Efficiently Answering Reachability and Path Queries on Temporal Bipartite Graphs

Xiaoshuang Chen*, Kai Wang*, Xuemin Lin*, Wenjie Zhang*, Lu Qin#, Ying Zhang#

*University of New South Wales, #University of Technology Sydney

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

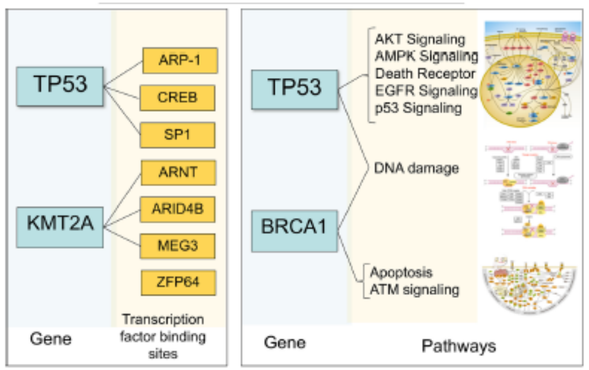
Outline

- Introduction
- Problem Definition
- Solution Overview
  - BFS-based Online Approaches
  - A 2-hop Labelling Index-based Solution
- Efficient Index Construction Algorithms
- Extensions
- Performance Study

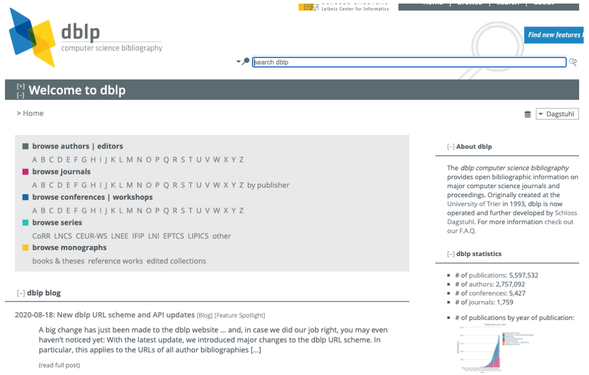
Introduction

Background

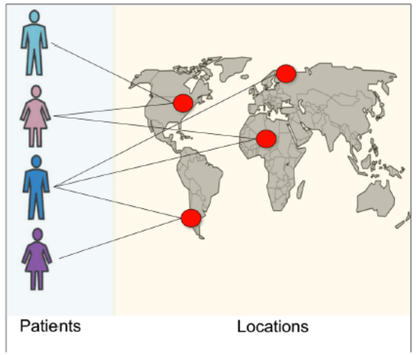
Bipartite graph serves as a useful data model when modelling relationships between **two different types of entities**. For example,



Biomolecular Network



Author-paper Network

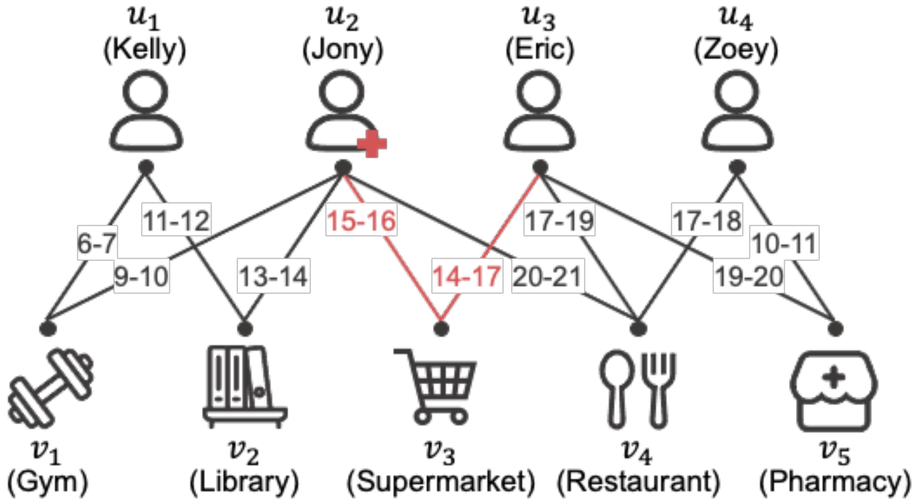


People-location Network

To capture complex situations and network dynamics, bipartite graphs are enriched with node attributes, edge importance and edge timestamps, yielding attributed bipartite graphs, weighted bipartite graphs and temporal bipartite graphs…..

Motivation

Specifically, the temporal bipartite graph further records two timestamps (i.e., the starting and ending times) for each edge, and it is an effective model in many real-world applications. For example



A people-location network for modeling disease outbreaks, which is derived from the paper [1] in Nature. Each edge has two timestamps denoting an individual's arriving and leaving times (in 24-hour clock) at the location.

Physical Contact Pattern: people visit a location simultaneously. For example, physical contact exists between Jony and Eric.

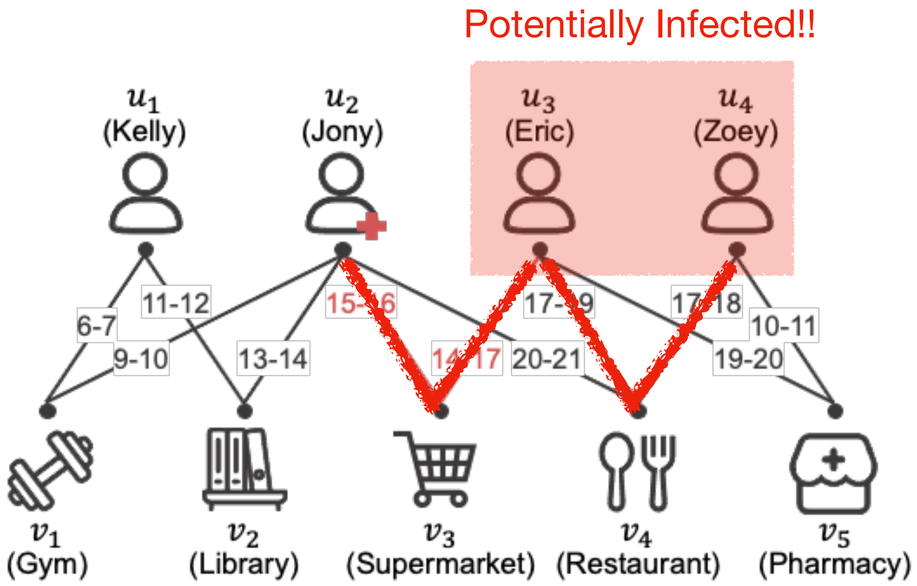
How to identify if an individual is potentially infected by a virus carrier through a series of physical contacts based on the temporal bipartite graph model?

Reachability!

[1] Eubank, S., Guclu, H., Kumar, V. A., Marathe, M. V., Srinivasan, A., Toroczkai, Z., & Wang, N. (2004). Modelling disease outbreaks in realistic urban social networks. Nature, 429(6988), 180-184.

Motivation

- Even though reachability has been extensively studied on (temporal) unipartite graphs, it remains largely unexplored on temporal bipartite graphs.
- Existing works on (temporal) unipartite graphs do not consider the special characteristics of temporal bipartite graph structure.



Temporal Bipartite Reachability: *we are the first to study the reachability problem on temporal bipartite graphs. By considering the special characteristics of temporal bipartite graph structure, the temporal bipartite reachability is actually in a 2-hop manner.*

To model the physical contact pattern: time-overlapping wedge.

To model the transmission: time-respecting path (i.e., a series of consecutive time-overlapping wedges and the times of the passing wedges follow a non-decreasing order).

Applications

The temporal bipartite reachability has many applications, which includes

- Supporting control of disease outbreaks in Epidemiology.
- Tracing metabolic pathways in Biochemistry.
- Modeling the flow of information in online social networks

….

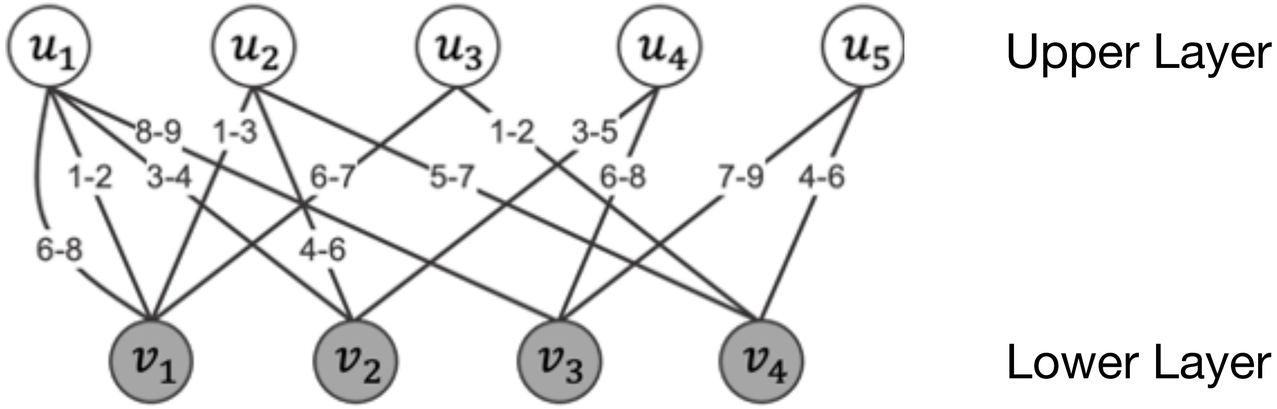
Problem Definition

Basic Concepts

A **temporal bipartite graph** $G(V = (U, L), E)$ includes:

- a set of vertices $U(G)$ in the upper layer
- a set of vertices $L(G)$ in the lower layer
- a set of temporal edges $E(G)$

s.t. $U(G) \cap L(G) = \emptyset$, $V(G) = U(G) \cup L(G)$ denotes the vertex set, and $E(G) \subseteq U(G) \times L(G)$ includes the temporal edges where each edge (u, v, t_s, t_e) records the starting time t_s and ending time t_e of the relationship between u and v.

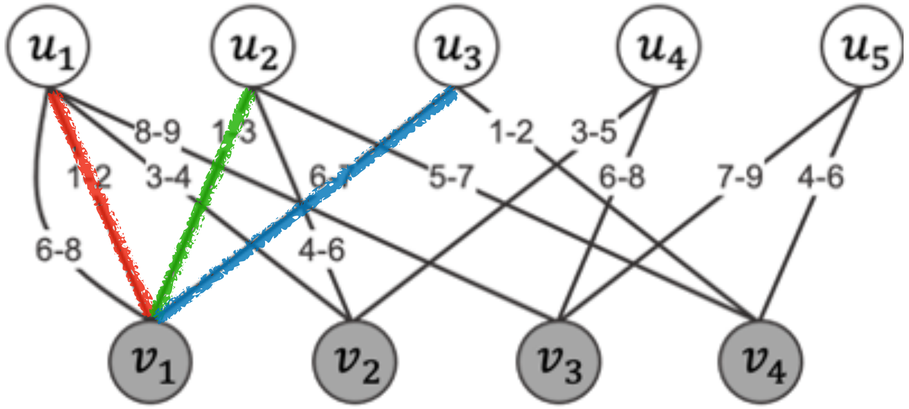


A temporal bipartite graph G

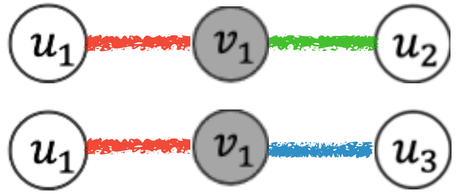
Basic Concepts

Wedge: given a temporal bipartite graph G and three vertices u, v, $w \in V(G)$, a wedge is a path starting from u, going through v and ending at w.

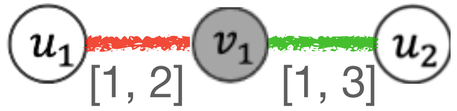
Time-overlapping Wedge: a wedge is a time-overlapping wedge, (denoted by $W = (e_1 = (u, v, t_s, t_e), e_2 = (v, w, t'_s, t'_e))$, if the time intervals of the two edges are overlapped, i.e., $\min(t_e, t'_e) > \max(t_s, t'_s)$.



Wedges:

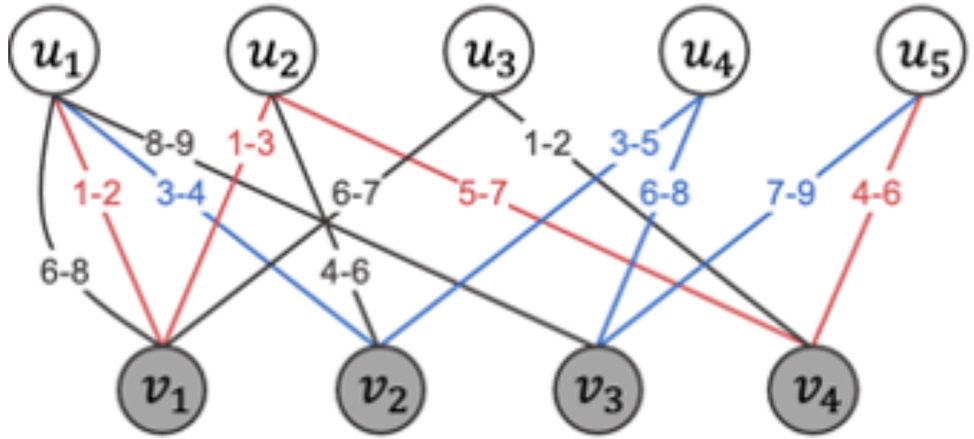


Time-overlapping Wedges:



Basic Concepts

Time-respecting Path: a time-respecting path, denoted by $P = \langle W_1, W_2 \cdots, W_k \rangle$, is a sequence of consecutive time-overlapping wedges such that for any $i \in [1, k-1]$, the ending vertex of W_i equals the starting vertex of W_{i+1}, and the ending time of W_i is not larger than the starting time of W_{i+1}.



$W_1 = ((u_1, v_1, 1, 2), (v_1, u_2, 1, 3))$, $W_2 = ((u_2, v_4, 5, 7), (v_4, u_5, 4, 6))$ are two time-overlapping wedges. $P = \langle W_1, W_2 \rangle$ is a time-respecting path from u_1 to u_5 (marked in red), which starts at 1 and ends at 6.

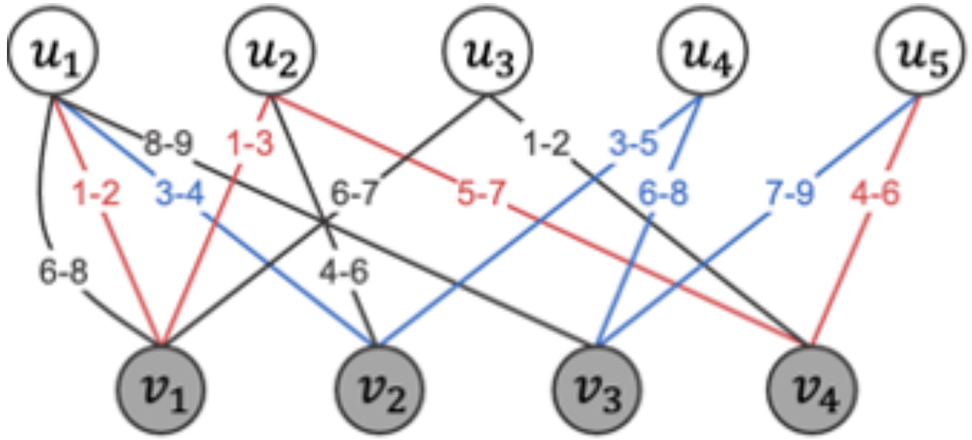
Similarly, the path marked in blue is also a time-respecting path.

Basic Concepts

Single-pair reachability: a vertex u reaches a vertex w within a time interval I (denoted by $u \rightsquigarrow_I w$), if there exists a time-respecting path P from u to w such that the starting time and the ending time of P fall into I.

Single-source reachability: given a vertex u and a time interval I, the single-source reachability aims to identify a vertex set including all the same-layer vertices that u can reach within I.

Earliest-arrival path: given two vertices u, w, and a time interval I, a time-respecting path P is an earliest-arrival path from u to w if it has the minimum ending time among all the paths in $\mathscr{P}$, where $\mathscr{P}$ is a set including all the time-respecting paths from u to w within I.



In the figure, we have $u_1 \rightsquigarrow_{[1,9]} u_5$. The set of vertices that u_1 can reach within [1,9] is $\{u_2, u_3, u_4, u_5\}$. P (marked in red) is an earliest-arrival path from u_1 to u_5 within [1,9].

Problem Statement

Given a temporal bipartite graph G, two vertices u, w, and a time interval I, we aim to efficiently answer the following queries:

(1) **single-pair reachability query**, which answers if u can reach w within I;

(2) **single-source reachability query**, which returns a set of vertices that u can reach within I;

(3) **earliest-arrival path query**, which retrieves an earliest-arrival path from u to w within I.

We first look into the solutions for single-pair reachability queries…

Solution Overview

BFS-based Online Approaches

- **BFS-based Online Approaches:** OReach and OReach$^+$ are two bfs-based online approaches. OReach$^+$ optimizes the computation of OReach by adopting the direction-optimizing breadth-first search [7].

[7] Beamer, S., Asanovic, K., & Patterson, D. (2012, November). Direction-optimizing breadth-first search. In SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (pp. 1-10). IEEE.

Algorithm 1: OReach (OReach$^+$)

Input : a temporal bipartite graph G, two vertices $u \in U(G)$ and $w \in U(G)$, and a time interval $\mathcal{I} = [I_s, I_e]$;
Output : the single-pair reachability from u to w

1 $Q \leftarrow$ an empty queue; $Q.push((u, I_s))$;
2 $minT[u] \leftarrow 0$; $minT[u'] \leftarrow \infty$ for $\forall u' \in U(G) \backslash \{u\}$;
3 $maxMinT[v] \leftarrow \infty$ for $\forall v \in L(G)$;
/* for reverse breadth-first search */
4 $P \leftarrow$ an empty queue; $P.push((w, I_e))$;
5 $maxT[w] \leftarrow \infty$; $maxT[u'] \leftarrow 0$ for $\forall u' \in U(G) \backslash \{w\}$;
6 $minMaxT[v] \leftarrow 0$ for $\forall v \in L(G)$;
7 **while** $Q \neq \emptyset$ **and** $P \neq \emptyset$ **do**
8 **if** $Q.size \leq P.size$ **then**
9 **while** $Q \neq \emptyset$ **do**
10 $(u', t_e) \leftarrow Q.pop()$;
11 **foreach** unvisited $e = (u', v, t_1, t_2)$ of $u' : t_1 \geq t_e$ **do**
12 mark e as visited; $t \leftarrow 0$;
13 **if** $t_1 \geq maxMinT[v]$ **then continue**;
14 **foreach** $w' \in \mathcal{V}(N_G(v))$ **do**
15 **foreach** $e' = (v, w', t'_1, t'_2) : t'_2 < minT[w']$ **do**
16 **if** e, e' are time-overlapping: $t'_2 \leq I_e$ **then**
17 **if** $w' = w$ **then return** *true*;
18 (**if** $minT[w'] \leq maxT[w']$ **then return** *true*;)
19 $Q.push((w', t'_2))$; $minT[w'] \leftarrow t'_2$;
20 **if** $minT[w'] > t$ **then** $t \leftarrow minT[w']$;
21 $maxMinT[v] \leftarrow t$;
22 **else**
23 The procedure of performing reverse breadth-first search based on P is similar to Lines 9-21;
24 **return** *false*;

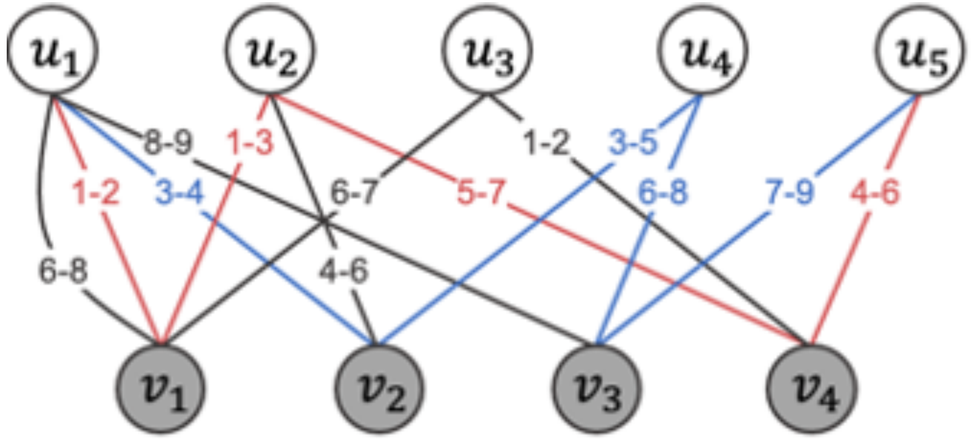
BFS starting from u

common vertex

Reverse BFS starting from w

A 2-hop Labeling Index-based Solution (Index Structure)

- **TBP-index:** Given a temporal bipartite graph G, a TBP-index L includes two label sets, namely in-label set L_{in} and out-label set L_{out}. For each vertex $u \in U(G)$, both L_{in} and L_{out} records a series of triplets. Specifically,
  - each triplet $(w, t_s, t_e) \in L_{in}(u)$ indicates that w reaches u within $[t_s, t_e]$;
  - each triplet $(w', t'_s, t'_e) \in L_{out}(u)$ indicates that u reaches w' within $[t'_s, t'_e]$.



Vertex	$\mathcal{L}_{out}$	$\mathcal{L}_{in}$
u_1	-	-
u_2	$(u_1, 1, 2)$, $(u_1, 5, 9)$	$(u_1, 1, 3)$
u_3	$(u_1, 6, 8)$	$(u_1, 6, 7)$
u_4	$(u_1, 3, 4)$, $(u_2, 3, 6)$	$(u_1, 3, 5)$, $(u_2, 5, 8)$, $(u_2, 4, 5)$
u_5	$(u_1, 7, 9)$, $(u_2, 4, 7)$, $(u_4, 7, 8)$	$(u_1, 8, 9)$, $(u_1, 1, 6)$, $(u_2, 5, 6)$, $(u_4, 6, 9)$

A TBP-index of G

A 2-hop Labeling Index-based Solution (Index Structure)

- **Minimal TBP-index:** a TBP-index L is a minimal TBP-index if it is **complete**, and each of its index entries is **necessary**.
  - By complete, it means we can correctly answer all the single-pair reachability queries based on L;
  - By necessary, it means L will become incomplete if we remove any index entry in it.

Vertex	$\mathcal{L}_{out}$	$\mathcal{L}_{in}$
u_1	-	-
u_2	$(u_1, 1, 2), (u_1, 5, 9)$	$(u_1, 1, 3)$
u_3	$(u_1, 6, 8)$	$(u_1, 6, 7)$
u_4	$(u_1, 3, 4), (u_2, 3, 6)$	$(u_1, 3, 5), (u_2, 5, 8), (u_2, 4, 5)$
u_5	$(u_1, 7, 9), (u_2, 4, 7), (u_4, 7, 8)$	$(u_1, 8, 9), (u_1, 1, 6), (u_2, 5, 6), (u_4, 6, 9)$

A minimal TBP-index of G

A 2-hop Labeling Index-based Solution (Query Process)

Query processing with the index: given a TBP-index L, two vertices $u \in U(G)$ and $w \in U(G)$, and a time interval $[I_s, I_e]$, u reaches w within $[I_s, I_e]$ if one of the following conditions holds:

(1) $\exists (w, t_s, t_e) \in L_{out}(u) : [t_s, t_e] \subseteq [I_s, I_e]$;

(2) $\exists (u, t_s, t_e) \in L_{in}(w) : [t_s, t_e] \subseteq [I_s, I_e]$

(3) $\exists (w', t_s, t_e) \in L_{out}(u), (w', t'_s, t'_e) \in L_{in}(w) : ([t_s, t_e] \subseteq [I_s, I_e]) \wedge ([t'_s, t'_e] \subseteq [I_s, I_e]) \wedge (t_e \leq t'_s)$

The time complexity for answering whether u reaches w is $O(|L_{out}(u)| + |L_{in}(w)|)$.

Vertex	$\mathcal{L}_{out}$	$\mathcal{L}_{in}$
u_1	-	-
u_2	$(u_1, 1, 2), (u_1, 5, 9)$	$(u_1, 1, 3)$
u_3	$(u_1, 6, 8)$	$(u_1, 6, 7)$
u_4	$(u_1, 3, 4), (u_2, 3, 6)$	$(u_1, 3, 5), (u_2, 5, 8), (u_2, 4, 5)$
u_5	$(u_1, 7, 9), (u_2, 4, 7), (u_4, 7, 8)$	$(u_1, 8, 9), (u_1, 1, 6), (u_2, 5, 6), (u_4, 6, 9)$

We have u_1 reaches u_5 within [1,9] since there exists an entry $(u_1,8,9) \in L_{in}(u_5)$ such that the above condition (2) is satisfied.

Efficient Index Construction Algorithms

Important Concepts

Dominance: given a vertex $u \in U(G)$, and two index entries (w, t_s, t_e) and (w', t'_s, t'_e) in $L_{out}(u)$ (or $L_{in}(u)$), (w', t'_s, t'_e) **dominates** (w, t_s, t_e) if $w = w'$ and $[t'_s, t'_e] \subseteq [t_s, t_e]$.

Minimal Index Entry: an index entry is a minimal index entry if it cannot be dominated by other index entries.

Canonical Index Entry: given a vertex $u \in U(G)$, an index entry $(w, t_s, t_e) \in L_{out}(u)$ (resp. $L_{in}(u)$) is a canonical index entry, if it satisfies the following two conditions:

(1) it is a minimal index entry;

(2) $\nexists w' \in U(G)$ s.t. $u \rightsquigarrow_{I_1} w'$, $w' \rightsquigarrow_{I_2} w$, $I_1 \subseteq [t_s, t_e]$, $I_2 \subseteq [t_s, t_e]$ and $end(I_1) \leq start(I_2)$.

Each index entry in a minimal TBP-index L is a canonical index entry.

Basic Index Construction

Algorithm 3: TBP-build

Input : a temporal bipartite graph G, and a vertex order O;
Output : $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$

1 $\mathcal{L}_{in}(u), \mathcal{L}_{out}(u) \leftarrow \emptyset$ for all $u \in U(G)$;
2 **for** $k = 1, 2, \cdots, |U(G)|$ **do**
3 $u_k \leftarrow$ the k-th vertex in O; $\mathcal{L}'_{in} \leftarrow \emptyset$;
4 **foreach** unique starting time t_s from u_k **do**
5 MOReach$(G, u_k, t_s, \mathcal{L}'_{in})$;
6 find the canonical index entries in $\mathcal{L}'_{in}$, and add them to $\mathcal{L}_{in}$;
7 the process to update $\mathcal{L}_{out}$ is similar to Lines 3 - 6;
8 **return** $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$;
9 **Procedure** MOReach$(G, u, I_s, \mathcal{L}'_{in})$
10 run Algorithm 1 Lines 1-3; mark all $e \in E(U(G))$ as unvisited;
11 mark all $e = (u, v, t'_s, t'_e)$ adjacent to u with $t'_s \neq I_s$ as visited;
12 **while** $Q \neq \emptyset$ **do**
13 $(u', t_e) \leftarrow Q.pop()$;
14 **if** $u' \neq u \wedge$ no entries in $\mathcal{L}'_{in}(u')$ can dominate (u, I_s, t_e) **then**
15 remove all entries in $\mathcal{L}'_{in}(u')$ that (u, I_s, t_e) can dominate;
16 insert (u, I_s, t_e) into $\mathcal{L}'_{in}(u')$;
17 run Algorithm 1 Lines 11-19, replace Line 14 by **foreach** $w' \in \mathcal{V}(N_G(v)) : O(w') > O(u)$, remove $t'_2 \leq I_2$ and Line 17;

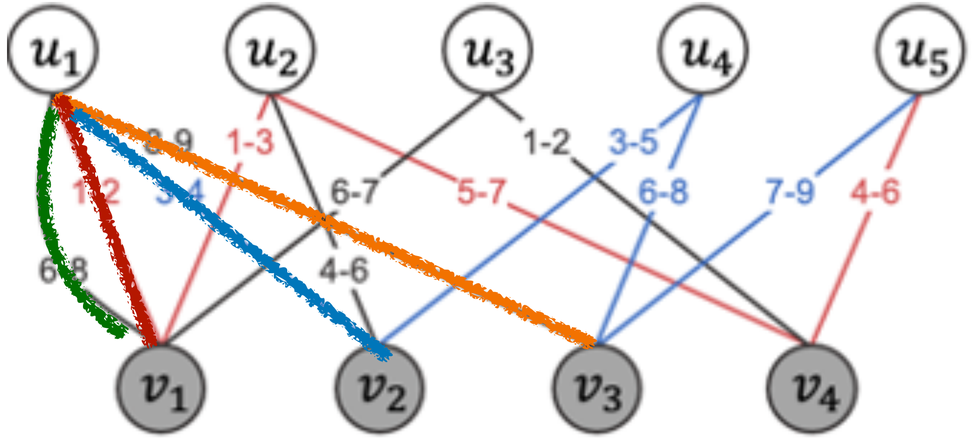
To Build a Minimal TBP-index:

(1) Performing BFS procedure starting from each vertex in the vertex order $\mathcal{O}$.

(2) For each starting vertex u_k, compute its related minimal index entries through MOReach.

(3) Compute canonical index entries and add them into the index.

Basic Index Construction - An Example

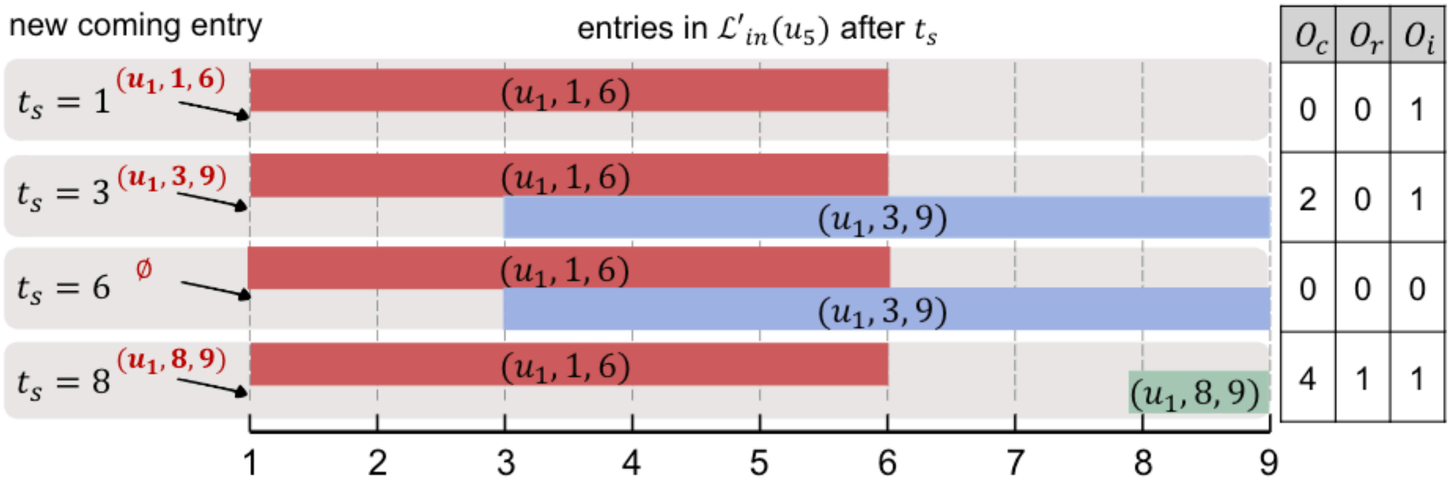


1. $\mathcal{O}(u_1) < \mathcal{O}(u_2) < \cdots < \mathcal{O}(u_5)$
2. At Line 4 of Algorithm 3, When indexing from u_1, t_s is processed in the order of 1, 3, 6, 8.

Vertex	$\mathscr{L}'_{in}$
u_1	-
u_2	$\{u_1,1,3\}$
u_3	$\{u_1,6,7\}$
u_4	~~$\{u_1,1,5\}$~~ $\{u_1,3,5\}$
u_5	$\{u_1,1,6\}$ ~~$\{u_1,3,9\}$~~ $\{u_1,8,9\}$

Advanced Index Construction

Observation 1: TBP-build incurs high computational cost in maintaining computing minimal index entries (or maintaining L'_{in}).



O_c	O_r	O_i
0	0	1
2	0	1
0	0	0
4	1	1

The computation of the minimal index entries starting from u_1 and ending at u_5, where O_c, O_r and O_i denote the number of comparison operations, removal operations and insertion operations, respectively

For example, to compute the minimal index entries between u_1 and u_5, TBP-build needs 10 comparisons in total. When the candidate index entry $\delta_1 = (u_1,3,9)$ comes, TBP-build will check if it can dominate or be dominated by the existing entry in $L'_{in}(u_5)$, i.e., $\delta_2 = (u_1,1,6)$ (two comparison operations).

Advanced Index Construction

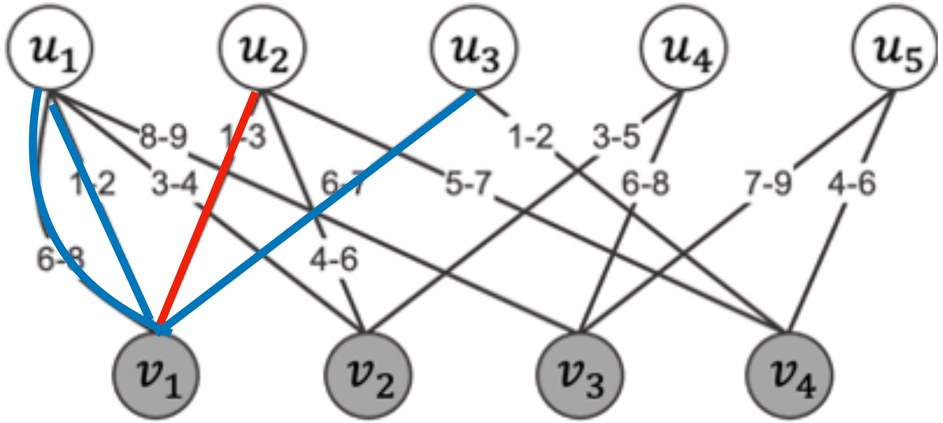
Our Solution - we propose *the time-priority-based traversal strategy:* processing the starting times from u_k in decreasing order when indexing from u_k, and for each starting time, exploring the index entries with the minimum ending time first.

By doing this, each found candidate index entry cannot be dominated by the entries that are explored afterward, and thus the computation in Line 15 of TBP-build can be completely pruned.

9 **Procedure** MOReach$(G, u, I_s, \mathcal{L}'_{in})$
10 run Algorithm 1 Lines 1-3; mark all $e \in E(U(G))$ as unvisited;
11 mark all $e = (u, v, t'_s, t'_e)$ adjacent to u with $t'_s \neq I_s$ as visited;
12 **while** $Q \neq \emptyset$ **do**
13 $(u', t_e) \leftarrow Q.pop()$;
14 **if** $u' \neq u \wedge$ no entries in $\mathcal{L}'_{in}(u')$ can dominate (u, I_s, t_e) **then**
15 ~~remove all entries in $\mathcal{L}'_{in}(u')$ that (u, I_s, t_e) can dominate;~~
16 insert (u, I_s, t_e) into $\mathcal{L}'_{in}(u')$;
17 run Algorithm 1 Lines 11-19, replace Line 14 by **foreach** $w' \in \mathcal{V}(N_G(v)):O(w') > O(u)$, remove $t'_2 \leq I_2$ and Line 17;

Advanced Index Construction

Observation 2: TBP-build incurs high computational cost in computing time-overlapping wedges.



Given v_1 and $e_1 = (u_2, v_1, 1, 3)$, TBP-build needs to compare e_1 with all the other edges adjacent to v_1. However, only 1 out of 3 comparisons leads to a valid time-overlapping wedge.

Advanced Index Construction

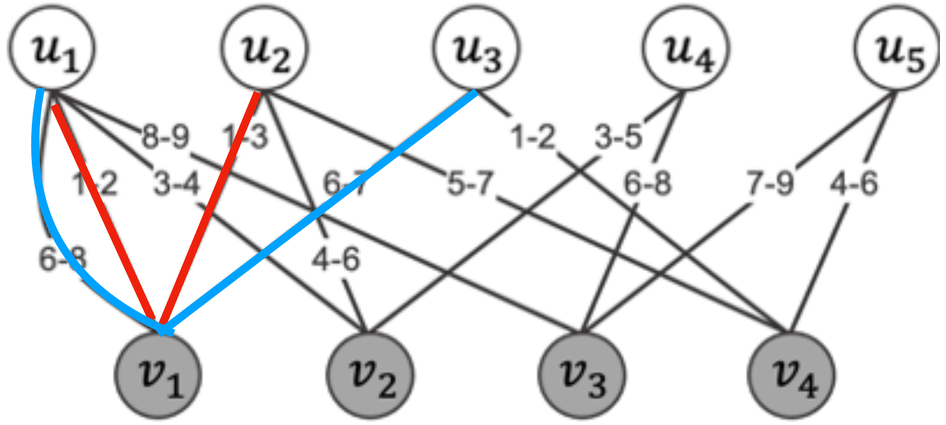
Reachability-equivalent Partition: *Given a temporal bipartite graph G, and a lower-layer vertex $v \in L(G)$, the reachability-equivalent partition of v, denoted by S_v, is a partition $\{E_1, E_2 \cdots E_T\}$ of $E(v)$ (i.e., the set of edges adjacent to v), such that $\forall 1 \leq i \neq j \leq T$:*

(1) $E_i \cap E_j = \emptyset$ and $E(v) = E_1 \cup \cdots \cup E_T$;

(2) $\forall e_1 \in E_i, e_2 \in E_j$, e_1 and e_2 cannot form a time-overlapping wedge;

(3) $|S_v|$ is maximized.

Our Solution: we propose the temporal-based edge partition technique. For each lower-layer vertex, we compute its reachability-equivalent partition. By doing this, only edges in the same subset of a partition can form time-overlapping wedges.



The reachability-equivalent partition of v_1 is $S_{v_1} = \{\{(u_1, v_1, 1, 2), (u_2, v_1, 1, 3)\}, \{(u_1, v_1, 6, 8), (u_3, v_1, 6, 7)\}\}$.

Given v_1 and $e_1 = (u_2, v_1, 1, 3)$, e_1 only needs to be compared with the edges in the same partition (i.e., $(u_2, v_1, 1, 2)$), which involves only 1 comparison.

Advanced Index Construction Algorithm

Algorithm 4: TBP-build*

Input : a temporal bipartite graph G and a vertex order O;
Output : $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$.

1 compute $\mathcal{S}_v$ (Definition 10) for $\forall v \in L(G)$;
2 $\mathcal{L}_{in}(u), \mathcal{L}_{out}(u) \leftarrow \emptyset$ for each vertex $u \in U(G)$;
3 **for** $k = 1, 2, \cdots, |U(G)|$ **do**
4 $u_k \leftarrow$ the k-th vertex in O; $Q \leftarrow$ an empty priority queue;
5 mark all $e \in E(U(G))$ as unvisited;
6 $minT[u] \leftarrow 0$; $minT[u'] \leftarrow \infty$ for $\forall u' \in U(G) \setminus \{u\}$;
7 $maxMinT[v] \leftarrow \infty$ for $\forall v \in L(G)$;
8 **foreach** unique starting time t_s *(in desc)* from u_k **do**
9 $Q.push((u_k, t_s, t_s))$;
10 **while** $Q \neq \emptyset$ **do**
11 $(w, t'_s, t'_e) \leftarrow Q.pop()$;
12 **if** $w \neq u_k$ **then**
13 **if** Query$(u_k, w, t'_s, t'_e, \mathcal{L})$ **then continue;**
14 **else** $\mathcal{L}_{in}(w) \leftarrow \mathcal{L}_{in}(w) \bigcup \{(u_k, t'_s, t'_e)\}$;
15 **foreach** unvisited $e = (w, v, t_1, t_2)$ of $w : t_1 \geq t'_e$ **do**
16 mark e as visited; $t \leftarrow 0$;
17 **if** $t_1 \geq maxMinT[v]$ **then continue**;
18 **foreach** $e' = (v, x, t'_1, t'_2)$ in $\mathcal{S}_v(e)$ **do**
19 **if** $O(x) \leq O(u_k)$ **then continue**;
20 **if** $t'_2 \geq minT[x]$ **then continue**;
21 **if** e and e' are time-overlapping **then**
22 $Q.push((x, t'_s, t'_2))$; $minT[x] \leftarrow t'_2$;
23 **if** $minT[x] > t$ **then** $t \leftarrow minT[x]$;
24 $maxMinT[v] \leftarrow t$;
25 **return** $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$.

(1) The time-priority-based traversal strategy.

(2) Compute the reachability-equivalent partition (Line 1), and when computing the time-overlapping wedges, only compare the edges within the same partition (Line 18).

Extensions

Extension 1: a lock- and atomic-free parallel algorithm, namely TBP-build*-PL, to compute a minimal TBP-index.

TBP-build*-PL contains two phases. In phase I, it generates local indexes for each thread and computes the index entries. In phase II, it cleans the none canonical index entries in each sub-index.

Algorithm 5: TBP-build*-PL

Input : G, O, and the number of threads t;
Output : $\mathcal{L}^1, \cdots, \mathcal{L}^t$

/* Phase I: compute a labeling that respects O. */

1 compute $\mathcal{S}_v$ (Definition 10) for $\forall v \in L(G)$;
2 initialize $\mathcal{L}^i_{in}, \mathcal{L}^i_{out} \leftarrow \emptyset$ for each thread $i \leftarrow 1 \cdots t$;
3 **for** $k = 1, 2, \cdots, |U(G)|$ **do**
4 $\quad u_k \leftarrow$ the k-th vertex in O, and allocate it to an idle thread i;
5 $\quad$ run Lines 5-24 in Algorithm 4; replace $\mathcal{L}$ in Line 13 and $\mathcal{L}_{in}$ in Line 14 with $\mathcal{L}^i$ and $\mathcal{L}^i_{in}$, respectively.

/* Phase II: clean non-canonical index entries. */

6 **for** $k = 1, 2, \cdots, |U(G)|$ **do**
7 $\quad u_k \leftarrow$ the k-th vertex in O, and allocate it to an idle thread;
8 $\quad$ **foreach** index entry $(w, t_s, t_e) \in \bigcup_{i=1}^{t} \mathcal{L}^i_{in}(u_k)$ **do**
9 $\quad\quad$ find the common vertex x with the highest rank in $\bigcup_{i=1}^{t} \mathcal{L}^i_{out}(w)$ and $\bigcup_{i=1}^{t} \mathcal{L}^i_{in}(u_k)$ s.t. $w \rightsquigarrow_{\mathcal{I}_1} x$, $x \rightsquigarrow_{\mathcal{I}_2} u_k$, $\mathcal{I}_1, \mathcal{I}_2 \subseteq [t_s, t_e]$, and $ET(\mathcal{I}_1) \leq ST(\mathcal{I}_2)$;
10 $\quad\quad$ **if** $O(x) < O(w)$ **then** remove (w, t_s, t_e);
11 $\quad$ the process to clean $\bigcup_{i=1}^{t} \mathcal{L}^i_{out}(u_k)$ is similar to Lines 8-10;
12 **return** $\mathcal{L}^1, \cdots, \mathcal{L}^t$;

Extensions

Extension 2: We propose the inverted in-label set by extending the TBP-index to efficiently answer single-source reachability query (SSRQ).

> ***Inverted in-label set:*** *the inverted in-label set of a minimal TBP-index, denoted as* $\widehat{L}_{in}$*, is a data structure recording each index entry in* L_{in} *reversely, that is, for each* $(u, t_s, t_e) \in L_{in}(w)$*,* $\forall w \in U(G)$*,* $\widehat{L}_{in}$ *records* (w, t_s, t_e) *in* $\widehat{L}_{in}(u)$*. Accordingly, the entry* (w, t_s, t_e) *in* $\widehat{L}_{in}(u)$ *indicates that u reaches w within* $[t_s, t_e]$*.*

Answer SSRQ: With the above inverted in-label set $\widehat{L}_{in}$ and the TBP-index L, SSRQ can be efficiently answered by linearly scanning $\widehat{L}_{in}$ and L.

Extensions

Extension 3: To support fast earliest-arrival path queries (EAPQ), we propose the path-aware TBP-index by additionally recording the information of passing vertices.

Path-aware TBP-index: *The path-aware TBP-index, denoted by L^*, extends a minimal TBP-index L by recording the information of passing vertices in the paths.*

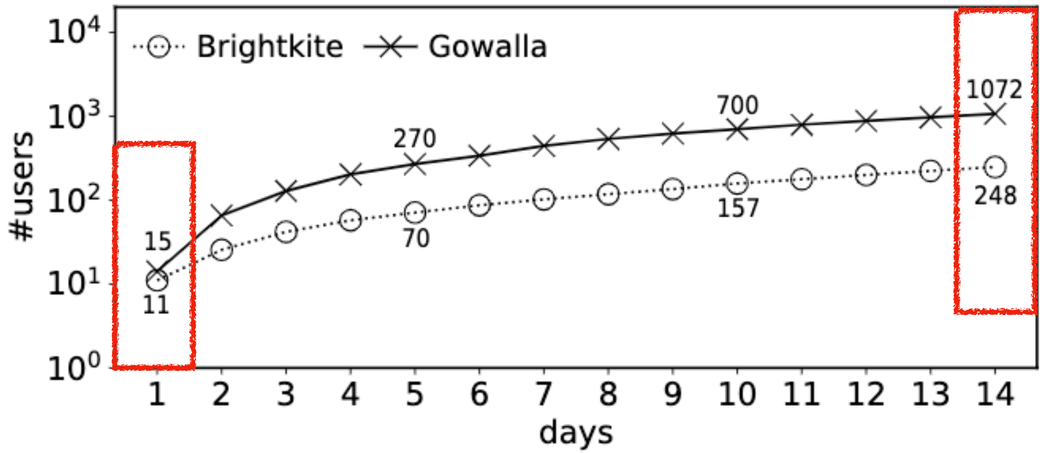
- *For $\forall u \in U(G)$, each index entry in $L^*_{out}(u)$ is a tuple $(w, t_s, t_e, \langle x, y, t_p \rangle)$ meaning that u reaches w within $[t_s, t_e]$, and the first time-overlapping wedge in the path is a wedge from u to x via y starting at t_s and ending at t_p .*
- *$\forall (w', t'_s, t'_e, \langle x', y', t'_p \rangle) \in L^*_{in}(u)$, it indicates that w' reaches u within $[t'_s, t'_e]$, and the last time-overlapping wedge in the path is from x' to u via y' starting at t'_p and ending at t'_e.*

Answer EAPQ: Using the path-aware TBP-index L^*, an earliest-arrival path P from u to w (two query vertices) can be efficiently retrieved in a recursive manner.

Performance Study

Case Studies

Case 1: we simulate the process of disease propagation using the temporal bipartite reachability. We can see that the number of potentially infected users continues to rise quickly day after day. For example, on Gowalla, 15 users are at risk of infection after 1 day, while this figure surges up to 1072 after 14 days.

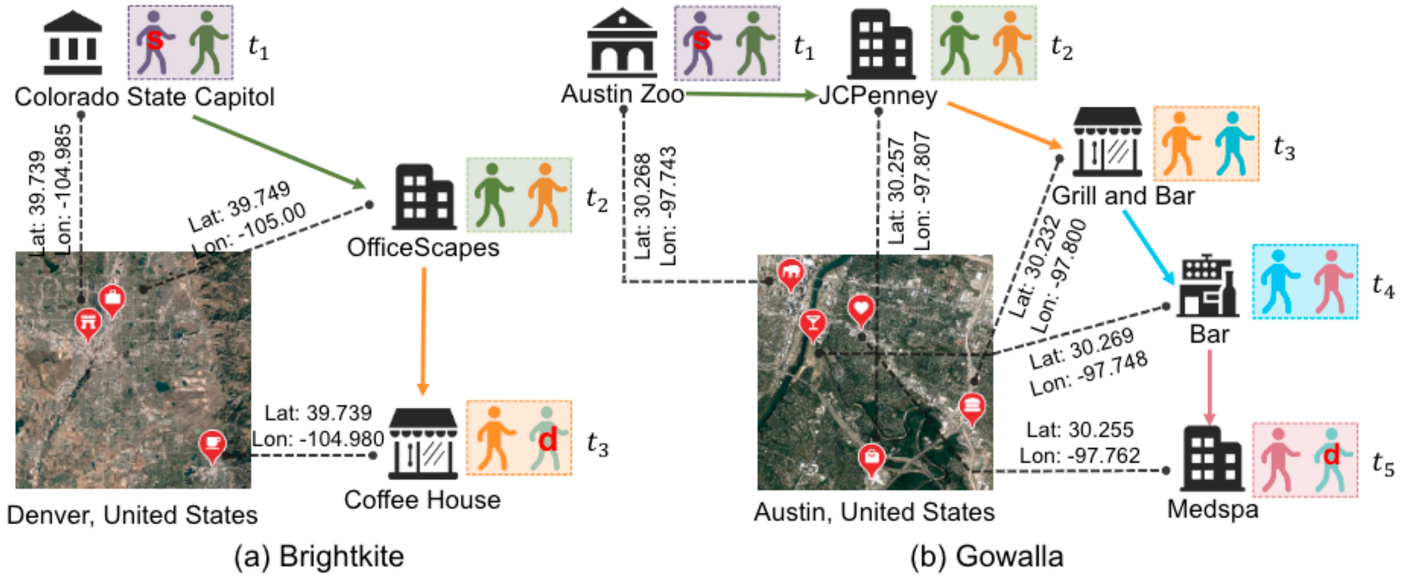


The number of potentially infected users

Our proposed TBP-index-based algorithm can efficiently identify the potential infected population. On Gowalla, our proposed algorithm spends merely 60s identifying the infected population for 10,000 infected sources, while the brute-force online algorithm needs more than 1.4 hours to do the same job.

Case Studies

Case 2: we show the potential of our proposed method on revealing the transmission chains.



The transmission chains

Datasets and Algorithms

Name	Dataset	$\|E(G)\|$	$\|U(G)\|$	$\|L(G)\|$
WQ	Wikiquote	549,210	21,607	92,924
WN	Wikinews	901,416	10,764	159,910
WB	Wikibooks	1,164,576	32,583	133,092
SO	Stackoverflow	1,301,942	545,195	96,678
LK	Linux-kernel	1,565,683	42,045	337,509
CU	Citeulike	2,411,820	153,277	731,769
BS	Bibsonomy	2,555,080	204,673	767,447
TW	Twitter	4,664,605	530,418	175,214
AM	Amazon	5,838,041	2,146,057	1,230,915
WT	Wiktionary	8,998,641	29,348	2,094,520
EP	Epinions	13,668,320	120,492	755,760
LF	Lastfm	19,150,868	992	1,084,620
IW	Itwiki	26,241,217	137,693	2,225,180
EF	Edit-frwiki	46,168,355	288,275	3,992,426
WP	Wikipedia	129,885,939	1,025,084	5,812,980
PM	PubMed	737,869,083	141,043	8,200,000

Index Construction Algorithms:

(1) TC-build: the TopChain algorithm, which is the state-of-the-art indexing-based algorithm to answer reachability queries on temporal unipartite graphs. The related index is called TC-index.
(2) TC-build-PL: the TopChain algorithm in parallel;
(3) TBP-build: the baseline algorithm to compute TBP-index;
(4) TBP-build*: the advanced algorithm to compute TBP-index;
(5) TBP-build*-PL: the algorithm that constructs a minimal TBP-index in parallel.

Query Algorithms:

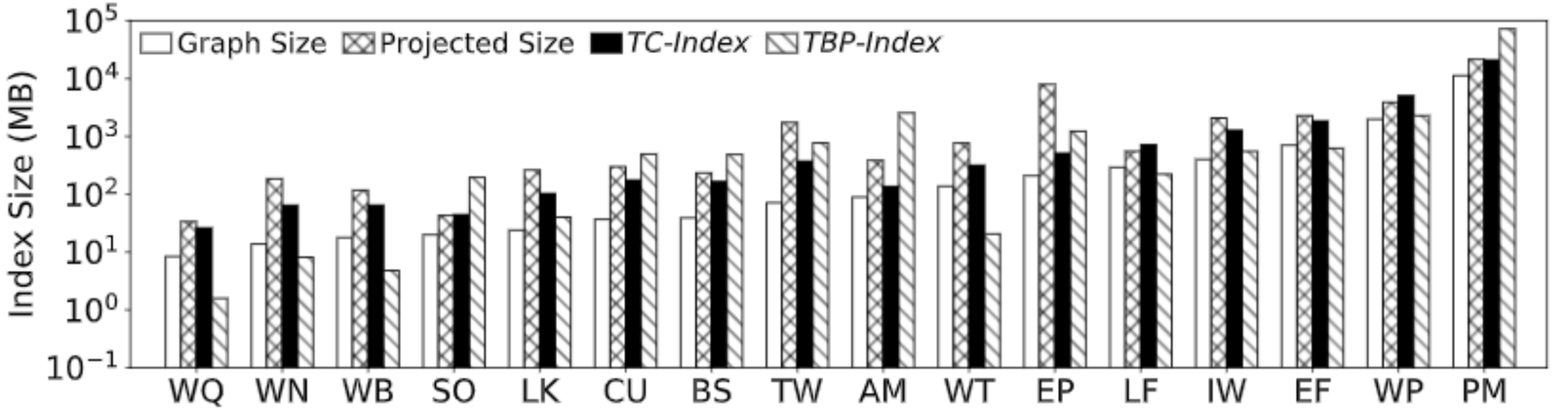
(1) OReach: the online query algorithm;
(2) OReach+: an optimized online query algorithm that bases on the direction-optimizing breadth-first search;
(3) TC-query: the query algorithm using the TC-index;
(4) TBP-query: the TBP-index-based algorithms that answers the single-pair reachability query, single-source reachability query and earliest-arrival path query.

Query Performance

Name	SPRQ - Answering Positive Queries				SPRQ - Answering Negative Queries				SSRQ		EAPQ
	OReach (ms)	OReach$^+$ (ms)	TC-query (μs)	TBP-query (μs)	OReach (ms)	OReach$^+$ (ms)	TC-query (μs)	TBP-query (μs)	TC-query (ms)	TBP-query (ms)	TBP-query (μs)
WQ	5.56	4.85	0.84	**0.18**	6.02	3.23	1.15	**0.20**	1.88	**0.03**	0.72
WN	9.60	4.05	3.41	**0.41**	9.34	6.52	2.53	**0.72**	1.07	**0.06**	1.08
WB	8.58	4.08	4.40	**0.30**	5.44	2.08	3.11	**0.31**	3.12	**0.02**	0.73
SO	11.68	0.91	9.12	**0.97**	8.57	1.75	1.57	**0.81**	45.32	**0.43**	2.69
LK	6.54	1.67	4.61	**0.85**	4.62	2.40	2.13	**1.91**	3.57	**0.25**	2.36
CU	20.24	2.92	24.62	**2.10**	35.40	21.20	8.77	**4.64**	28.78	**5.38**	5.76
BS	10.65	2.16	52.49	**1.93**	7.71	6.52	6.57	**3.82**	141.30	**4.08**	6.00
TW	79.48	9.32	83.59	**1.79**	53.19	32.34	3.88	**2.98**	118.71	**4.71**	5.76
AM	74.77	4.78	65.83	**2.24**	62.87	11.90	4.09	**2.45**	345.04	**19.52**	6.89
WT	249.00	90.84	8.49	**0.58**	88.63	61.43	3.71	**1.68**	2.44	**0.12**	1.73
EP	35.84	7.50	1269.41	**4.96**	165.42	37.66	90.74	**3.19**	976.89	**7.56**	8.19
LF	312.90	101.86	29.45	**7.87**	91.57	60.37	12.90	**8.68**	0.35	**0.34**	20.83
IW	510.00	119.34	51.00	**2.99**	234.76	152.12	6.26	**3.64**	12.93	**2.04**	10.76
EF	903.24	228.80	36.30	**2.43**	671.80	226.11	5.34	**2.60**	24.24	**4.00**	9.73
WP	25.60s	1.20s	19.32	**4.17**	65.30s	41.40s	5.96	**4.09**	276.62	**26.02**	17.40
PM	58.59s	1.48s	2827.75	**16.96**	172.0s	90.37s	1032.30	**140.25**	12360.0	**180.22**	207.09

1. OReach+ performs better than OReach, but it is still at least an order of magnitude slower than TBP-query and TC-query (index-based algorithms).
2. Our TBP-query is faster than TC-query with up to two orders (on positive cases) or one order (negative cases) of magnitude.
3. When answering SSRQ, TBP-query is faster than TC-query with up to two orders of magnitude.
4. TC-query does not support EAPQ, while our TBP-query can answer EAPQ with reasonable time

Index Construction



Index Size

1. The size of the projected graph is substantially larger than the original graph size.
2. TBP-index is smaller than TC-index (a partial index) on 9 out of 16 datasets.

Index Construction

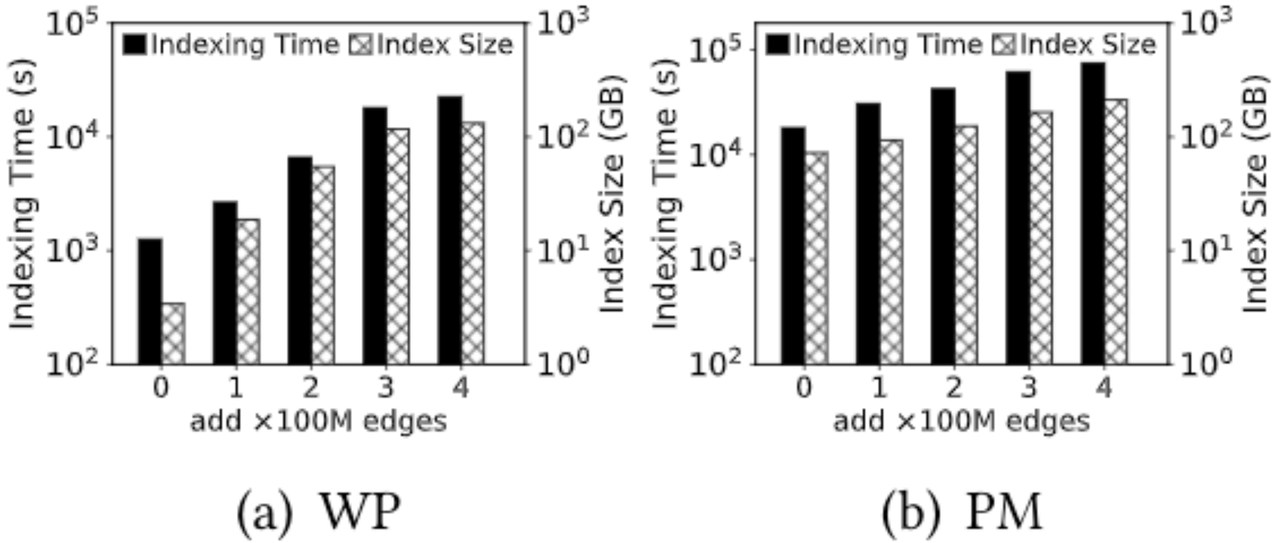
Name	Partial Index		Complete and Minimal Index (TBP-)			
	TC-build	TC-build-PL	build	build$^+$	build*	build*-PL
WQ	1.16	1.11	5.89h	2.61	0.41	0.13
WN	6.26	4.12	26.48h	9.84	4.21	0.38
WB	3.81	2.98	20.15h	7.38	1.36	0.21
SO	2.66	1.55	25.41h	184.58	51.88	6.94
LK	6.94	2.43	-	47.33	29.66	2.39
CU	14.03	4.67	-	1158.59	710.16	70.27
BS	10.09	4.76	-	534.20	447.13	49.02
TW	46.43	15.02	-	1207.85	869.89	112.00
AM	16.75	9.17	-	3160.54	2016.39	320.63
WT	27.65	23.76	-	202.84	88.5	5.00
EP	215.17	45.88	-	7451.13	5007.3	339.92
LF	36.19	10.55	-	5463.48	1226.39	70.59
IW	78.58	28.16	-	7031.39	2453.3	131.62
EF	110.54	38.11	-	4.48h	3875	221.00
WP	568.35	292.78	-	-	6.36h	1256.88
PM	1021.98	523.90	-	-	-	4.22h

Index Time

1. TBP-build and TBP-build* are slower than TC-index, which is expected as they both compute a complete and minimal index, and need additional time to check if an index entry is redundant.
2. TBP-build* can be accelerated by parallelization, making TBP-build*-PL finish the computation on graphs with hundreds of millions of edges within reasonable time.

Scalability

We randomly add edges into the two largest datasets to test the scalability of TBP-build*-PL. Accordingly, we can see that TBP-build*-PL can handle graphs at billion scale.



Scalability of TBP-build-PL on large-scale graphs*

Conclusion

Conclusion

- A Temporal Bipartite Reachability Model
- A 2-Hop Labeling Index-based Solution
- Efficient Index Construction Algorithms
- Extensive Experiments and Case Studies

Thanks