} \]
Incremental Maintenance of Spanning Trees

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

\[ \text{start} \xrightarrow{a} s_0 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \]

Cross-edge 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

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

(start) $s_0$ $s_1$ $s_2$

Cross-edge 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

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]

$(v, 1)$
Incremental Maintenance of Spanning Trees

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

Start $s_0 \rightarrow a \rightarrow s_1 \rightarrow b \rightarrow s_2$

Cross-edge 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

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

Cross-edge 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

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

Cross-edge

Cross-edge
How to Find Simple Paths?

Finding simple paths for arbitrary graphs and queries is NP-complete [Mendelzon and Wood, 1995]
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 $p_s : u \rightarrow v$

- **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
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
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*
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
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
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 ($99^{th}$ 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 (99\textsuperscript{th} 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 (99$^{th}$ percentile)
- Performance matching the complexity analysis
- Finding simple paths
  - Over 70% of the workloads
  - 2 – 5× 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<sup>th</sup> percentile)
- Performance matching the complexity analysis
- Finding simple paths
  - Over 70% of the workloads
  - 2 – 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
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

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])

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
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
Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G. H., Lemay, A., and Advokaat, N. (2016). gmark: schema-driven generation of graphs and queries. *IEEE Trans. Knowl. and Data Eng.*, 29(4):856–869.

Bagan, G., Bonifati, A., and Groz, B. (2013). A trichotomy for regular simple path queries on graphs. In *Proc. 32nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 261–272.

Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E., and Grossniklaus, M. (2009). C-sparql: Sparql for continuous querying. In *Proc. 18th Int. World Wide Web Conf.*, pages 1061–1062.

Bonifati, A., Martens, W., and Timm, T. (2019). Navigating the maze of wikidata query logs. In *Proc. 28th Int. World Wide Web Conf.*, pages 127–138. ACM.

Calbimonte, J.-P., Corcho, O., and Gray, A. J. (2010). Enabling ontology-based access to streaming data sources. In *Proc. 9th Int. Semantic Web Conf.*, pages 96–111. Springer.

Choudhury, S., Holder, L. B., Jr, G. C., Agarwal, K., and Feo, J. (2015). A Selectivity based approach to Continuous Pattern Detection in Streaming Graphs. In *Proc. 18th Int. Conf. on Extending Database Technology*, pages 157–168.
Dell’Aglio, D., Calbimonte, J.-P., Della Valle, E., and Corcho, O. (2015). Towards a unified language for rdf stream query processing. In Proc. 12th Extended Semantic Web Conf., pages 353–363. Springer.

Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.-D., and Boncz, P. (2015). The ldbc social network benchmark: Interactive workload. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 619–630. http://doi.acm.org/10.1145/2723372.2742786.

Gupta, A., Mumick, I. S., and Subrahmanian, V. S. (1993). Maintaining views incrementally. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 157–166.

Iyer, A. P., Li, L. E., Das, T., and Stoica, I. (2016). Time-evolving graph processing at scale. In Proc. 4th Int. Workshop on Graph Data Management Experiences and Systems, pages 1–6.

Kent, A. D., Liebrock, L. M., and Neil, J. C. (2015). Authentication graphs: Analyzing user behavior within an enterprise network. Computers & Security, 48:150–166.

Kumar, P. and Huang, H. H. (2020). Graphone: A data store for real-time analytics on evolving graphs. ACM Trans. Storage, 15(4):1–40.
Le-Phuoc, D., Dao-Tran, M., Parreira, J. X., and Hauswirth, M. (2011). A native and adaptive approach for unified processing of linked streams and linked data. In Proc. 10th Int. Semantic Web Conf., pages 370–388. Springer.

Mariappan, M. and Vora, K. (2019). Graphbolt: Dependency-driven synchronous processing of streaming graphs. In Proc. 14th ACM SIGOPS/EuroSys European Conf. on Comp. Syst., pages 1–16.

Mendelzon, A. O. and Wood, P. T. (1995). Finding regular simple paths in graph databases. SIAM J. on Comput., 24(6):1235–1258.

Qiu, X., Cen, W., Qian, Z., Peng, Y., Zhang, Y., Lin, X., and Zhou, J. (2018). Real-time constrained cycle detection in large dynamic graphs. Proc. VLDB Endowment, 11(12):1876–1888.

Yakovets, N., Godfrey, P., and Gryz, J. (2016). Query planning for evaluating sparql property paths. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 1875–1889. ACM.
Additional Slides
1. A streaming RDF engine
   - C-SPARQL [Barbieri et al., 2009], CQELS [Le-Phuoc et al., 2011], SPARQL$_{stream}$ [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

2. 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
Table: Most common RPQs in real workloads [Bonifati et al., 2019].

| Name | Query                  | Name | Query                  |
|------|------------------------|------|------------------------|
| Q₁   | \( a^* \)              | Q₇   | \( a \circ b \circ c^* \) |
| Q₂   | \( a \circ b^* \)     | Q₈   | \( a? \circ b^* \)    |
| Q₃   | \( a \circ b^* \circ c^* \) | Q₉   | \( (a_1 + a_2 + \cdots + a_k)^+ \) |
| Q₄   | \( (a_1 + a_2 + \cdots + a_k)^* \) | Q₁₀  | \( (a_1 + a_2 + \cdots + a_k) \circ b^* \) |
| Q₅   | \( a \circ b^* \circ c \) | Q₁₁  | \( a_1 \circ a_2 \circ \cdots \circ a_k \) |
| Q₆   | \( a^* \circ b^* \)    |      |                        |
Throughput & Tail Latency

(a) Yago2s

(b) LDBC SF10

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

Figure: Tail latency with various $|W|$ and $\beta$
Figure: The average window maintenance cost with various $|W|$ and $\beta$. 
**Explicit Deletions**

**Figure:** Impact of the ratio of explicit deletions on tail latency for all queries on Yago2’s RDF graph with $|W| \approx 10M$ tuples.
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 $+$. 