TimeReach: Historical Reachability Queries on Evolving Graphs

Paweł Borycki

7 May 2015

Konstantinos Semertzidis
Kostas Lillis
Evaggelia Pitoura

University of Ioannina, Greece

Plan of presentation

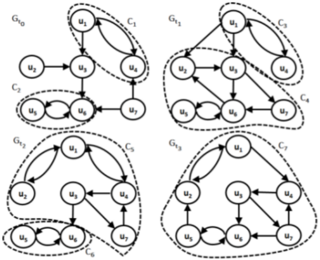
1. Historical Reachability Problem
   - Definitions
   - Version Graph (VG)
     - Historical Transitive Closure
     - Online Traversal of the Version Graph (INS and INT)
   - TimeReach (TR)
     - Strongly connected components (SCC)
   - Condensed TimeReach (TRC)
   - Interval 2Hop Labels
2. Experimental Evaluation
   - Index Size
   - Index Construction Time
   - Query Processing

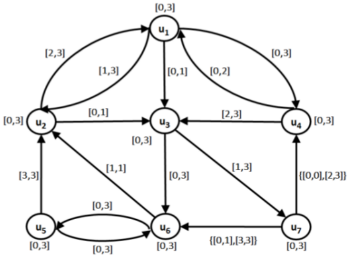
Historical Reachability Query

- $G = (V, E)$
  - Directed graph
- $G_t = (V_t, E_t)$
  - Graph snapshot at time instant t
- $\mathcal{G}_{[t_i, t_j]} = \{G_{t_i}, ..., G_{t_j}\}$
  - Evolving graph in time interval $[t_i, t_j]$

- **Historical reachability query**
  - **Social networks, citations, computer and hyperlink networks**
  - Evolving graph $\mathcal{G}_{[t_i, t_j]}$
  - Time interval I_Q
  - Nodes u, v
  - **Conjunctive –** $u \overset{I_{Q\wedge}}{\rightsquigarrow} v$ **?**
  - **Disjunctive –** $u \overset{I_{Q\vee}}{\rightsquigarrow} v$ **?**

Version Graph





- $\mathcal{L}(u), \mathcal{L}(e)$
  - **Lifespan** of temporal elements
  - Set of intervals
    - $[t_i, t_j] \in \mathcal{L}(u) \leftrightarrow \forall\ m\ t_i \leq t_m \leq t_j\ .\ u \in V_{t_m}$
- $VG_I = (V_I, E_I, \mathcal{L}_u, \mathcal{L}_e)$
  - **Version graph**
  - $V_I = \cup_{t_m \in I} V, \quad E_I = \cup_{t_m \in I} E$
  - $\mathcal{L}_u : V_I \to \mathcal{I}, \quad \mathcal{L}_e : E_I \to \mathcal{I}$
    - $\mathcal{I}$ – set of intervals

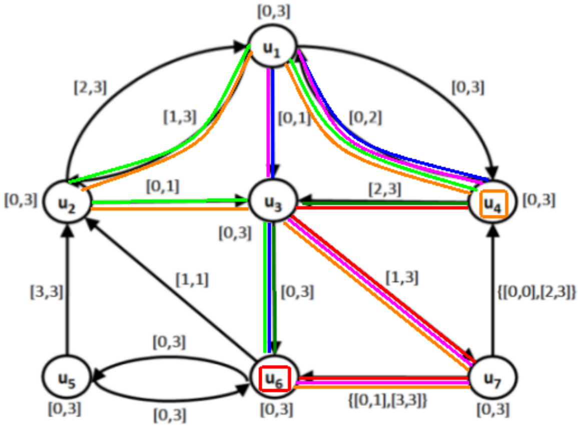
Operations on lifespans

- **Minimum** set of intervals
  - Contains no pairs of **overlapping** or **continous** intervals
  - Intervals as **bit arrays** or **ordered lists**
- $\mathcal{I} \otimes \mathcal{I}'$
  - **Join**
  - Set of time instants common to both intervals
- $\mathcal{I} \oplus \mathcal{I}'$
  - **Merge**
  - Minimum set equivalent to $\mathcal{I} \cup \mathcal{I}'$

- $\mathcal{L}(p) = \mathcal{L}_e(e_1) \otimes ... \otimes \mathcal{L}_e(e_m)$
  - **Lifespan of a path** $p = e_1 ... e_m$
- $P(u, v) = \{p_1, ..., p_l\}$
  - Set of all paths from u to v
- $\mathcal{L}(u, v) = \mathcal{L}(p_1) \oplus ... \oplus \mathcal{L}(p_l)$
  - **Lifespan of the reachability** between u and v
- $u \overset{I_{Q\wedge}}{\rightsquigarrow} v$ = true
  - $\{I_Q\} \otimes \mathcal{L}(u, v) \sqsupseteq I_Q$
- $u \overset{I_{Q\vee}}{\rightsquigarrow} v$ = true
  - $\{I_Q\} \otimes \mathcal{L}(u, v) \neq \emptyset$

Lifespan of the reachability

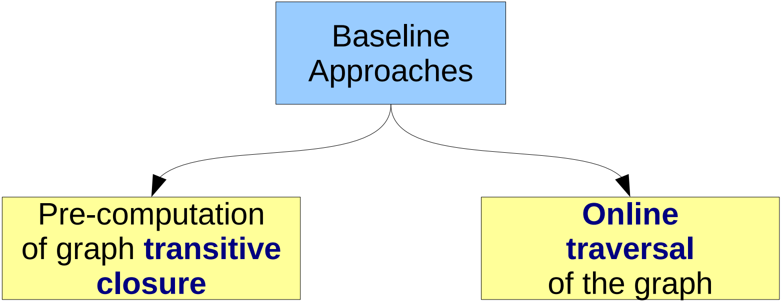
Example: $\mathcal{L}(u_4, u_6) = [0, 3]$



$$
\begin{array}{r}
[2,3] \\
[3,3] \\
[0,1] \\
[1,1] \\
[1,1] \\
\oplus\ [1,1] \\
\hline
[0,3]
\end{array}
$$

Approaches to reachability queries

Basic approaches in **static graphs**:



Historical Transitive Closure

Algorithm 1 TransitiveClosure(VG_I)

Input: Version graph VG_I
Output: The transitive closure CL_I

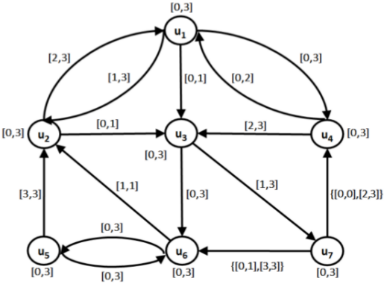
for all $u, v \in V_I \times V_I$ **do**
  if $(u,v) \in E_I$ **then**
    $CL_I(u,v) = \mathcal{L}_e((u,v))$
  else
    $CL_I(u,v) = \emptyset$
  end if
end for
for $w = 1$ **to** $|V_I|$ **do**
  for all $u, v \in V_I \times V_I$ **do**
    $CL_I(u,v) = CL_I(u,v) \oplus (CL_I(u,w) \otimes CL_I(w,v))$
  end for
end for

Path from u to v through w

Transitive Closure computation

- CL_I
  - Single transitive closure for version graph VG_I
  - Contains $\mathcal{L}(u, v)$
- Construction
  - Variation of Floyd-Warshall algorithm
    - **Time** – $O(|V_I|^3 \cdot T)$
    - **Storage** – $O(|V_I|^2)$
- **Transitive Closure (TC)**
  - Solving $u \overset{I_{Q\wedge}}{\rightsquigarrow} v$, $u \overset{I_{Q\vee}}{\rightsquigarrow} v$
    - **Constant time**

Online Traversal of the Version Graph



Version graph

- **Instant-based traversal (INS)**
  - Traverse only edges with $\mathcal{L}_e(e) \sqsupseteq t$
  - Once for each $t \in I_Q$
- **Interval-based traversal (INT)**
  - Avoid multiple traversals in conjunctive queries
- Pruning tests
  - Not passing **node or edge** not living during query interval
  - $\mathcal{IN}(w)$ – parts of query interval for which node w has been traversed
    - $\mathcal{IN}(w) \sqsupseteq I \rightarrow$ **traversal is pruned**
- Worst case complexity of INT and INS
  - $O(|V_I| + |E_I|) \cdot |I_Q|)$
  - **INT outperforms INS**

Interval-based traversal of the Version Graph

Algorithm 2 Disjunctive-BFS(VG_I, u, v, $\{I_Q\}$)

Input: Version graph VG_I, nodes u, v, interval $I_Q \subseteq I$
Output: True if v is reachable from u in any time instant in I_Q and false otherwise

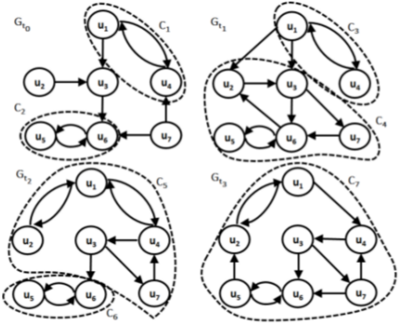
create a queue N, create a queue INT
enqueue u onto N, enqueue I_Q onto INT
while $N \neq \emptyset$ **do**
  $n \leftarrow N.dequeue()$
  $i \leftarrow INT.dequeue()$
  for all w s.t. (n, w) in VG_I and $\{I_Q\} \otimes \mathcal{L}_e((n,w)) \neq \emptyset$ **do**
    if w $== v$ **then**
      Return(true)
    end if
    $\mathcal{I}' = \{I_Q\} \otimes \mathcal{L}_e(u,w)$
    if $\mathcal{IN}(w) \not\supseteq \mathcal{I}'$ **then**
      $\mathcal{IN}(w) = \mathcal{IN}(w) \oplus \mathcal{I}'$
      enqueue w onto N
      enqueue $\mathcal{I}'$ onto INT
    end if
  end for
end while
Return(false)

Algorithm 3 Conjunctive-BFS(VG_I, u, v, $\{I_Q\}$)

Input: Version graph VG_I, nodes u, v, interval $I_Q \subseteq I$
Output: True if v is reachable from u in all time instants in I_Q and false otherwise

create a queue N, create a queue INT
enqueue u onto N, enqueue I_Q onto INT
while $N \neq \emptyset$ **do**
  $n \leftarrow N.dequeue()$
  $i \leftarrow INT.dequeue()$
  for all w s.t. (n, w) in VG_I and $\{I_Q\} \otimes \mathcal{L}_e((n,w)) \neq \emptyset$ **do**
    $\mathcal{I}' = \{I_Q\} \otimes \mathcal{L}_e(n,w)$
    if $w == v$ **then**
      $R = R \oplus \mathcal{I}'$ *Part of $L(u,v) \oplus I_Q$ already covered*
      if $\mathcal{R} \sqsupseteq I_Q$ **then**
        Return(true)
      end if
      continue
    end if
    if $\mathcal{IN}(w) \not\supseteq \mathcal{I}'$ **then**
      $\mathcal{IN}(w) = \mathcal{IN}(w) \oplus \mathcal{I}'$
      enqueue w onto N
      enqueue $\mathcal{I}'$ onto INT
    end if
  end for
end while
Return(false)

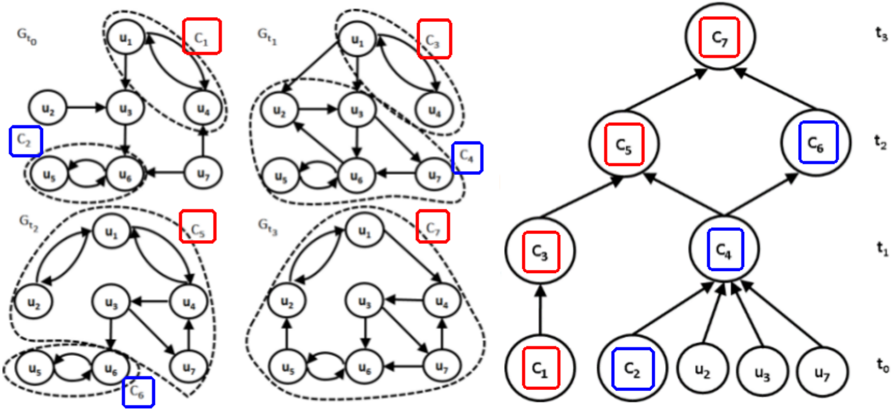
Strongly connected components (SCC)



Strong connected components (SCC) in an evolving graph

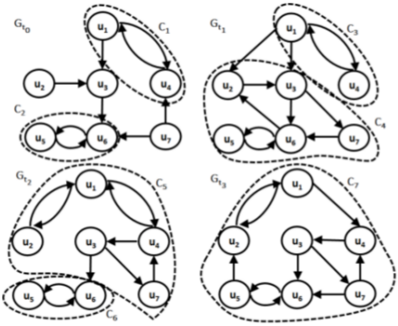
- **Social graphs** characterized by large SCC
- Every node in SCC reachable from any other node in this SCC
  - Identified by Tarjan's algorithm
    - **Time complexity –** $O(|V_I| + |E_I|) \cdot |I|)$
  - Different SCCs at each snapshot
- $P(u) = \{(C, t)\}$
  - Posting list for node u
  - C – component, t – time instant
    - **Storage complexity –** $\Omega(|V_I| \cdot |I|)$
  - Strong connections
    - **Posting sharing**

Strongly connected components (SCC)



SCC evolution over time

TimeReach (TR)

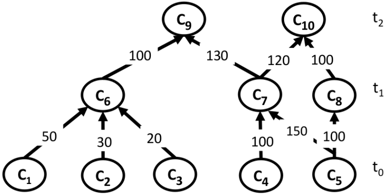


Strong connected components (SCC) in an evolving graph

- $G_{S_{t_k}} = (V_{S_{t_k}}, E_{S_{t_k}})$
  - **SCC graph snapshot**
    - **Nodes – SCC in G_{t_k}**
    - **Edges – edges between any nodes of SCC in G_{t_k}**
- $\mathcal{G}_{S_I} = \{G_{S_{t_i}}, ..., G_{S_{t_j}}\}$
  - Evolving SCC graph in a time interval $I = [t_i, t_j]$
- **TimeReach (TR) approach**
  - $u \overset{I_{Q\wedge}}{\rightsquigarrow} v$, $u \overset{I_{Q\vee}}{\rightsquigarrow} v$?
  - For each $t \in I_Q$:
    - **u and v belong to the same SCC?**
    - **Their SCC are reachable in corresponding G_{S_t}?**

Condensed TimeReach (TRC)

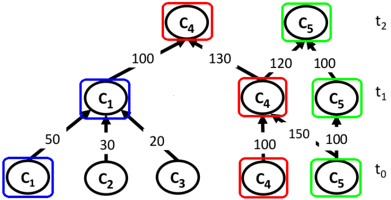
Nodes	Posting List
1-50	$(C_1,t_0),(C_6,t_1),(C_9,t_2)$
51-80	$(C_2,t_0),(C_6,t_1),(C_9,t_2)$
81-100	$(C_3,t_0),(C_6,t_1),(C_9,t_2)$
101-200	$(C_4,t_0),(C_7,t_1),(C_9,t_2)$
201-230	$(C_5,t_0),(C_7,t_1),(C_9,t_2)$
231-350	$(C_5,t_0),(C_7,t_1),(C_{10},t_2)$
351-450	$(C_5,t_0),(C_8,t_1),(C_{10},t_2)$



- Optimal SCC-ID assignment
  - **Minimum number of postings** for time interval *I* and a set of SCC
  - Re-assign IDs of postings
- $G_C = (V_C, E_C, \mathcal{W})$
  - Weighted graph
    - **Nodes – SCC at any time instant**
    - $(U, V) \in E_C \leftrightarrow \exists u \in VG_I$ **.** $(U, t) \in P(u) \wedge (V, t+1) \in P(u)$
    - $\mathcal{W}(U, V)$ **– number of nodes that belong to both U and V**
- $G_{C_{[t_k, t_{k+1}]}}$
  - Subgraph of G_C
  - Only SCC having existed at interval $[t_k, t_{k+1}]$
    - **Bipartite graph**

Condensed TimeReach (TRC)

Nodes	Posting List
1-50	$(C_1,[t_0,t_1])$, $(C_4,[t_2,t_2])$
51-80	$(C_2,[t_0,t_0])$, $(C_1,[t_1,t_1])$, $(C_4,[t_2,t_2])$
81-100	$(C_3,[t_0,t_0])$, $(C_1,[t_1,t_1])$, $(C_4,[t_2,t_2])$
101-200	$(C_4,[t_0,t_2])$
201-300	$(C_5,[t_0,t_0])$, $(C_4,[t_1,t_2])$
231-350	$(C_5,[t_0,t_0])$, $(C_4,[t_1,t_1])$, $(C_5,[t_2,t_2])$
351-450	$(C_5,[t_0,t_2])$



- Optimal SCC-ID assignment
  - **Minimum number of postings** for time interval *I* and a set of SCC
  - Re-assign IDs of postings
- Maximize the weight of edges between node with the same ID
  - **Maximum weight bipartite matching** M_k of each $G_{C_{[t_k, t_{k+1}]}}$
    - **Approximation greedy algorithm –** $O(|E|)$
- **Condensed TimeReach (TRC)**
  - Evolving SCC graph
- $VG_{S_I} = (V_{S_I}, E_{S_I}, \mathcal{L}_u, \mathcal{L}_e)$
  - **Condensed version graph**
  - Version graph of evolving SCC graph

Condensed TimeReach (TRC)

Algorithm 4 ConstructSccPostings(G_t, P_{t-1}, $G_{S_{[t-2,t-1]}}$)

Input: Snapshot G_t, SCC postings P_{t-1}
Output: SCC postings P_t

```
S_SCC_t = ∅, M = ∅
Run Tarjan's algorithm on G_t
S_SCC_t is the set of the detected SCCs where each
  SCC_i ∈ S_SCC_t is assigned a unique id C_i
if t > 0 then
   Construct G_S[t-1,t] from S_SCC_t and G_S[t-2,t-1]
   Compute maximum weight matching M
   for all edges e = (U, V) ∈ M do
      C_v = C_u
   end for
end if
for all nodes u ∈ V_t do
   find SCC_i ∈ S_SCC_t s.t. u ∈ SCC_i
   if P_t-1(u) ≠ ∅ then
      if P_t-1(u)[end].C ≠ C_i then
         P_t-1(u)[end].I = [t_s, t − 1]
         P_t-1(u).add(C_i, [t, curr])
      end if
   else
      P_t-1(u).add(C_i, [t, curr])
   end if
end for
P_t = P_t-1
```

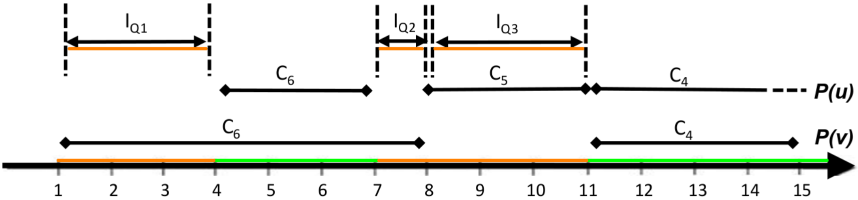
Weighted graph

MWM computation

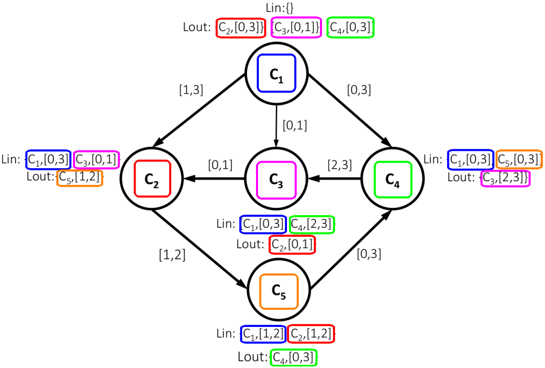
- Tarjan's algorithm for computing SCC
  - $O(|V_t| + |E_t|)$
- Weighted bipartite graph construction
  - $O(|E_{C_{[t-1,t]}}|)$
- MWM computation and new SCC IDs assignment
  - Greedy algorithm
  - $O(|E_{S_{[t-1,t]}}|)$
- Update SCC postings for nodes
  - only if changed since $t-1$
  - $O(|V_t|)$
- **Algorithm complexity –** $O(|V_t| + |E_t|)$

Query Processing in Condensed TimeReach (TRC)

- Query Processing ($u \overset{I_Q \wedge}{\rightsquigarrow} v$, $u \overset{I_Q \vee}{\rightsquigarrow} v$?)
  - u and v belong to the same SCC during the whole interval
  - Otherwise check reachability between SCC in other subintervals
    - $u \overset{I_{Q_1} \wedge}{\rightsquigarrow} C_6$, $u \overset{I_{Q_2} \wedge}{\rightsquigarrow} C_6$, $C_5 \overset{I_{Q_3} \wedge}{\rightsquigarrow} v$
  - Combine results using $\wedge$ or $\vee$
- Time cost at worst case
  - Traverse condensed version graph for all time instants t
  - $O(|I_Q| \cdot (|V_{S_i}| + |E_{S_i}|))$
    - **Size od condensed version graph $\leftarrow$ Optimal CSS assignment**



Interval 2Hop Labels



Interval 2Hop Labels

- Labels for each node u
  - $L_{in}(u), L_{out}(u)$
  - For each node $w \in L_{in}(u) \cup L_{out}(u)$
    - **Reachability lifespan** $\mathcal{L}(u, w)$ **or** $\mathcal{L}(w, u)$
  - Constructed by BNF search
    - **Start from largest** $(indeg(u) + 1) \times (outdeg(u) + 1)$
- u reaches v
  - $L_{in}(v) \cap L_{out}(u) \neq \emptyset$
  - Lifespans satisfy $\overset{I_{Q\wedge}}{\rightsquigarrow}$ or $\overset{I_{Q\vee}}{\rightsquigarrow}$

Experimental Evaluation

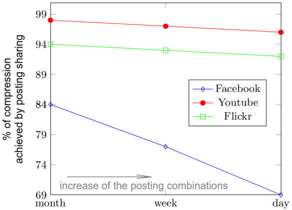
Snapshot Granularity		# nodes		# edges		# SCC		Max SCC (# nodes)	
		first	last	first	last	first	last	first	last
FB	(daily) 871	117	61,096	128	1,139,081	10	374	3	51,286
	(weekly) 125	1,429	61,096	2,365	1,139,081	138	374	18	51,286
	(monthly) 29	4,239	61,096	12,224	1,139,081	279	374	96	51,286
YT	(daily) 37	1,004,777	1,138,499	4,379,283	4,452,646	9,807	11,360	457,932	509,332
	(weekly) 6	1,025,536	1,138,499	4,379,283	4,452,646	9,807	11,360	465,668	509,332
	(monthly) 2	1,116,602	1,138,499	4,446,042	4,452,646	10,664	11,360	485,273	509,332
FL	(daily) 134	1,487,058	2,302,925	17,022,083	33,140,018	42,163	58,636	1,004,426	1,605,184
	(weekly) 20	1,507,700	2,302,925	17,393,321	33,140,018	42,163	58,636	1,010,498	1,605,184
	(monthly) 5	1,585,173	2,302,925	18,987,847	33,140,018	42,459	58,636	1,081,499	1,605,184

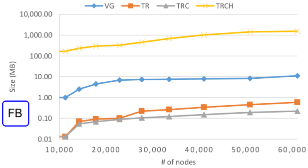
VG	Version Graph
TC	Transitive Closure
TR	(Simple) TimeReach
TRC	Condensed TimeReach
TRCH	Condensed TimeReach with 2hop labels
INS	Instant-based traversal of the version graph
INT	Interval-based traversal of the version graph

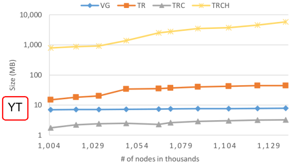
Experimental Evaluation – Index Size

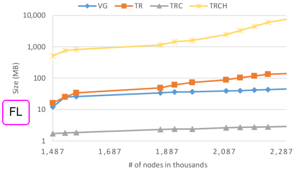
- Graph size
  - Larger SCC → higher TRC compression
- Percentage of deletes
  - ↓ TR, TRC – isolated nodes are disconnected from SCC
  - ↑ TRCH – additional nodes must be included in labels to ensure reachability tests
  - VG – size of labels remains constant

% of deletes	Size (MB)			
	VG	TR	TRC	TRCH
0	11	0.5	0.21	1,493
10	11	0.58	0.22	1,528
20	11	0.45	0.19	1,612
30	11	0.47	0.18	1,664



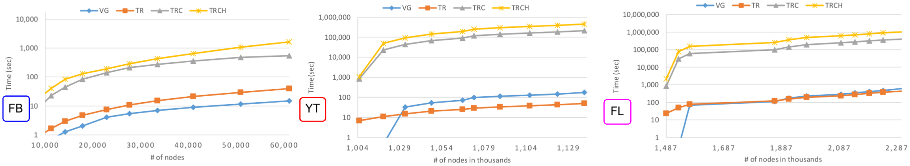






Experimental Evaluation – Construction Time

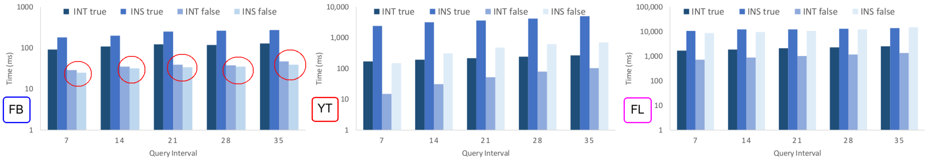
- VG → TR → TRC → TRCH
- Greedy algorithm for TRC fast and precise enough



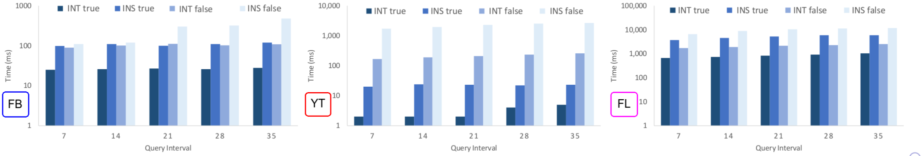
Experimental Evaluation – Query Processing

- Source and target chosen randomly
- **Interval-based faster than Instant-based**
  - Instant-based false conjunctive queries in FB (answer found immediately)

$$u \overset{I_{Q\wedge}}{\rightsquigarrow} v \ ?$$

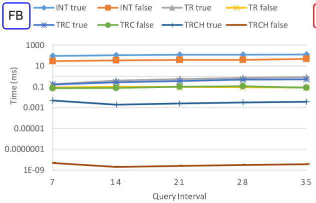


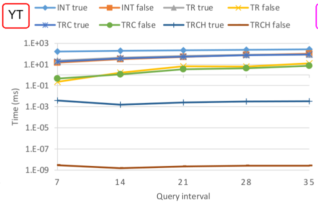
$$u \overset{I_{Q\vee}}{\rightsquigarrow} v \ ?$$

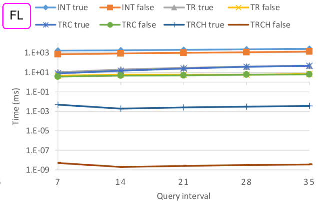


Experimental Evaluation – Query Processing

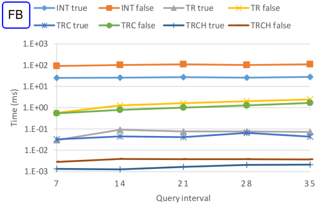
$u \overset{I_{Q\wedge}}{\rightsquigarrow} v$?

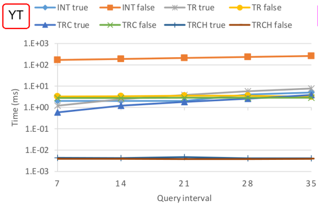


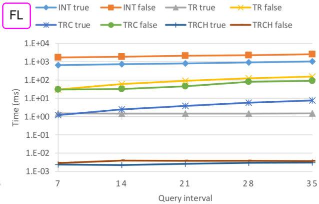




$u \overset{I_{Q\vee}}{\rightsquigarrow} v$?







TimeReach: Historical Reachability Queries on Evolving Graphs

Thank you.