A Contraction Framework for Answering Reachability Queries on Temporal Bipartite Graphs

Lijun Sun[1], Junlong Liao[1], Ting Deng[1], Ping Lu[1], Richong Zhang[1],
Jianxin Li[1], Xiangping Huang[2], Zhongyi Liu[2]
[1]Beihang University [2]TravelSky Technology Limited
{sunlijun1, liaojunlong, dengting, luping, zhangrichong, lijx}@buaa.edu.cn, {xphuang, liuzy}@travelsky.com.cn

***Abstract*—In this paper, we study the span-reachability problem for temporal bipartite graphs, which model interactions between two distinct entity types over time, and can be used to control disease, trace metabolic pathways and detect fake news. Existing algorithms rely on index construction to answer such problem, but often struggle with issues like size inflation or inefficient parallel processing. To address these, we propose a contraction-based indexing framework CBSR. This framework first transforms a given temporal bipartite graph into a directed acyclic graph (DAG) preserving the reachability relation between vertices, and then partitions the DAG into multiple fragments to reduce used space and dependency among vertices. Finally, it constructs indices for these fragments using a contraction strategy. Using real-life and synthetic data, we experimentally show that CBSR outperforms the state-of-the-art algorithms by on average 1361.8 times in constructing indices, and 2.96 times in querying.**

I. Introduction

Bipartite graphs (bigraphs) are specialized graphs that partition vertices into two disjoint sets with edges solely connecting vertices from different sets. Such structure can represent interactions between entities of distinct types, and may carry temporal information about interactions. Bipartite graphs are applied in various domains including disease prevention [61], [49], [36], recommendation systems [34], [7], [55], social networks analysis [8], [4] and user intent analysis [35], [19].

Various reachability problems have been studied for temporal graphs, including temporal reachability [10], [13], [45], restless temporal reachability [11], [46], [18], shortest path distance [57], [56], [30] and span-reachability query [13], [53], [58], [65]. We focus on the span-reachability query (*a.k.a.* temporal reachability query in [65]), which is to decide whether there is a temporal path between two vertices within a time interval. Such problem has been applied to control disease outbreaks [20], [25], [66], [29], [13], trace metabolic pathways [41], [13], [12] and detect fake news [38], [44].

There exist two approaches to answer the span-reachability queries. (1) The first approach projects bipartite temporal graphs to unipartite graphs, and applies existing techniques to conduct the queries. However, this method is inefficient due to the size inflation and information loss [13], since (a) the projection may create large cliques, representing that multiple top vertices connect to the same lower vertex at the same time, enabling them to reach one another; and (b) some vertices may be removed during projections, resulting in information loss. (2) The second method conducts the queries on temporal bipartite graphs directly. For example, the algorithm in [13] extends the 2-hop technique to temporal bipartite graphs, and performs queries on such graphs directly. However, it cannot yield correct answers. Consider the following example.

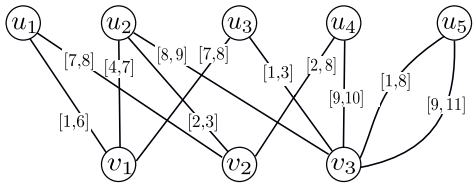

Fig. 1: A temporal bipartite graph G

Example 1: Figure 1 illustrates a people-location graph, where vertices in the top layer represent people and vertices in the bottom layer denote locations. Intervals $[t_1, t_2]$ indicate the presence of people at specific locations during $[t_1, t_2]$, where t_1 and t_2 indicate the arriving time and leaving time, respectively.

Consider a disease control scenario. When an individual is infected, it is crucial to identify those at risk of infection. Assume that u_2 becomes infected at timestamp 2. Then u_4 should be regarded as a potential infector, since u_2 meets u_4 during time interval $[2, 3]$ at location v_2. Subsequently, during interval $[7, 8]$, u_4 meets u_1 at v_2, putting u_1 at risk as well. However, existing algorithms TBP [13] and WTB [38] cannot identify u_1 as a potential infector since they define "contact" as the union of time intervals of the two edges, which may miss some results. According to definitions in TBP [13] and WTB [38], the "contact" between u_2 and u_4 occurs during $[2, 8]$, while the "contact" between u_4 and u_1 also occurs during $[2, 8]$, which cannot follow the "contact" between u_2 and u_4, since the starting time of the "contact" between u_4 and u_1 (*i.e.,* 2) is not larger than the ending time of the "contact" between u_2 and u_3 (*i.e.,* 8). So, u_1 cannot be infected by u_2 via u_4 based on TBP and WTB. Such coarse-grained definition of "contact" leads to information loss, resulting in query outcomes that deviate from real-world scenarios. □

The example illustrates that existing methods for span-reachability lack sufficient "resolution", which motivates us to develop a more refined algorithm to address the problem.

Contributions. We propose a new method to answer the span-reachability query on temporal bipartite graphs, which can not only avoid size inflation and information loss, but also

be paralleled easily. Our contributions are as follows:

(1) Graph transformation (Section V). We propose a transformation method converting temporal bipartite graphs into directed acyclic graphs (DAGs). To avoid size inflation, we introduce the concept of snapshots to compactly capture reachability information, such that the DAG maintains all critical information to answer the span-reachability query.

(2) Index construction (Section VI). We introduce a contraction-based indexing algorithm CBSR. It first partitions a DAG into multiple fragments, next constructs indices for each fragment, then connects these fragments based on the reachability relations of vertices, and finally applies a graph contraction strategy to construct indices for these connections.

(3) Query algorithm (Section VII). We propose a two-stage query algorithm SRPQ for span-reachability. It first checks whether the source and target vertices are in the same fragment. If so, it applies existing query algorithms [13] on intra-fragment indices; otherwise, it conducts the query on inter-fragment indices. The complexity of SRPQ matches that of existing algorithms [13], due to the contraction-based indices.

(4) Experimental study (Section VIII). Using real-life datasets, we experimentally find the following. (1) Our graph transformation method can turn bipartite graphs into DAGs that preserve the reachability relations among vertices and are on average 29.5% the size of original graphs. (2) CBSR is efficient. On a graph with 1.5M vertices and 234M edges, CBSR constructs the index in less than 300s, while the state-of-the-art algorithm TBP [13] cannot process the graph. (3) On average, SRPQ is $3\times$ faster than the existing algorithm TopChain [57] on reachable queires, and has similar performance on unreachable queries.

Organization. This paper is organized as follows. We introduce the necessary notations in Section III and give an overview of the solution in Section IV. Then we present algorithms for graph transformation, index construction and querying in Sections V, VI and VII, respectively. Extensive experiments are conducted and introduced in Section VIII.

II. Related work

We categorize the related work as follows.

General graph. The reachability problem in general graphs has been extensively studied. The problem is to determine whether there is a path between two nodes in a graph. Two well-known algorithms for this problem are the depth-first search (DFS) algorithm and the breadth-first search (BFS) algorithm. However, these methods cannot handle large-scale graphs [33]. Many indexing methods are proposed to accelerate the computation: (1) tree-cover based methods (*e.g.,* path-tree [31], gripp [47] and grail [59]), which construct (a) a tree containing most part of reachability information, and (b) a supplementary index table storing the remaining information; (2) hop-based methods (*e.g.,* 2-hop [15], 3-hop [32], HHL [3], O'Reach [27], BL [60] and TF-Label [14]), which pick a set of nodes as the landmarks and compute the reachability queries using the landmarks as bridges; and (3) Pruned-based methods (*e.g.,* IP [52], BFL [42], PReaCH [37] and ELF [43]), which compute some conditions for vertices, and filter out unreachable queries. Contraction-based methods have been developed to convert a graph into a DAG, where each vertex represents a strongly connected component of the graphs [67]. See [64], [62] for more discussion.

Closer to this work are [63], [28], which support reachability queries on DAGs. MGTag [63] is a graph labeling method that recursively partitions graphs into multiple fragments and computes four-dimensional labels for indexing. TDS [28] improves MGTag with more reasonable dimension selection.

Discussion. This work differs from prior work as follows. (1) We target the reachability problem on temporal bipartite graphs, which is more challenging than the problem studied in [67], [63], [28]. (2) To handle large-scale graphs, we develop a contraction-based strategy to reduce the sizes of graphs and accelerate query processing, which is not studied in [63], [28]. (3) Using contracted graphs, we can exploit existing algorithms to conduct queries, rather than developing a new one.

Temporal graphs. The reachability problem has been studied on temporal graphs [40], [51], [57], [65], [53], [54]. ReachGraph transforms temporal graphs into DAGs by materializing connection components at each timestamp [40]. Timetable Labeling (TTL) extends the hop-based methods to temporal graphs and adopts index partitioning strategy to handle consecutive time intervals [51]. TopChain transforms temporal graphs into a DAG, decomposes the DAG into a set of chains, and extends the hop-based methods to construct indices [57]. Temporal Vertex Labeling (TVL) proposes TVL indices for the reachability queries [65]. Time Interval Labeling (TILL-Index) [53], [54] extends the 2-hop method with timestamps to answer the span-reachability problem in temporal graphs.

Closer to this work is [13], [38], [45]. Algorithm TBP targets the span-reachability problem (*i.e.,* the single-pair reachability problem in [13]) on temporal bipartite graphs, and devises TBP-indices to accelerate query processing [13]. Algorithm WTB success the definition of temporal path of TBP, but its query is not constrained by the time interval, which aims to answer global approximate reachability problem [38]. Algorithm RQ adopts a partition-based strategy to construct indices for reachability queries within budget [45].

Discussion. In contrast to prior work, (1) we adopt a different definition of temporal paths, which can answer the reachability problem more accurate than [13], [38], [45] (see Section III), (2) We convert temporal bipartite graphs into DAGs, and apply existing algorithms for reachability on generated graphs, rather than developing new ones as [13], [45]. (3) We use bottom vertices to construct vertices, and connect these vertices based on edges of top vertices, while TopChain links vertices in DAGs using both timestamps and edges in the original temporal graphs, which results in redundant edges [57]. (4) We compute equivalence relations to reduce the size of graphs, and exploit the trie data structure

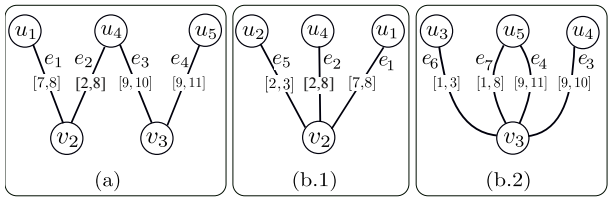

Fig. 2: Consecutive connections in a temporal path.

to reduce used memory, which is not studied in [13], [45].

III. Preliminary

Assume a countably infinite set Ω of timestamps, which are ordered *w.r.t.* a discrete order $\leq$. A time interval is a range $[t_1, t_2]$ of timestamps with $t_1, t_2 \in \Omega$ and $t_1 \leq t_2$. Let $\mathcal{T} \subseteq \Omega \times \Omega$ be the set of all time intervals. Intervals $I_1 = [t_1, t_2]$ and $I_2 = [t_3, t_4]$, are *consecutive* if the ending time of interval I_1 is the same as the starting time of interval I_2, *i.e.,* $t_2 = t_3$. They are *overlapped* if $\min(t_2, t_4) \geq \max(t_1, t_3)$. To simplify the presentation, given a time interval $I = [t_1, t_2]$ denote by $I.t_s$ and $I.t_e$ the starting time and the ending time of I, respectively, *i.e.,* $I.t_s = t_1$ and $I.t_e = t_2$.

Temporal bigraph. An undirected temporal bigraph is $G{=}(V, E)$, where (1) V is a finite set of vertices and partitioned into two disjoint sets U and L, *i.e.,* $V{=}U \cup L$ and $U \cap L = \emptyset$; vertices in U and L are called top vertices and bottom vertices, respectively; and (2) $E \subseteq U \times L \times \Omega \times \Omega$ is a finite set of temporal edges, where each edge $e = (u, v, t_s, t_e)$ in E connects a top vertex u in U to a bottom vertex v in L during the time interval $[t_s, t_e]$. We also use $e.t_s$ and $e.t_e$ to denote the *starting time* and *ending time* of e, respectively.

For vertex u, denote by $E(u)$ the set of edges incident to u, and d_u the degree of u, *i.e.,* $d_u = |E(u)|$.

Connection. Given a temporal bigraph $G = (U{\cup}L, E)$ and top vertices $u, u' \in U$, there is a connection between u and u' if they both connect to a bottom vertex at some timestamp; *e.g.,* two persons visit the same location at the same time.

More specifically, a *connection* between two top vertices u and u' at bottom vertex v, denoted by $\langle u, e_1, v, e_2, u' \rangle$, consists of two edges $e_1 = (u, v, t_s^1, t_e^1)$ and $e_2 = (u', v, t_s^2, t_e^2)$ in E with overlaped time intervals, *i.e.,* $\min(t_e^1, t_e^2) {\geq} \max(t_s^1, t_s^2)$. The time interval of the connection, denoted by $\Theta(e_1, e_2)$, is the *overlap* of the time intervals in edges e_1 and e_2, *i.e.,* $\Theta(e_1, e_2){=}[\max(t_s^1, t_s^2), \min(t_e^1, t_e^2)]$.

Note that connections $\langle u, e_1, v, e_2, u' \rangle$ and $\langle u', e_2, v, e_1, u \rangle$ are considered the same regardless of direction.

Example 2: Consider bigraph G in Fig. 1, Connection $c_1 = \langle u_1, e_1, v_2, e_2, u_4 \rangle$ is constructed from edges $e_1{=}(u_1, v_2, 7, 8)$ and $e_2{=}(u_4, v_2, 2, 8)$, and its interval is $\Theta(e_1, e_2){=}[7, 8]$, since the overlap of intervals $[7, 8]$ in e_1 and $[2, 8]$ in e_2 is $[7, 8]$. □

Temporal Path. Given a temporal graph $G{=}(U{\cup}L, E)$ and two top vertices u and u' in U, a *temporal path* ρ from u to u' of length $2k$ (*i.e.,* consists of $2k$ edges) is a sequence of vertices $\rho = u_1 v_1 u_2 v_2 \dots u_k v_k u_{k+1}$ with $u{=}u_1$ and $u' = u_{k+1}$, where for each $i \in [1, k{-}1]$, u_i and u_{i+1} are the same top vertex or link to the same bottom vertex at the same time; more specifically, there exist two consecutive connections $c_1 = \langle u_i, e_1^i, v_i, e_2^i, u_{i+1} \rangle$ and $c_2 = \langle u_{i+1}, e_1^{i+1}, v_{i+1}, e_2^{i+1}, u_{i+2} \rangle$ in G such that: (1) the starting time of c_1 and c_2 increase, *i.e.,* $\Theta(e_1^i, e_2^i).t_s \leq \Theta(e_1^{i+1}, e_2^{i+1}).t_s$; and (2) timestamps of adjacent edges e_2^i and e_1^{i+1} of the common top vertex u_{i+1} increases, *i.e.,* (a) if $v_i \neq v_{i+1}$, then $e_2^i.t_e \leq e_1^{i+1}.t_s$(Fig. 2(a)), (b) if $v_i = v_{i+1}$, then $e_2^i = e_1^{i+1}$ (*i.e.,* e_2^i and e_1^{i+1} are the same edge) (Fig 2(b.1)) or $e_2^i.t_e \leq e_1^{i+1}.t_s$(Fig 2(b.2)).

Intuitively, when $v_i \neq v_{i+1}$, top vertex u_{i+1} moves from one bottom vertex v_i to another bottom vertex v_{i+1}, and the timestamp increases, *i.e.,* $e_2^i.t_e \leq e_1^{i+1}.t_s$; and when $v_i = v_{i+1}$, top vertex u_{i+1} "stays at" the same bottom vertex v_i; that is, either e_2^i and e_1^{i+1} are the same edge, or u_{i+1} "leaves" and "reenters" the bottom vertex v_i.

Example 3: Fig. 2 shows three temporal paths of length 4, each consisting of two connections, all from graph G in Fig. 1.

(a) Temporal path $\rho_1{=}u_1 v_2 u_4 v_3 u_5$ consists of connections c_1 and c_2 (*i.e.,* Case (a) in Fig. 2.), where (i) $c_1{=}\langle u_1, e_1, v_2, e_2, u_4 \rangle$ is defined by edges $e_1{=}(u_1, v_2, 7, 8)$ and $e_2{=}(u_4, v_2, 2, 8)$ with $\Theta(e_1, e_2){=}[7, 8]$, and (ii) $c_2{=}\langle u_4, e_3, v_3, e_4, u_5 \rangle$ is composed by edges $e_3{=}(u_4, v_3, 9, 10)$ and $e_4{=}(u_5, v_3, 9, 11)$ with $\Theta(e_3, e_4){=}[9, 10]$. Starting time of c_1 and c_2 increase, *i.e.,* $\Theta(e_1, e_2).t_s{=}7 \leq \Theta(e_3, e_4).t_s{=}9$, and u_4 links to two distinct bottom vertices v_2 and v_3, and $e_2.t_e \leq e_3.t_s$ (*i.e.,* $8 < 9$).

(b) Temporal path $\rho_2{=}u_2 v_2 u_4 v_2 u_1$ consists of connections c_3 and c_1 (*i.e.,* Case (b.1) in Fig. 2.), where $c_3{=}\langle u_2, e_5, v_2, e_6, u_4 \rangle$ is composed by edges $e_5{=}(u_2, v_2, 2, 3)$ and $e_2 = (u_4, v_2, 2, 8)$ with $\Theta(e_5, e_2){=}[2, 3]$. Starting time of c_3 and c_1 increase, *i.e.,* $\Theta(e_5, e_6).t_s{=}2 \leq \Theta(e_1, e_2).t_s{=}7$. Both c_3 and c_1 have e_2.

(c) Temporal path $\rho_3{=}u_3 v_3 u_5 v_3 u_5 v_3 u_4$ consists of connections c_4 and c_2 (*i.e.,* Case (b.2) in Fig. 2.), where $c_4 = \langle u_3, e_6, v_3, e_7, u_5 \rangle$ consists of edges $e_6{=}(u_3, v_3, 1, 3)$ and $e_7{=}(u_5, v_3, 1, 8)$ with $\Theta(e_6, e_7){=}[1, 3]$. Starting time of c_4 and c_2 increase, *i.e.,* $\Theta(e_6, e_7).t_s{=}1 \leq \Theta(e_3, e_4).t_s = 9$, c_4 and c_2 link to the same vertex v_3, and $e_7.t_e{\leq}e_4.t_s$ (*i.e.,* $8{\leq}9$). □

Span-Reachability. We say that one top vertex u can reach another top vertex u' in G during time interval I if there exists a temporal path ρ from u to u' such that (1) the ending time $\Theta(e_1^1, e_2^1).t_e$ of the first connection $\langle u, e_1^1, v_1, e_2^1, u_2 \rangle$ in ρ is not smaller than $I.t_s$, and (2) the starting time $\Theta(e_1^k, e_2^k).t_s$ of the last connection $\langle u_{k-1}, e_1^k, v_k, e_2^k, u' \rangle$ in ρ is not larger than $I.t_e$, We use the ending time of the first connection and the starting time of the last connection to ensure that all connections are within interval I, which results in more temporal paths between two vertices, and more precise contact tracing.

Example 4: Consider graph G in Figure 1. Top vertex u_1 can reach u_5 during interval $[8, 9]$, since u_1 first reaches u_4 at bottom vertex v_2 at timestamp 8, and then u_4 reaches u_5 at bottom vertex v_3 at timestamp 9, which can be characterized by the temporal path $\rho_1{=}u_1 v_2 u_4 v_3 u_5$ given in Example 3. Recall that ρ_1 consists of two consecutive connections $c_1{=}\langle u_1, e_1, v_2,$

$e_2, u_4\rangle$ and c_2=$\langle u_4, e_3, v_3, e_4, u_5\rangle$, with $\Theta(e_1, e_2) = [7, 8]$ and $\Theta(e_3, e_4).t_s = [9, 10]$. Then u_1 can reach u_5 during $[8, 9]$ since $\Theta(e_1, e_2).t_e = 8 \leq 8$ and $\Theta(e_3, e_4).t_s = 9 \leq 9$. Although the starting time of edge e_2 is 2 and the ending time of e_3 is 10, which are not in the interval $[8, 9]$, we can confirm that u_1 can reach u_5 during interval $[8, 9]$ by inspecting the starting time and ending time of the overlap time of these edges. □

Problem Statement. We study *the span-reachability problem*, denoted by SRP, which is stated as follows.

- *Input*: A temporal bigraph G, two top vertices u and u', and a time interval I
- *Question*: Whether u can reach u' in G in time interval I?

Denote by $Q(u, u', G, I)$ the span-reachability query with u as the *source vertex* and u' as the *target vertex*.

Remark. (1) Our definition of temporal paths differs from the one presented in [13], [38]. More specifically, the time interval of a connection $c = \langle u, v, e_1, e_2, u'\rangle$ is the overlap of the time intervals of e_1 and e_2, rather than their union as in [13], [38]. Then the time interval of c is a subset of the time intervals of e_1 and e_2, allowing the same edge to appear multiple times in a temporal path. In contrast, under the union-based semantics as depicted in [13], [38], each edge can appear at most once in a temporal path. Such difference leads to more temporal paths in our SRP setting than in the problem studied in [13], [38].

Example 5: Consider path ρ_2=$u_2v_2u_4v_2u_1$ from Example 3.

(I) Path ρ_2 is not a valid path under the definitions of [13], [38]. (a) Subpath $u_2v_2u_4$ consists of two edges $e_5 = (u_2, v_2, 2, 3)$ and e_2=$(u_4, v_2, 2, 8)$, forming a wedge $\mathcal{W}_1 = (e_5, e_2)$ with starting time 2 (*i.e.,* $\min(e_5.t_s, e_2.t_s)$=2) and ending time 8 (*i.e.,* $\max(e_5.t_e, e_2.t_e)$=8). (b) Similarly, path $u_4v_2u_1$ forms another wedge $\mathcal{W}_2$=$((u_4, v_2, 2, 8), (u_1, v_2, 7, 8))$, whose starting time and ending time are 2 and 8, respectively. (c) Wedges $\mathcal{W}_1$ and $\mathcal{W}_2$ cannot be concatenated to form a temporal path from u_2 to u_1, since the starting time of $\mathcal{W}_2$ is 2, which is earlier than the ending time of $\mathcal{W}_1$ (*i.e.,* 8).

(II) However, path ρ_2 is valid in disease tracing. After being infected by u_2 during interval $[2, 3]$, u_4 remains at location v_2, and infects u_1 during $[7, 8]$, even though u_2 has left v_2 at timestamp 3. Under our definition, such indirect propagation is allowed, and thus u_2 can reach u_1 in interval $[2, 8]$.

Note that edge $(u_4, v_2, 2, 8)$ appears twice in path ρ_2, which is not allowed in the definition of [13], [38]. □

(2) Methods developed in this paper can be extended to solve the following problems [13]: (a) the single-source reachability query, which is to compute the set of all top vertices that u can reach within time interval I; and (b) the earliest-arrival problem, which is to compute the minimum ending time among all temporal paths from u to w within time interval I.

The notations used in the paper are listed in Table I.

TABLE I: Notations

Notations	Definitions
$G = (U \cup L, E)$	an undirected temporal bigraph
$Q(u, u', G, I)$	the span-reachability query
$\mathcal{G}_G$	the traject graph of temporal graph G
$\mathcal{T}_G$	the set of timestamps appearing G
$\langle u, e_1, v, e_2, u'\rangle$	connection between top vertices u and u' at bottom vertex v
$\Theta(e_1, e_2)$	common time interval of e_1 and e_2
$[t_1, t_2].t_s$/$[t_1, t_2].t_e$	starting/ending time of $[t_1, t_2]$
$[t_1, t_2]_v$	A snapshot of a bottom vertice v
$\mathcal{S}(G)$/$\mathcal{S}(u)$	the set of all snapshots constructed in graph G/from top vertex u

IV. Solution Overview

In this section, we overview a contraction-based framework for the SRP problem, denoted by CBSR.

Architecture. Given a temporal bigraph G, CBSR first converts it into a directed acyclic graph (DAG) $\mathcal{G}_G$. It then partitions $\mathcal{G}_G$ into multiple fragments, and constructs both intra-fragment and inter-fragment indices for $\mathcal{G}_G$. These indices are used to efficiently check whether two vertices are connected .

Step 1: Graph transformation. Given a temporal bigraph G, CBSR first constructs a traject graph $\mathcal{G}_G$ that is a DAG, to represent the reachability relation in G. Such transformation reduces the span-reachability problem in G to the standard reachability problem in the traject graph $\mathcal{G}_G$. Most existing approaches for the reachability problem on graphs convert the graphs into a DAG, where each vertex represents a strongly connected component of G (*e.g.,* [67]). Different from these approaches, CBSR partitions time intervals on edges to form the DAGs and optimize the reachability analysis (see below).

Step 2: partition and indexing. To handle large-scale graphs, CBSR first partitions $\mathcal{G}_G$ into fragments, and then constructs intra-fragement and inter-fragment indices to answer span-reachability queries. For intra-fragment indexing, CBSR leverages existing algorithms (*e.g.,* [31], [27]). For inter-fragment indices, CBSR applies a graph contraction strategy to reduce graphs [23], and computes equivalent sets for indexing.

Step 3: Answering reachability queries on DAG. We develop a query algorithm SRPQ for SRP. Given two top vertices u_1 and u_2 in G, it first identifies vertices w_1 and w_2 in $\mathcal{G}_G$ that are derived from u_1 and u_2, respectively. It then checks whether w_1 and w_2 belong to the same fragment. If so, intra-fragment indices are used to answer the query; otherwise inter-fragment indices is to determine whether w_1 can reach w_2.

Properties. CBSR has the following properties.

(1) The traject graph $\mathcal{G}_G$ is only marginally larger than the original graph G, and it avoids the size inflation commonly seen in the projection-based methods [65], [57] (Section V).

(2) We eliminate the global order dependency presented in [13] by utilizing both intra-fragment and inter-fragment indices, which can be effectively parallelized. No dependency exists in the construction of these indices (Section VI).

(3) For large graphs on which existing methods in [13], [38] fail to build indices due to memory limits, CBSR can still construct both intra-fragment and inter-fragment indices. By partitioning the graph into multiple fragments, CBSR avoids building indices for vertices far from each other, thereby reducing memory usage. Inter-fragment indices are then used to check reachability between such vertices, to strike a balance between query efficiency and space consumption (See Sections VII).

V. Graph Transformation

We transform the temporal bipartite graph G into a DAG $\mathcal{G}_G$, preserving the reachability relations between vertices. The DAG is constructed based on top vertices, capturing their "movements" across different bottom vertices.

Challenge. To construct a directed acyclic graph (DAG) from a temporal bigraph, we must address the following challenges: the temporal bigraph is undirected and may contain timestamps and cycles (see Figure 1), how can we construct a DAG from such an undirected graph, while preserving the reachability relationships between top vertices?

To address these challenges, we proposed the following strategies. (1) We introduce a data structure called snapshots, to capture time intervals during which a top vertex is linked to a bottom vertex. (2) These snapshots are sequentially ordered based on their timestamps, and linked according to the adjacent edges of top vertices. This ensures that the generated graph is a DAG. As a result, the reachability relation between top vertices can be determined by examining snapshots in the DAG. Intuitively, if a top vertex u can reach another top vertex v in the original graph G, this reachability relation can be confirmed by tracing the u's path through the DAG.

Snapshots. We start with the definition of snapshots. Given a temporal bigraph $G{=}(U \cup L, E)$, let $\mathcal{T}_G$ be the set of timestamps appearing in G. For any two timestamps t_1 and t_2 in $\mathcal{T}_G$, a *snapshot* of a bottom vertex v over the interval $[t_1, t_2]$, denoted by $[t_1, t_2]_v$, indicates that there exist two top vertices in U connected to v throughout $[t_1, t_2]$. Intuitively, $[t_1, t_2]_v$ is a snapshot if there exists a connection $\langle u, e_1, v, e_2, u'\rangle$ such that its time interval $\Theta(e_1, e_2)$ contains $[t_1, t_2]$. We call u and u' the top vertices of snapshot $[t_1, t_2]_v$.

Construction. Given a temporal bigraph G, we can construct snapshots from each pair of edges incident to a bottom vertex. However, this method may introduce redundant edges in the resulting DAG, as multiple top vertices may simultaneously connect to the same bottom vertex. To address this, we construct snapshots based on the timestamps of edges incident to each bottom vertex v. Let $t_1{<}t_2{<}\ldots{<}t_n$ be the sorted timestamps of such edges. For each consecutive pair of t_i and t_{i+1}, we generate *candidate* snapshots $[t_i, t_i]_v$, $[t_i{+}1, t_{i+1}{-}1]$ and $[t_{i+1}, t_{i+1}]$ when $t_{i+1}{-}t_i > 1$, and $[t_i, t_i]_v$ and $[t_{i+1}, t_{i+1}]$ when $t_{i+1} = t_i{+}1$. We next remove useless snapshots $[t', t'']_v$ during which there exists no two top vertices both incident to v, *i.e.,* there exists no connection such that its time interval contains $[t', t'']$. For each remaining snapshot $s{=}[t_i, t_j]_v$, we denote its starting and ending time by $s.t_s = t_i$ and $s.t_e = t_j$.

Denote by $\mathcal{S}(G)$ the set of all snapshots constructed above from graph G. Computing $\mathcal{S}(G)$ takes $O(|E| \cdot \log d_{\mathsf{max}})$ time, where d_{max} is the maximum degree in G, since for each bottom vertex v, it takes $O(d_v \log d_{\mathsf{max}})$ time to sort timestamps, where d_v is the degree of v in G.

Example 6: Consider graph G in Figure 1. For v_2, the sorted timestamps are $2{<}3{<}7{<}8$, yielding candidate snapshots $[2, 2]_{v_2}, [3, 3]_{v_2}$, $[4, 6]_{v_2}$, $[7, 7]_{v_2}$ and $[8, 8]_{v_2}$. For connections c_1 and c_3 in Example 3, since they are constructed from edges incident to bottom vertex v_2, we have snapshots $[2, 2]_{v_2}$ and $[3, 3]_{v_2}$ from connection c_3, and $[7, 7]_{v_2}$ and $[8, 8]_{v_2}$ from connection c_1. Similarly, we get snapshots $[4, 4]_{v_1}$, $[5, 5]_{v_1}$, $[6, 6]_{v_1}$ and $[7, 7]_{v_1}$ from bottom vertex v_1, and $[1, 1]_{v_3}$, $[2, 2]_{v_3}$, $[3, 3]_{v_3}$, $[8, 8]_{v_3}$, $[9, 9]_{v_3}$ and $[10, 10]_{v_3}$ from bottom vertex v_3.

A snapshot may be constructed from multiple connections, *e.g.,* $[9, 9]_{v_3}$ can be constructed from either (a) $c_5{=}\langle u_2, e_8, v_3, e_4, u_5\rangle$ with $e_8{=}(u_2, v_3, 8, 9)$ and $e_4{=}(u_5, v_3, 9, 11)$, or (b) $c_6 = \langle u_2, e_8, v_3, e_3, u_4\rangle$ with $e_3 = (u_4, v_3, 9, 10)$. □

Note that a connection can be split into multiple snapshots, to reduce the size of the DAG $\mathcal{G}_G$.

Example 7: Consider a graph G with k top vertices $u_t^1, u_t^2, \ldots, u_t^k$, one bottom vertex v and k edges $e_t^1 = (u_t^1, v, 1, k+1)$, $e_t^2 = (u', v, 2, k+2)$, …, and $e_t^k = (u_t^k, v, k, 2k)$. We have $\frac{k(k-1)}{2}$ connections: $c = \langle u_t^i, e_t^i, v, e_t^j, u_t^j\rangle$ with $i < j$ and $\Theta(e_t^i, e_t^j){=}[j, i+k]$. If we construct the DAG using connections, then when k is large, resulting DAG would be large.

We can construct $2k$ snapshots $[l, l{+}1]_v$ ($l \in [1, 2k{-}1]$), and link these snapshots to represent reachability relation among these vertices, leading to smaller DAGs (see below). □

Traject Graphs. For a temporal bigraph G, we define its traject graph $\mathcal{G}_G$ using snapshots in $\mathcal{S}(G)$ as vertices. Intuitively, edges in $\mathcal{G}_G$ represent the movements of top vertices from one bottom vertex to another bottom vertex.

Given a temporal bigraph $G{=}(U{\cup}L, E)$ and its snapshot set $\mathcal{S}(G)$, the traject graph of G is $\mathcal{G}_G = (\mathcal{V}, \mathcal{E})$, with vertice set $\mathcal{V} = \mathcal{S}(G)$. Edges in $\mathcal{E}$ connect two snapshots both constructed from at least one edge incident to the same top vertex.

More specifically, for each top vertex u in G, let $\mathcal{S}(u) {=}\{[t_s^1, t_e^1]_{v_1}, \ldots, [t_s^k, t_e^k]_{v_k}\}$ be the set of snapshots constructed from at least one edge incident to u. A snapshot $s_i{=}[t_s^i, t_e^i]_{v_i}$ has an edge to $s_j{=}[t_s^j, t_e^j]_{v_j}$, if either (1) $v_i = v_j$, $t_e^i \le t_s^j$, and there exists an edge $e{=}(u, v_i, t_s, t_e)$ in G such that its time interval "cover" the time intervals of both s_i and s_j (*i.e.,* $t_s{\le}t_s^i$ and $t_e^j{\le}t_e$); this represent a top vertex keeps linking to a bottom vertex, although other vertices leave; or (2) $v_i{\neq}v_j$, and there exist two edges $e'{=}(u, v_i, t_s^1, t_e^1)$ and $e'' = (u, v_j, t_s^2, t_e^2)$ incident to top vertex u such that $t_e^1 = t_e^i$ and $t_s^2 = t_s^j$, that is, some top vertex leaves bottom vertex v_i to bottom vertex v_j. Intuitively, by the definition of temporal path, when these conditions hold, there exists two consecutive connections forming a temporal path uv_iuv_ju in G.

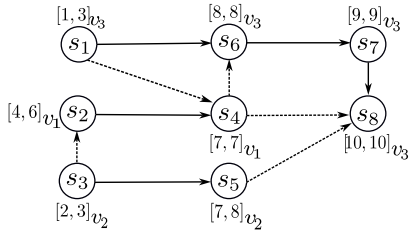

Fig. 3: The traject graph $\mathcal{G}_G$ of graph G in Figure 1

Optimization. Connecting all snapshots satisfying the above conditions may produce redundant edges in the traject graph.

(a) Transitivity. If snapshot s_1 can reach snapshot s_2, which can reach snapshot s_3, the edge from s_1 to s_3 is redundant.

To eliminate such redundancy, we require that there exists an edge from $s_1 = [t_s^i, t_e^i]_{v_i}$ to $s_2 = [t_s^j, t_e^j]_{v_j}$, only if $s_1, s_2 \in \mathcal{S}(u)$ and there does not exist a snapshot $s_3 = [t_s^l, t_e^l]_{v_l}$ in $\mathcal{S}(u)$ such that its time interval lies between the ending time of s_1 and the staring time of s_2, *i.e.,* no $s_3 = [t_s^l, t_e^l]_{v_l}$ in $\mathcal{S}(u)$ satisfies $t_e^i \leq t_s^l \leq t_e^l \leq t_s^j$.

(b) Chains. When snapshots $[t_1, t_2]_v, [t_2{+}1, t_3]_v, \ldots, [t_{n-1}+1, t_n]_v$ form a chain in $\mathcal{G}_G$, where each intermediate snapshot $[t_i{+}1, t_{i+1}]_v$ ($i \in [2, n-2]$) has exactly one incoming edge and one outgoing edge, and the first and last snapshots, $[t_1, t_2]_v$ and $[t_{n-1}+1, t_n]_v$, have only one outgoing edge and one incoming edge, respectively, we can merge the entire chain into a single snapshot $[t_1, t_n]_v$. This merging preserves the reachability relationships among vertices in $\mathcal{G}_G$.

Example 8: Figure 3 illustrates the traject graph $\mathcal{G}_G$ of the temporal bigraph G in Figure 1, with solid and dashed arrows indicating edges constructed under conditions (1) and (2), respectively. For instance, $\mathcal{S}(u_2){=}\{[4,6]_{v_1}, [7,7]_{v_1}, [2,3]_{v_2}, [8,8]_{v_3}, [9,9]_{v_3}\}$. A solid edge exists from $s_2 = [4,6]_{v_1}$ to $s_4 = [7,7]_{v_1}$ since s_2 ends before s_4 starts; and a dashed edge connects $s_3 = [2,3]_{v_2}$ to $s_2 = [4,6]_{v_1}$ due to the edges $(u_2, v_2, 2, 3)$ and $(u_2, v_1, 4, 7)$.

Redundant edges of types (a) and (b) have been removed from $\mathcal{G}_G$ in Figure 3. For instance, although an edge from s_3 to s_6 could be added based on edges $(u_2, v_2, 2, 3)$ and $(u_2, v_1, 8, 9)$ of u_2. We omit this edge in $\mathcal{G}_G$ since s_3 can already reach s_6 via path $s_3 s_2 s_4 s_6$. Additionally, s_3 is the result of merging $[2,2]_{v_2}$ and $[3,3]_{v_2}$. □

Properties. We next present properties of traject graphs.

Theorem 1: *Given a bigraph $G{=}(U{\cup}L, E)$, the traject graph $\mathcal{G}_G{=}(\mathcal{V}, \mathcal{E})$ satisfies $|\mathcal{V}|{\leq}4|G|$ and $|\mathcal{E}|{\leq}4d_{\max}|G|$, where $d_{\max}$ is the maximum degree of vertices in G.* □

Proof: (1) We first show that $|\mathcal{V}|{\leq}d_{\max}|G|$. Note that for each bottom vertex v, there are at most $4d_v$ many snapshots where d_v is the degree of v. This is because each edge incident to v contributes at most two distinct timestanps, resulting in at most $2d_v$ timestamps, and for each timestamp, there can be at most two snapshots. Therefore, the total number of vertices in $\mathcal{V}$ is bounded by $\Sigma_{v\in L} 4d_v \leq 4|G|$.

(2) We next show that $|\mathcal{E}|{\leq}4d_{\max}|G|$. Observe that (a) there exist at most $O(2|G|)$ many snapshots; and (b) each snapshot $[t_1, t_2]_v$ has at most $O(|d_v|)$ many edges, one for each edge of the bottom vertex v, since these edges link to snapshots constructed from (a) the same edges in G as s denoting that a top vertex remains linking to v, or (b) the edges whose ending time is t_2 denoting that a top vertex leaving v. Moreover, we assume that each top vertex is connected to at most one bottom vertex at timestamp t, as usually found in disease prevention [61], [49], [36]. Then a top vertex link to only one bottom vertex after a timestamp, and the number of edges in case (b) is bound by $O(d_v)$. Therefore, the number of edges for each snapshot $[t_1, t_2]_v$ is bounded by $O(|d_v|)$, and the number of edges in $\mathcal{G}_G$ is bounded by $\Sigma_{v\in L} 4d_v^2 \leq 4d_{\max}|G|$. □

Span-reachability to reachability. We show that the span-reachability problem for a temporal bigraph G can be reduced to the reachability problem in its traject graph $\mathcal{G}_G$.

We start with some notations. Assume that snapshots in $\mathcal{S}(G)$ are ordered by their starting times. Given time interval $[t_s, t_e]$ and a top vertex u, let src_u (resp. dest_u) be the *first snapshot* (resp. *last snapshot*) in $\mathcal{S}(G)$ *after* timestamp t_s (resp. *before* timestamp t_e), which are constructed using the timestamps of edges of a top vertex u. There exists only one snapshot after and before a given timestamp for a top vertex u, since we assume that each top vertex is connected to at most one bottom vertex at timestamp t (see above).

Theorem 2: *Given a temporal bigraph $G{=}(U \cup L, E)$, top vertices u and v, and interval $[t_s, t_e]$, u can reach v in G in $[t_s, t_e]$ if and only if src_u can reach dest_v in $\mathcal{G}_G$.* □

Proof: We provide a proof sketch; see [2] for details.

($\Rightarrow$) Assume that u can reach v in G in time interval $[t_s, t_e]$. Then there exists a temporal path $\rho = u_1 v_1 u_2 v_2 \ldots u_k v_k u_{k+1}$ with $u = u_1$ and $u' = u_{k+1}$ such that the starting time and ending time of ρ fall in the time interval $[t_s, t_e]$. We can construct a path from src_u to dest_v in $\mathcal{G}_G$ by constructing a path in $\mathcal{G}_G$ from each two consecutive connections $\langle u_i, e_1^i, v_i, e_2^i, u_{i+1}\rangle$ and $\langle u_{i+1}, e_1^{i+1}, v_{i+1}, e_2^{i+1}, u_{i+2}\rangle$.

($\Leftarrow$) Assume that src_u can reach dest_v. Let $P_{\mathcal{G}} = (s_0 = [t_s^0, t_e^0]_{v_0}, s_1 = [t_s^1, t_e^1]_{v_1}, \ldots, s_n = [t_s^n, t_e^n]_{v_n})$ be a path witnessing the reachability. Starting from the first edge, we construct a path for each edge from s_i to s_{i+1} ($i \in [1, n-1]$), such that they form a temporal path in G. But the path may not start with u or end with u'. To solve these, we extend the path with two more connections to link u and u'. □

Complexity. We can verify that $\mathcal{G}_G$ is a DAG. Moreover, $\mathcal{G}_G$ can be constructed in $O(d_{\max}|G| \log |G|)$ time, where $d_{\max}$ is the maximum degree in G. Observe that (a) graph $\mathcal{G}_G{=}(\mathcal{V}, \mathcal{E})$ has at most $d_{\max}|G|$ vertices and $d_{\max}|G|$ edges; and (b) it takes $O(\log |G|)$ time to sort the edges.

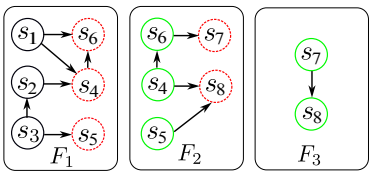

Fig. 4: A partition $\{F_1, F_2, F_3\}$ of $\mathcal{G}_G$ given in Figure 3.

VI. Index Construction

We present a contraction-based index schema for SRP. We first partition traject graphs into multiple fragments (Section VI-A), then construct both intra-fragment and inter-fragment indices for these fragments, and finally parallelize index construction to handle large-scale graphs (Section VI-B).

A. Graph Partitioning

We first present some notations about partitioning.

Partition. Given a number k, we partition graph $\mathcal{G}=(\mathcal{V}, \mathcal{E})$ into k fragments $F_1, F_2, \ldots, F_k$ such that (1) $F_i = (\mathcal{V}_i, \mathcal{E}_i, \mathsf{IN}_i, \mathsf{OUT}_i)$ is a fragment; (2) $\mathcal{V} = \cup_{i\in\{1,\ldots,k\}}\mathcal{V}_i$; (3) $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$ for $i \neq j$; (4) IN_i is the set of vertices $v\in \mathcal{V}_i$ with incoming edges from a vertex outside V_i; (5) OUT_i is the set of vertices $v\in \mathcal{V} \setminus \mathcal{V}_i$ with incoming edges from a vertex in V_i; and (6) $\mathcal{E}_i$ consists of all edges between vertices in $\mathcal{V}_i \cup \mathsf{IN}_i \cup \mathsf{OUT}_i$. Such partitions are called edge-cut [9], *i.e.,* each vertex in $\mathcal{V}_i$ has all its adjacent edges in F_i as in $\mathcal{G}$.

Vertices in $\mathsf{IN}_i \cup \mathsf{OUT}_i$ are called *border vertices*. Vertices in IN_i are contained in V_i, while vertices in OUT_i are not. Such difference is used to connect border nodes (see below).

A partition is ε-balanced, if $|\mathcal{V}_i| \leq (1+\varepsilon)\frac{|\mathcal{V}|}{k}$ for each fragment $F_i=(\mathcal{V}_i, \mathcal{E}_i, \mathsf{IN}_i, \mathsf{OUT}_i)$ [9]. Here, the parameter $\varepsilon \geq 0$ is the tolerance threshold for a partition.

Multiple algorithms have been developed for edge-cut partitions [9], [5], [48]. Leveraging the temporal information in the traject graph $\mathcal{G}$ and the order of timestamps, we adopt a greedy strategy to group vertices [21]. More specifically, it (1) first sorts vertices based on the timestamps on their adjacent edges, and (2) then iteratively allocates these vertices to fragments following this order as follows: for the first vertex v in the order, (a) if F_i is not full, *i.e.,* $|\mathcal{V}_i| \leq (1+\varepsilon)\frac{|\mathcal{V}|}{k}$, assign v to F_i; otherwise, it initializes another new fragment F_{i+1} with v; and (b) remove v from the order. These steps are repeated until all vertices are allocated to fragments.

Example 9: Graph $\mathcal{G}_G$ in Fig. 3 is partitioned into fragments F_1, F_2 and F_3 (Fig. 4), with sets $\{s_1, s_2, s_3\}$, $\{s_4, s_5, s_6\}$ and $\{s_7, s_8\}$, respectively. Vertices in IN_i (resp. OUT_i) are marked by green solid (resp. red dashed) circles. □

Reachability queries. Given a partitioned graph, the span-reachability queries are divided into two categories:

- the *intra-fragment* reachability queries, where the source vertex and target vertex are in the same fragment; and
- the *inter-fragment* reachability queries, where the source vertex and target vertex are in two different fragments.

Intra-fragment indices. To answer intra-fragment reachability queries, we pre-process each fragment using existing indexing strategies, such as TopChain [57], pathtree [31], grail [59] and ferrari [39] (see Section VII).

In the following, we focus on inter-fragment indices.

B. Inter-fragment indexing

We derive the inter-fragment indices by constructing a contraction graph from the traject graph $\mathcal{G}_G$. Intuitively, the contraction graph merges vertices that share the same connectivity to the border nodes [23].

Equivalent Sets. We first define two equivalent relations among vertices in traject graph $\mathcal{G}_G$, to categorize vertices based on their connectivity to the border nodes.

More specifically, given a fragment $F_i=(\mathcal{V}_i, \mathcal{E}_i, \mathsf{IN}_i, \mathsf{OUT}_i)$, we define two equivalent relations EOut_i and EIn_i on $\mathcal{V}_i$, called *out-equivalence relation* and *in-equivalence relation*, respectively. For each $v \in \mathcal{V}_i$, (1) its *out-equivalence class* $[v]_{\mathsf{EOut}_i}$ in EOut_i consists of vertices in $\mathcal{V}_i$ that reach exactly the same set of border vertices in OUT_i as v does. (2) The *in-equivalence class* $[v]_{\mathsf{EIn}_i}$ in EIn_i is the set of all vertices in $\mathcal{V}_i$ that are reachable from exactly the same set of vertices in IN_i as v is. Here, $[v]_{\mathsf{EIn}_i}$ is defined by checking vertices in IN_i, *i.e.,* vertices in the same fragment as v, which is used to connect different fragments, for reachability (see Section VII).

Furthermore, for each $v \in \mathcal{V}_i$, we define $\mathsf{BIn}_i[v]$ (resp. $\mathsf{BOut}_i[v]$) as the set of border vertices that can reach (resp. be reached from) v. Let BIn_i and BOut_i denote the sets of all $\mathsf{BIn}_i[v]$ and $\mathsf{BOut}_i[v]$, respectively.

Example 10: (1) For F_1 in Figure 4, s_1 and s_2 reach the same set of border vertices (*i.e.,* $\{s_4, s_6\}$), and s_3 can reach all border vertices. We have $[s_1]_{\mathsf{EOut}_1}=[s_2]_{\mathsf{EOut}_1}=\{s_1, s_2\}$, and $[s_3]_{\mathsf{EOut}_1}=\{s_3\}$. Moreover, $[s_1]_{\mathsf{EIn}_1}=[s_2]_{\mathsf{EIn}_1}=[s_3]_{\mathsf{EIn}_1}=\{s_1, s_2, s_3\}$ since IN_1 is $\emptyset$. (2) For F_2 in Figure 4, $\mathsf{IN}_2=\{s_4, s_5, s_6\}$ since these vertices all have incoming edges from vertices in F_1. Because s_6 can be reached from both s_4 and s_5, and s_4 and s_5 can be reached from only themselves, we get $[s_4]_{\mathsf{EIn}_2}=\{s_4\}$, $[s_5]_{\mathsf{EIn}_2}=\{s_5\}$ and $[s_6]_{\mathsf{EIn}_2}=\{s_6\}$. □

Construction. We can construct the equivalence relations EOut_i and EIn_i as follows: (1) for each vertex v in $\mathcal{V}_i$, determine the set of border vertices, that v can reach, along with the subset of vertices in IN_i that can reach v; (2) cluster vertices in $\mathcal{V}_i$ based on these sets, *i.e.,* two vertices are in the same cluster if they can reach or be reached from the same sets of border vertices (see below). However, such algorithm takes $O(|\mathcal{V}_i||\mathcal{E}_i|)$ time and is costly when F_i is large.

To solve this, we develop Algrithm 1, to compute the out-equivalence relation EOut_i by iteratively removing vertices without outgoing edges, which is extended from the topological sorting algorithm [17]. Given a fragment F_i, it first identifies a set B_i of border vertices in OUT_i that have no outgoing edges (line 1). Since F_i is a DAG, such vertices are guaranteed to exist. It then initializes $\mathsf{Bout}_i[v]=\{v\}$ for each border vertex

Algorithm 1: OutEquivalentSet

Input: Fragment $F_i = (\mathcal{V}_i, \mathcal{E}_i, \mathsf{IN}_i, \mathsf{OUT}_i)$.
Output: Equivalent sets $[v]_{\mathsf{EOut}_i}$ for all $v \in V_i$.

```
1  build B_i of border vertices in without outgoing edges;
2  set Bout_i[v] = {v} for all v ∈ B_i;
3  while B_i ≠ ∅ do
4      v = B_i.pop();
5      for each u having an edge e leading to v do
6          Bout_i[u] := Bout_i[u] ∪ Bout_i[v];
7          remove edge e from F_i;
8          if u has no outgoing edge then
9              push u to B_i;
10 EOut_i := GroupEquiv(Bout_i);
11 return EOut_i;
```

$v \in B_i$ (line 2). The iterative process to compute EOut_i works as follows (lines 3-9). Starting with a border vertex $v \in B_i$ (line 3), for each incoming neighbor u of v, the algorithm merges $\mathsf{Bout}_i[u]$ and $\mathsf{Bout}_i[v]$ (line 5-6), removes the edge from u to v from the fragment (line 7), and checks whether all edges of u have been processed (line 8). If so, u is added to B_i for further processing (line 9). The process continues until B_i becomes empty (line 3). Finally, it groups vertices in $\mathcal{V}_i$ based on the sets Bout_i (line 10); that is, vertices v_j and v_k are in the same out-equivalence class if $\mathsf{Bout}_i[v_j] = \mathsf{Bout}_i[v_k]$.

The equivalence-class EIn_i can be constructed similarly.

Analysis. One can readily verify the correctness of the algorithm. OutEquivalentSet runs in $O(|\mathcal{E}_i||\mathsf{OUT}_i|)$ time, since (1) each edge is accessed only once (lines 5 and 7), and there exist $O(|\mathcal{E}_i|)$ many edges; (2) it computes the union of two sets in $O(|\mathsf{OUT}_i|)$ time (line 6); and (3) it uses a hashmap to group vertices (line 10), which takes $O(|\mathcal{V}_i||\mathsf{OUT}_i|)$ time. In practices, $|\mathsf{OUT}_i|$ is much smaller than $|\mathcal{E}_i|$, *i.e.,* OutEquivalentSet is faster than the algorithm that directly compare vertices.

Contraction graphs. For inter-fragment queries, we construct a contraction graph to encode reachability between fragments.

For a traject graph $\mathcal{G}_G{=}(\mathcal{V}, \mathcal{E})$ and its partition $\{F_1, F_2, \ldots, F_k\}$, we define its *contraction graph* $\mathcal{G}_G^c{=}(\mathcal{V}^c, \mathcal{E}^c)$ as follows.

(1) The vertex set $\mathcal{V}^c{=}\bigcup_{i\in[1,k]}(\mathsf{EIn}_i{\cup}\mathsf{EOut}_i{\cup}\mathsf{BIn}_i{\cup}\mathsf{BOut}_i)$, *i.e.,* $\mathcal{V}^c$ includes equivalence classes and sets of border vertices.

(2) The edge set $\mathcal{E}^c$ consists of edges connecting border vertices. More specifically, for each vertex $v \in \mathcal{V}_i$,

(a) add an edge from its out-equivalence class $[v]_{\mathsf{EOut}_i}$ to the set $\mathsf{Bout}_i[v]$ of border vertices that v can reach; and add an edge from $\mathsf{Bin}_i[v]$ to the in-equivalence class $[v]_{\mathsf{EIn}_i}$; here $\mathsf{BIn}_i[v]$ consists of border vertices that can reach v in F_i;

(b) add edge from $\mathsf{Bout}_i[v]$ to $\mathsf{Bin}_j[v']$ with $i{\neq}j$, if $\mathsf{Bout}_i[v]{\cap}\mathsf{Bin}_j[v']{\neq}\emptyset$, *i.e.,* there is a vertex u in F_j such that u is a border vertex in F_i and v can reach u in F_i, which can reach v' in F_j;

(c) add an edge from $\mathsf{Bout}_i[v]$ to $\mathsf{Bout}_j[v']$, if $i{\neq}j$ and v' is in $\mathsf{Bout}_i[v]$; recall that if v' is in $\mathsf{Bout}_i[v]$ and v' is a vertex in F_j, then v can reach v', and subsequently, all vertices in $\mathsf{Bout}_j[v']$.

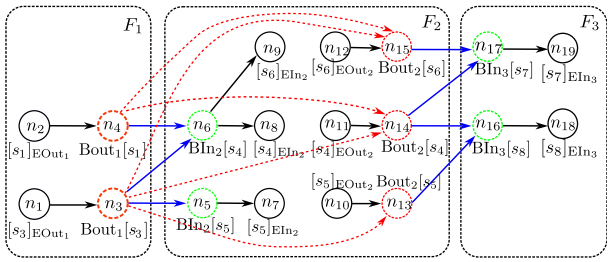

Fig. 5: Contraction Graph $\mathcal{G}_G^c$

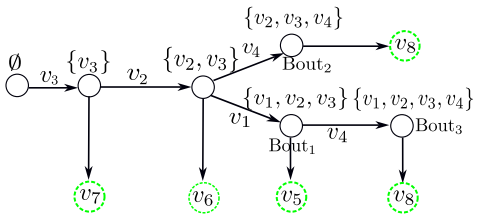

Fig. 6: The tries data structure

Example 11: We construct contraction graph $\mathcal{G}_G^c$ (see Fig. 5) based on partitions in Fig. 4. (1) $[s_3]_{\mathsf{EOut}_1}$ can reach $\mathsf{Bout}_1[s_3]$, since s_3 can reach vertices in $\mathsf{Bout}_1[s_3]$, *i.e.,* snapshots s_4, s_5, s_6; (2) $\mathsf{Bin}_2[s_5]$ can reach $[s_5]_{\mathsf{EIn}_2}$, as $\mathsf{Bin}_2[s_5]$ consists of border vertices that can reach s_5; (3) $\mathsf{Bout}_1[s_3]$ can reach $\mathsf{Bin}_2[s_5]$, since $\mathsf{Bout}_1[s_3]{\cap}\mathsf{Bin}_2[s_5] = \{s_5\}$; and (4) $\mathsf{Bout}_1[s_3]$ can reach $\mathsf{Bout}_2[s_5]$, as $s_5 \in \mathsf{Bout}_1[s_3] = \{s_4, s_5, s_6\}$ and $\mathsf{Bout}_2[s_5]$ consists of all vertices that s_5 can reach in F_2. □

Calibration. The contraction graph may contain redundant edges, which degrades query performance. Consider an edge (u, v) in graph G, where u exists in fragment F_2, and it also belongs to m equivalence classes $\mathsf{Bout}_1[u_1], \ldots, \mathsf{Bout}_1[u_m]$ within fragment F_1. It results in m edges in the contraction graph, one for each equivalence class $\mathsf{Bout}_1[u_i]$ $(i{\in}[1, m])$. To address this, we use the trie data structure [26], to reduce such redundant edges. Specifically, we first assign a unique number to each vertex in $\mathcal{G}_G^c$, and sort them accordingly, and represent equivalence classes $\mathsf{Bout}_1[u_1], \ldots, \mathsf{Bout}_1[u_m]$ using a prefix tree based on the sorted order.

Example 12: Assume that the contraction graph $\mathcal{G}_G^c$ contains equivalence classes: $\mathsf{BOut}_1[u_1] = \{v_1, v_2, v_3\}$, $\mathsf{BOut}_2[u_2] = \{v_2, v_3, v_4\}$ and $\mathsf{BOut}_1[u_3] = \{v_1, v_2, v_3, v_4\}$. Moreover, $v_1, \ldots, v_4$ have an edge linking to vertex $v_5, \ldots, v_8$, respectively, which exist in another fragment. Observe that v_1, v_2, v_3 and v_4 appear for 2, 3, 3 and 2 times in these equivalence classes, resulting in 10 edges in $\mathcal{G}_G^c$. However, when we store these equivalence classes using a trie as shown in Figure 6, $\mathcal{G}_G^c$ contains only 5 edges to link equivalence classes in other fragments (marked by green dashed circles), which reduces the size of $\mathcal{G}_G^c$ and accelerate the query (see Section VIII). □

Analysis. Due to the constructions of equivalence classes, we can verify that given two snapshots s_1 and s_2, s_1 can reach s_2 in $\mathcal{G}_G$ if and only if s_1 can reach s_2 in the contraction graph $\mathcal{G}_G^c$. Indeed, $\mathsf{Bout}_i[v]$ and $\mathsf{Bin}_i[v]$ are the sets of border

vertices that can be reach and reach v in $\mathcal{G}_G$, respectively.

We can construct $\mathcal{G}_G^c$ in $O(\Sigma_{i\in[1,k]}|\mathcal{V}_i||\mathsf{Out}_i|)$, since each $v \in V_i$ leads to at most four edges.

Remark. (1) To accelerate queries for vertices across different fragments, we construct the chain cover on $\mathcal{G}_G^c$ [57]. More specifically, (a) we first decompose $\mathcal{G}_G^c$ into a set of chains such that each vertex in $\mathcal{G}_G^c$ exists in exactly one chain. Here, a chain is a sequence of vertices $u_1, \ldots, u_n$ such that u_i has an outgoing edge to u_{i+1} with $(i \in [1, n-1])$; and (b) we next compute the reachability labels for each vertex in $\mathcal{G}_G^c$; intuitively, the labels for vertex u record the chains that u can reach and can be reached from u. This differs from algorithm TopChain [57], which directly applies chain covers on temporal graphs. We construct the chain covers on both fragments and contraction graphs, which are smaller than the original graphs, leading to better performance (see Section VIII).

(2) Upon graph updates, we leverage incremental algorithms from [24], [22] to maintain the contraction graph $\mathcal{G}_G$. Intuitively, given a set of updates, we first identify vertices impacted by these changes. Then we recompute equivalent classes for these vertices. Finally, we update $\mathcal{G}_G$ accordingly. In this way, we avoid recomputing $\mathcal{G}_G$ starting from scratch.

(3) For large-scale graphs, constructing indices is costly [50]. To address this, we can develop a parallel index construction algorithm as follows: (a) transformation processes the adjacent edges of each lower vertex independently; (b) partition the graph using timestamps; (c) *independently* construct the intra-fragment index within each fragment using existing algorithms; (d) *independently* compute equivalent sets for vertices within each fragment; and (e) construct the contraction graph by linking each vertex *independently*. Steps (a), (c), (d) and (e) are easily parallelizable, since they are independent tasks. Step (a) can be achieved through a linear scan of all edges, sorted by their timestamps, which takes less time than the other steps.

VII. Query on Contraction Graphs

We develop algorithm SRPQ for the span-reachability problem using indices presented in Section VI.

Overview. To answer the reachability in partitioned graphs, we must tackle the following challenges: (1) convert the reachability problem from a bipartite graph to the one in contraction graphs preserving the reachability property. (2) How to conduct reachability queries on the contraction graphs, especially when the given two vertices are not in the same fragment.

We solve these problems as follows.

(1) Given vertices u and v, and time interval $[t_s, t_e]$, we first identify two snapshots src_u and dest_v in the contraction graph such that the reachability is preserved, *i.e.,* u can reach v in $[t_s, t_e]$ if and only if src_u can reach dest_v (see Section V).

(2) We first check whether src_u and dest_v are in the same fragment. If so, we use intra-fragment indices to check the reachability; recall that each fragment is preprocessed by state-of-the-art indexing strategies, *e.g.,* TopChain [57], pathtree [31], grail [59] and ferrari [39]. (2) When they are in different fragments, we take path-tree covers [31] as a case study, to construct indices on contraction graphs for span-reachability.

Algorithm 2: SRPQ

Input: Traject Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and vertices v_1 and v_2.
Output: Whether v_1 can reach v_2?
1 **if** v_1 and v_2 are in the same fragment F_i **then**
2 **return** $\mathsf{Query}(v_1, v_2, F_i)$;
3 **else**
4 **return** $\mathsf{TopC}(v_1, v_2, \mathcal{G})$;

Algorithm 3: TopC

Input: Traject Graph $\mathcal{G}=(\mathcal{V}, \mathcal{E})$ and vertices src_u and dest_v.
Output: Whether src_u can reach dest_v?
1 $(F_1, F_2) := \mathsf{Locate}(\mathsf{src}_u, \mathsf{dest}_v, \mathcal{G})$;
2 $\mathsf{src}_{v_1} := \mathsf{Border}_{\mathsf{out}}(v_1, F_1, \mathcal{G})$;
3 $\mathsf{dest}_{v_2} := \mathsf{Border}_{\mathsf{in}}(v_2, F_2, \mathcal{G})$;
4 bReach := $\mathsf{CheckReach}(\mathsf{src}_{v_1}, \mathsf{dest}_{v_2})$;
5 **return** bReach;

Algorithm. We present SRPQ in Algorithm 2. Given vertices src_u and dest_v, and traject graph $\mathcal{G}$, it first checks whether src_u and dest_v are in the same fragment; if so, it exploits the intra-partition indices to check whether src_u can reach dest_v (line 1); otherwise, it applies procedure TopC (line 4).

Procedure TopC. Given vertices src_u and dest_v, and a contraction graph $\mathcal{G}$, it checks whether src_u can reach dest_v using the chain covers [57]. (1) At first, it identifies fragments F_1 and F_2, in which src_u and dest_v are located, respectively (line 1). Recall that graph G is partitioned via edge-cut, and each vertex is located in only one fragment. (2) After that it fetches the set $\mathsf{Bout}_1[\mathsf{src}_u]$, *i.e.,* the set of vertices that src_u can reach in F_1 (line 2). This can be done by accessing equivalence class $[\mathsf{src}_u]_{\mathsf{EOut}_1}$. Similarly, it collects $\mathsf{Bin}_2[\mathsf{dest}_v]$, *i.e.,* the set of vertices that can reach dest_v in F_2 (line 3). (3) Finally, it checks whether $\mathsf{Bout}_1[\mathsf{src}_u]$ can reach $\mathsf{Bin}_2[\mathsf{dest}_v]$ via CheckReach using path-tree covers defined on $\mathcal{G}$ (line 4).

Example 13: Given bipartite graph G in Fig. 1 and the contraction graph in Fig. 4, consider the following two queries.

(1) For query $Q(u_1, u_4, G, [3, 7])$, observe that $S^f_{u_1} = \{[4, 7]_{v_1}\} = \{s_3\}$ and $S^l_{u_4} = \{[6, 7]_{v_2}\} = \{s_6\}$, which are located in the same fragments, *i.e.,* F_1. The query can be answered by intra-fragment indices.

(2) For query $Q(u_1, u_4, G, [3, 9])$, $S^f_{u_1} = \{[4, 7]_{v_1}\} = \{s_3\}$ and $S^l_{u_4} = \{[8, 8]_{v_3}\} = \{s_{11}\}$, which are located in different fragments. Observe that (a) s_3 belongs to $[s_3]_{\mathsf{EOut}_1}$ and s_{11} belongs to $[s_{11}]_{\mathsf{EIn}_3}$; and (b) $[s_3]_{\mathsf{EOut}_1}$ can reach $[s_{11}]_{\mathsf{EIn}_3}$ in $\mathcal{G}_G^c$. Therefore, u_1 can reach u_4 in time interval $[3, 9]$. □

Analysis. We next analyze the complexity of SRPQ. Assume that TopChain [57] takes $O(f(|G|))$ time to answer a query. Here, $f(|G|)$ is a polynomial to denote the complexity of

TABLE II: Dataset Summary

Name	Dataset	$\|E\|$	$\|U\|$	$\|L\|$	Δ
WP	wikiquote-pl	378,979	4,312	49,500	4,750
SO	stack-overflow	1,301,943	545,195	96,678	1,155
LK	linux-kernel	1,565,684	42,045	337,509	12,543
CU	citeulike	2,411,820	153,277	731,769	1,204
BS	bibsonomy	2,555,081	5,794	204,673	7,667
TW	twitter	4,664,606	175,214	530,418	27,742
AM	amazon	5,838,042	2,146,057	1,230,915	3,651
EP	epinion	13,668,321	120,492	755,760	505
LF	lastfm	19,150,869	992	1,084,620	3,149
EJ	edit-jawiki	41,998,341	444,563	2,963,672	5,574
ED	edit-dewiki	129,885,940	1,025,084	5,812,980	5,953

TABLE III: Average Positive and Negative Query Time (μs)

Name	Positive queries				Negative queries			
	TC	PT	TBP	SRPQ	TC	PT	TBP	SRPQ
WP(8)	31.52	27.11	0.21	26.23	3.28	2.61	0.52	3.24
SO(8)	57.92	16.45	2.03	29.53	0.23	0.86	3.37	0.33
LK(8)	33.76	29.64	7.97	12.79	1.36	2.53	19.26	1.27
CU(4)	66.65	105.91	OT	31.85	0.35	6.78	OT	0.47
BS(16)	127.92	106.79	5.81	107.09	21.37	21.66	12.95	23.33
TW(8)	1,012.66	333.14	OT	128.95	1.40	8.27	OT	1.46
AM(4)	59.97	6.77	1.82	20.40	0.46	0.92	1.86	0.45
EP(4)	771.45	OM	OT	127.07	0.42	OM	OT	0.64
LF(2)	71.76	52.51	6.14	59.14	6.81	6.60	11.99	6.63
EJ(4)	92.19	200.08	13.70	38.56	4.26	10.07	41.70	4.33
ED(4)	763.74	OM	OT	202.02	15.52	OM	OT	16.40

TopChain, and is a monotonic function, *i.e.,* if $x < y$, then $f(x) < f(y)$. Indeed, when the graph gets larger, TopChain takes more time to answer a query. Then SRPQ takes $O(\max(f(\max_i |F_i|), f(|\mathcal{G}_G^c|)))$ to answer a query, and is faster than TopChain, since F_i and $\mathcal{G}_G^c$ are much smaller than G, when the number of fragments is large (see Section VIII).

VIII. Experiments

Using real-life and synthetic data, we experimentally evaluated CBSR for its (1) efficiency (2) effectiveness and (3) scalability. We also conducted (4) an ablation study on CBSR.

Experimental setting. We start with the setting.

Graphs. We used eleven real-life graphs and a synthetic graph, summarized in Table II. Here, Name denotes the abbreviation of dataset, $|E|$ is the number of edges, and $|U|$ and $|L|$ are the numbers of top and bottom vertices in the graph, respectively. Δ is the time span(day) of the graph, i.e., the difference between the maximum and minimum timestamps in the graph. All real-life graphs are from the Konect dataset [1]. However, the edges carry only one timestamp, then we generated the time intervals by sampling an interval length Δt from a power-law distribution $p(\Delta t) = C\tau^{-\alpha}$ [6], with C ranging from 10 minutes to 8 hours and $\alpha = -2.5$ following [13].

Algorithms. We evaluated the following algorithms.

Index construction algorithms. We implemented in C++: (1) CBSR, the algorithm combining graph transformation (Section V) and index construction (Section VI); the intra-fragment indices are constructed by TopChain [57] as a case study;

We compared with (2) TBP [13], an implementation provided in [38]. We also compared CBSR with algorithms for DAGs: (3) TopChain (TC) [57], chain-based indices for temporal DAGs; and (4) PathTree (PT) [31], a path-tree based index for temporal bipartite graphs.

Query algorithms. We compared query algorithm SRPQ (Section VII) with two kinds of algorithms: (a) query algorithms on bipartite temporal graphs, and (b) query algorithms on DAGs.

(a) For queries on temporal bipartite graphs, we implemented in C++: (5) SRPQ, our proposed query algorithm from Section VII; and We compared with (6) query algorithm of TBP from [13]. As we previously emphasized, the problem defined in TBP [13] differs from ours (see Example 5). To ensure a fair comparison, we preprocess each edge $e = (u, v, t_s, t_e)$ by splitting it into multiple edges with smaller intervals, such that SRPQ and TBP produce the same answers on the same input.

(b) For queries on DAGs, we compared SRPQ with TopChain [57] and PathTree [31]. Since their original implementations do not support bipartite graphs, they are evaluated on the DAGs transformed with the strategy in Section V.

Environment. We run experiments on a Linux server with Intel Xeon Platinum 8358 CPU (2.6GHz) and 1TB memory, which has 32 cores and 64 threads. Each experiment was run 5 times, and the average is reported.

Experimental results. We next report our findings. In the following, OT denotes that the algorithm cannot finish within 24 hours, and OM for running out of memory (OOM).

Exp-1: Query processing. We first evaluated the performance of the query algorithms using indices. To evaluate the performance of the algorithms, we partition the queries into (a) positive queries, *i.e.,* a pair (u, v, t_s, t_e) of top vertices that u *can* reach v in the time interval $[t_s, t_e]$; and (b) negative queries, *i.e.,* a pair (u, v, t_s, t_e) of top vertices that u *cannot* reach v in the time interval $[t_s, t_e]$. And we sampled 1M positive queries and negative queries from the graphs, and reported the average query time. The results are reported in Table III, where (a) WP(8) indicates the number of partitions used for dataset WP; similarly for other datasets.

(1) On large graph ED, SRPQ achieves an average of $3.78\times$ speedup over TopChain on positive queries. And TBP fails to build indices. For negative queires, SRPQ is slightly slower than TopChain, since TopChain has an effective optimization for negative queries, which does not depend on the graph size.

(2) On median-scale graphs (TW, AM, EP, LF, EJ), SRPQ achieves an average of $4\times$ speedup over TopChain on positive queries. Notably, it achieves $7.85\times$ and $6.07\times$ speedups on TW and EP, respectively, while TBP fails to produce any results due to index construction timeout.

(3) On small-scale graphs (WP, SO, LK, CU, BS), SRPQ consistently outperforms TopChain on positive queries, with speedup of $1.2\times$ on BS and over $2.6\times$ on LK.

Exp-2: Index construction. We then evaluated the indexing

TABLE IV: Graph Transformation

Name	$\|E\|$	$\|E_{\mathsf{DAG}}\|$	$\|V_{\mathsf{DAG}}\|$
WP	378,979	12,746	10,399
SO	1,301,943	556,044	520,871
LK	1,565,684	506,037	398,311
CU	2,411,820	2,262,206	1,381,283
BS	2,555,081	111,783	98,884
TW	4,664,606	3,445,596	2,684,630
AM	5,838,042	476,363	529,769
EP	13,668,321	5,815,905	4,163,505
LF	19,150,869	112,935	85,873
EJ	41,998,341	1,830,929	1,474,038
ED	129,885,940	24,225,743	18,946,499

TABLE V: Indexing Time and Memory Usage

Name	Indexing Time (ms)				Memory Usage (GB)			
	TC	PT	TBP	CBSR	TC	PT	TBP	CBSR
WP(8)	139	182	78,169	114	0.08	0.03	0.08	0.08
SO(8)	2,282	38,615	3,673,724	5,747	0.83	8.25	1.32	1.09
LK(8)	1,630	115,426	530,398	2,139	0.75	18.97	0.40	0.80
CU(4)	6,492	1,915,183	OT	15,638	2.33	209.89	OT	2.73
BS(16)	988	1,658	3,094,355	1,016	0.52	0.69	0.50	0.53
TW(8)	13,609	4,086,346	OT	69,760	4.38	596.73	OT	4.94
AM(4)	3,509	11,960	7,186,620	4,419	1.60	4.13	2.74	1.65
EP(4)	24,246	OM	OT	47,732	7.25	OM	OT	8.04
LF(2)	6,592	10,692	11,349,943	6,663	3.05	3.55	3.38	3.05
EJ(4)	19,499	2,407,021	35,224,180	22,205	6.69	298.79	7.12	6.69
ED(4)	134,457	OM	OT	261,702	36.90	OM	OT	36.95

time and memory usage of indexing algorithms.

Graph Transformation We first evaluated the graph transformation algorithm. The results are shown in Table IV, where $|E|$ is the number of edges in the temporal bipartite graphs, and $|E_{\mathsf{DAG}}|$ and $|V_{\mathsf{DAG}}|$ are numbers of edges and vertices of transformed DAG, respectively. We found that the numbers $|E_{\mathsf{DAG}}|$ of edges in DAGs are smaller than the given temporal bipartite graphs. And the ratio $\frac{|E_{DAG}|}{|E|}$ ranges from 0.59% (*e.g.,*LF) to 93.8% (*e.g.,*CU), which justifies the constructions of the snapshots and the traject graphs.

Indexing time. We compared the indexing time. The results are shown in Table V. We found the following.

(1) CBSR outperforms TBP in index construction over all datasets. CBSR is at least $248\times$ (*e.g.,* LK) and up to $3,045\times$ faster (*e.g.,* BS) than TBP, demonstrating the advantage of the contraction-based method. This is because TBP cannot handle fine-grained reachability queries as CBSR, and splitting edges can ensure the correctness but degrades its performance.

(2) Although CBSR is generally slower than TopChain in index construction due to the overhead for constructing contraction graphs, the performance gap is relatively small. Algorithm SRPQ is on average $3\times$ faster than TopChain on positive queries, when using the indices built by CBSR (see **Exp-1**).

Memory usage. Table V reports the used memory during index construction. CBSR uses similar memory as TopChain, *e.g.,* 36.90GB vs 36.95GB on ED, although CBSR constructs a hypergraph to accelerate conducting queries;

TABLE VI: The Running Time of CBSR (s)

Name	Trans	EqSet	Part	Inner	Intra
WP(8)	0.096	0.003	0.005	0.009	0.001
SO(8)	0.739	3.593	0.534	0.549	0.332
LK(8)	0.676	0.599	0.326	0.468	0.070
CU(4)	2.004	9.056	1.587	2.318	0.673
BS(16)	0.816	0.044	0.057	0.093	0.006
TW(8)	3.930	55.864	3.471	4.499	1.796
AM(4)	2.082	1.053	0.601	0.554	0.118
EP(4)	8.600	23.075	6.236	8.057	1.764
LF(2)	6.458	0.035	0.064	0.100	0.006
EJ(4)	14.951	2.756	1.859	2.277	0.182
ED(4)	64.554	129.994	31.612	34.513	1.029

TABLE VII: The impact of fragment number on four metrics

Name	Metric	Fragment Number						
		1	2	4	8	16	32	64
EP	IT(s)	24.24	28.13	47.73	79.50	126.88	212.67	333.39
	MU(G)	7.25	7.38	8.04	8.54	9.01	9.43	9.82
	PQT(μs)	771.45	329.99	127.07	169.13	480.85	409.34	630.62
	NQT(μs)	0.42	0.44	0.64	0.58	0.57	0.53	0.57
TW	IT(s)	13.61	22.52	41.05	69.76	106.36	150.33	195.06
	MU(G)	4.38	4.68	4.94	5.35	5.69	6.14	6.46
	PQT(μs)	1012.66	368.56	171.31	128.95	290.77	395.75	414.66
	NQT(μs)	1.40	1.39	1.46	1.46	1.44	1.41	1.45

Cost breakdown. We conducted a cost breakdown analysis on the runtime of CBSR, which is divided into five phases: graph transformation (Trans), graph partitioning (Part), inner-index construction (Inner), equivalent sets construction (EqSet), and intra-index construction (Intra). As shown in Table VI, (1) Computing the equivalent classes dominates the total cost for index construction. On SO, CU TW and ED, EqSet consumes 63%, 58%, 80% and 50% of the execution time of CBSR, respectively. But (2) on WP, BS, AM LF and EJ, Trans consumes 84%, 80%, 47%, 97% and 68% of the execution time of CBSR, respectively. This is because these graphs are sparse and the number of equivalence classes are small, which accelerates the computation of EqSet.

Exp-3: Scalability. We tested the scalability of CBSR.

Varying $|G|$. We varied the scale factor from 0 to 1 with a step length 0.2, to evaluate the scalability of CBSR. The initial bigraph is edit-enwiki [1], which has 8.1M top vertices, 42.5M bottom vertices and 572M edges. The fragment number is 4.

Indexing time. As shown in Fig. 7(a), (1) when $|G|$ gets larger, CBSR takes longer, as expected. (2) CBSR scales well. It takes 1850.97s to construct indices on a graph with 572M edges.

Querying time. As shown in Figure 7(b), (1) when $|G|$ gets larger, the query time of CBSR increases, as expected. (2) CBSR scales well with $|G|$. It takes 102μs to process a query on a graph with 50.6M vertices and 572M edges.

Memory usage. We evaluated memory usage of CBSR. As shown in Fig. 7(c), (1) when $|G|$ gets larger, CBSR uses more memory, as expected. (2) CBSR scales well. It takes 162GB to process a graph with 50.6M vertices and 572M edges.

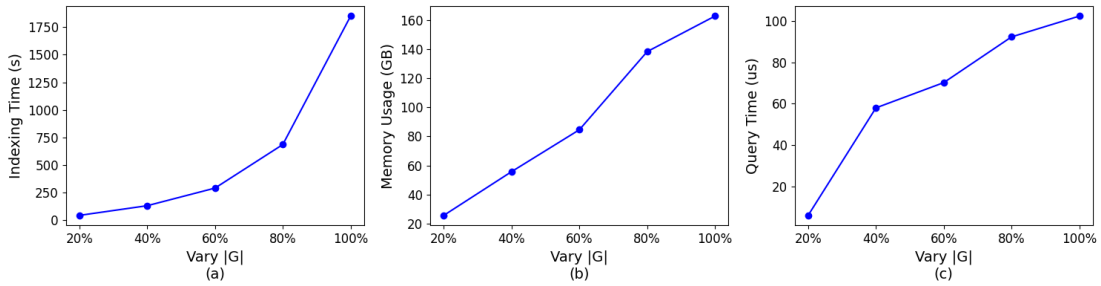

Fig. 7: Scalability

Exp-4: Ablation study. We evaluated the impact of (1) the number of fragments and (2) the index algorithm for fragments, on the performance of CBSR.

Varying the number of fragments. We varied the number N_f of fragments from 1 to 64, to evaluated the performance of CBSR and SRPQ. The results are shown in Table VII, where IT, MU, PQT and NQT represent indexing time, memory usage, positive query time, and negative query time, respectively.

(1) Algorithm CBSR takes more time to construct the index when N_f increases. This is because when N_f increases, the number of border vertices increases, which leads to more equivalence sets and larger contraction graphs. As shown Table VI, computing equivalence classes dominates the index construction time on EP and TW. When N_f varies from 1 to 64, CBSR is $12.8\times$ and $13.3\times$ slower when constructing indices on EP and TW, respectively.

(2) The space overhead of TopChain used in a fragment is linear to the number of vertices. Therefore the memory augmentation comes mainly from the contraction graph and its indices. From 1 to 64 fragments, the memory usage of EP and TW increases by 35% and 47%, respectively.

(3) For the negative query, SRPQ is insensitive to N_f, due to the optimization strategy of TopChain for negative queries, which can identify negative queries in the early stage of the query. We generated queries that could not be filtered out in early stages. For these queries, TopChain's performance degraded considerably. For example, the average negative query time on graph TW becomes $10.68\mu s$, which is $6.6\times$ slower than random generated negative queries (see Table III).

Exp-5: Connection latency. We next studied the impact of a parameter, namely latency, on performance of CBSR. Recall that given two snapshots $s_1 = [t_s^1, t_e^1]_{v_1}$ and $s_2 = [t_s^2, t_e^2]_{v_2}$, we add an edge from s_1 to s_2, if $s_1.t_e \leq s_2.t_s$ and there exists a top vertex u links to v_1 and v_2 at time t_e^1 and t_s^2, respectively. However, when t_s^2 is sufficiently larger than t_e^1, *e.g.,* t_s^2 is one year after t_e^1, the edge from s_1 to s_2 is useless for the reachability problem, especially for disease tracing, in which the disease has a much shorter latency. Then we define the latency as the maximum time span that two snapshots can be linked, *i.e.,* two snapshots $s_1 = [t_s^1, t_e^1]_{v_1}$ and $s_2 = [t_s^2, t_e^2]_{v_2}$ cannot have an edge if $|t_e^1 - t_s^2| \geq \delta$ or $|t_e^2 - t_s^1| \geq \delta$.

TABLE VIII: The impact of δ on four metrics

Name	Metric	δ(day)					
		1	2	4	8	16	32
EP	IT(ms)	39,095	39,456	39,820	43,058	43,427	44,275
	MU(G)	7.75	7.76	7.77	7.79	7.80	7.82
	PQT(μs)	120.80	116.32	120.35	122.36	125.56	128.23
	NQT(μs)	0.45	0.44	0.43	0.44	0.44	0.44
TW	IT(ms)	17,131	21,187	29,427	42,072	52,980	61,529
	MU(G)	4.36	4.50	4.68	4.93	5.11	5.23
	PQT(μs)	62.67	65.19	76.59	103.61	142.44	136.11
	NQT(μs)	0.91	0.90	0.93	0.93	0.93	0.93

As shown in Table VIII, (1) when δ gets larger, the number of edges in traject graphs increases, as expected. Then both CBSR and SRPQ get slower. (2) The latency has different impacts on CBSR for different graphs. On TW, CBSR is $3.6\times$ faster when δ varies from 32 to 1, since TW models relations between users and tags that they used, which is not "updated" frequently. However, on EP algorithm CBSR is only $1.1\times$ faster, when δ varies from 32 to 1. This is because each user (*i.e.,* top vertex) may star products during a short time interval. The latency has a slight impact on the sizes of traject graphs.

Summary. We found the following: (1) Our graph transformation algorithm is effective. On average, the generated DAG is only 29.5% the size of the original graphs. (2) CBSR is efficient. On large-scale graph ED with 7M vertices and 129M edges, it can construct the index in 261.7s while TBP fails to construct the index in 24 hours. (3) CBSR is effective. Using the indices constructed by CBSR, on average SRPQ is $3.78\times$ faster than TopChain on graph ED.

IX. Conclusion

This paper proposes algorithms to answer the span-reachability problem on temporal bipartite graphs. Our contributions are listed as follows: (1) a graph transformation method that converts the span-reachability problem on temporal bipartite graphs to the reachability problem on DAGs; (2) a contraction-based indexing schema, which process large-scale graphs; and (3) a two-stage query algorithm for the span-reachability queries on temporal bipartite graphs. Experiments show that the methods are promising in practice.

One topic for future work is to develop an effective partition algorithm for traject graphs, to reduce border vertices.

REFERENCES

[1] The konect project, 2013. *http://konect.cc/*.

[2] Full version, 2025. *https://github.com/dengt2000/SRPfull/blob/main/SRPfull.pdf*.

[3] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *ESA*, pages 24–35, 2012.

[4] S. Al-Eidi, Y. Chen, O. Darwishand, and A. M. Alfosool. Time-ordered bipartite graph for spatio-temporal social network analysis. In *ICNC*, pages 833–838, 2020.

[5] K. Andreev and H. Räcke. Balanced graph partitioning. *Theory Comput. Syst.*, 39(6):929–939, 2006.

[6] A.-L. Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005.

[7] I. Benouaret and S. Amer-Yahia. A comparative evaluation of top-n recommendation algorithms: Case study with total customers. In *IEEE BigData*, pages 4499–4508, 2020.

[8] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Class-based graph anonymization for social network data. *Proc. VLDB Endow.*, 2(1):766–777, 2009.

[9] F. Bourse, M. Lelarge, and M. Vojnovic. Balanced graph edge partition. In *KDD*, pages 1456–1465, 2014.

[10] A. Casteigts, T. Corsini, and W. Sarkar. Simple, strict, proper, happy: A study of reachability in temporal graphs. *Theor. Comput. Sci.*, 991:114434, 2024.

[11] A. Casteigts, A. Himmel, H. Molter, and P. Zschoche. Finding temporal paths under waiting time constraints. In *ISAAC*, volume 181, pages 30:1–30:18, 2020.

[12] W. Chen, A. M. Rocha, W. Hendrix, M. C. Schmidt, and N. F. Samatova. The multiple alignment algorithm for metabolic pathways without abstraction. In *ICDMW*, pages 669–678, 2010.

[13] X. Chen, K. Wang, X. Lin, W. Zhang, L. Qin, and Y. Zhang. Efficiently answering reachability and path queries on temporal bipartite graphs. *Proc. VLDB Endow.*, 14(10):1845–1858, 2021.

[14] J. Cheng, S. Huang, H. Wu, and A. W. Fu. Tf-label: a topological-folding labeling scheme for reachability querying in a large graph. In *SIGMOD*, pages 193–204, 2013.

[15] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[16] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.

[18] A. Deligkas and I. Potapov. Optimizing reachability sets in temporal graphs by delaying. *Inf. Comput.*, 285:104890, 2022.

[19] H. Deng, M. R. Lyu, and I. King. A generalized co-hits algorithm and its application to bipartite graphs. In *SIGKDD*, pages 239–248, 2009.

[20] S. Eubank, H. Guclu, V. Anil Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180–184, 2004.

[21] W. Fan, R. Jin, P. Lu, C. Tian, and R. Xu. Towards event prediction in temporal graphs. *Proc. VLDB Endow.*, 15(9):1861–1874, 2022.

[22] W. Fan, Y. Li, M. Liu, and C. Lu. A hierarchical contraction scheme for querying big graphs. In *SIGMOD*, pages 1726–1740, 2022.

[23] W. Fan, Y. Li, M. Liu, and C. Lu. Making graphs compact by lossless contraction. *VLDB J.*, 32(1):49–73, 2023.

[24] W. Fan and C. Tian. Incremental graph computations: Doable and undoable. *TODS*, 47(2):6:1–6:44, 2022.

[25] L. Ferreri, E. Venturino, and M. Giacobini. Do diseases spreading on bipartite networks have some evolutionary advantage? In *EvoBio*, volume 6623, pages 141–146, 2011.

[26] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[27] K. Hanauer, C. Schulz, and J. Trummer. O'reach: Even faster reachability in large graphs. *JEA*, 27:1–27, 2022.

[28] D. He and P. Yuan. TDS: fast answering reachability queries with hierarchical traversal trees. *Clust. Comput.*, 28(3):178, 2025.

[29] B. H. Hong, J. Labadin, W. K. Tiong, T. Lim, M. H. L. Chung, et al. Modelling covid-19 hotspot using bipartite network approach. *Acta Informatica Pragensia*, 10(2):123–137, 2021.

[30] W. Huo and V. J. Tsotras. Efficient temporal shortest path queries on evolving social graphs. In *SSDBM*, pages 38:1–38:4, 2014.

[31] R. Jin, N. Ruan, Y. Xiang, and H. Wang. Path-tree: An efficient reachability indexing scheme for large directed graphs. *TODS*, 36(1), 2011.

[32] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *SIGMOD*, pages 813–826, 2009.

[33] R. Jin, Y. Xiang, N. Ruan, and H. Wang. Efficiently answering reachability queries on very large directed graphs. In *SIGMOD*, pages 595–608, 2008.

[34] B. Li, A. Maalla, and M. Liang. Research on recommendation algorithm based on e-commerce user behavior sequence. In *ICIBA*, volume 2, pages 914–918, 2021.

[35] X. Li, Y.-Y. Wang, and A. Acero. Learning query intent from regularized click graphs. In *SIGIR*, pages 339–346, 2008.

[36] H. Lu and S. Uddin. A weighted patient network-based framework for predicting chronic diseases using graph neural networks. *Scientific reports*, 11(1):22607, 2021.

[37] F. Merz and P. Sanders. Preach: A fast lightweight reachability index using pruning and contraction hierarchies. In *ESA*, volume 8737, pages 701–712. Springer, 2014.

[38] L. Meunier and Y. Zhao. Reachability queries on dynamic temporal bipartite graphs. In *SIGSPATIAL/GIS*, 2023.

[39] S. Seufert, A. Anand, S. Bedathur, and G. Weikum. Ferrari: Flexible and efficient reachability range assignment for graph indexing. In *ICDE*, pages 1009–1020, 2013.

[40] H. Shirani-Mehr, F. B. Kashani, and C. Shahabi. Efficient reachability query evaluation in large spatiotemporal contact datasets. *Proc. VLDB Endow.*, 5(9):848–859, 2012.

[41] A. G. Smart, L. A. Amaral, and J. M. Ottino. Cascading failure and robustness in metabolic networks. *Proceedings of the National Academy of Sciences*, 105(36):13223–13228, 2008.

[42] J. Su, Q. Zhu, H. Wei, and J. X. Yu. Reachability querying: Can it be even faster? *TKDE*, 29(3):683–697, 2016.

[43] Z. Su, D. Wang, X. Zhang, L. Cui, and C. Miao. Efficient reachability query with extreme labeling filter. In *WSDM*, pages 966–975, 2022.

[44] E. Tacchini, G. Ballarin, M. L. D. Vedova, S. Moret, and L. de Alfaro. Some like it hoax: Automated fake news detection in social networks. *CoRR*, abs/1704.07506, 2017.

[45] B. Tesfaye, N. Augsten, M. Pawlik, M. H. Böhlen, and C. S. Jensen. Speeding up reachability queries in public transport networks using graph partitioning. *Inf. Syst. Frontiers*, 24(1):11–29, 2022.

[46] S. Thejaswi, J. Lauri, and A. Gionis. Restless reachability problems in temporal graphs. *arXiv preprint arXiv:2010.08423*, 2020.

[47] S. Trißl and U. Leser. Fast and practical indexing and querying of very large graphs. In *SIGMOD*, page 845–856, 2007.

[48] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. FENNEL: streaming graph partitioning for massive scale graphs. In *WSDM*, pages 333–342, 2014.

[49] D. L. Venkatraman, D. Pulimamidi, H. G. Shukla, and S. R. Hegde. Tumor relevant protein functional interactions identified using bipartite graph analyses. *Scientific Reports*, 11(1):21530, 2021.

[50] K. Wang, M. Cai, X. Chen, X. Lin, W. Zhang, L. Qin, and Y. Zhang. Efficient algorithms for reachability and path queries on temporal bipartite graphs. *The VLDB Journal*, pages 1–28, 2024.

[51] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou. Efficient route planning on public transportation networks: A labelling approach. In *SIGMOD*, page 967–982, 2015.

[52] H. Wei, J. X. Yu, C. Lu, and R. Jin. Reachability querying: An independent permutation labeling approach. *Proc. VLDB Endow.*, 7(12):1191–1202, 2014.

[53] D. Wen, Y. Huang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Efficiently answering span-reachability queries in large temporal graphs. In *ICDE*, pages 1153–1164, 2020.

[54] D. Wen, B. Yang, Y. Zhang, L. Qin, D. Cheng, and W. Zhang. Span-reachability querying in large temporal graphs. *VLDB J.*, 31(4):629–647, 2022.

[55] M. Wijanto, R. Rahmadiany, and O. Karnalim. Thesis supervisor recommendation with representative content and information retrieval. *JISEBI*, 6:143–150, 10 2020.

[56] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, 2014.

[57] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke. Reachability and time-based path queries in temporal graphs. In *ICDE*, pages 145–156, 2016.

[58] H. Xie, Y. Fang, Y. Xia, W. Luo, and C. Ma. On querying connected components in large temporal graphs. *Proc. ACM Manag. Data*, 1(2):170:1–170:27, 2023.

[59] H. Yildirim, V. Chaoji, and M. J. Zaki. Grail: Scalable reachability index for large graphs. *Proc. VLDB Endow.*, 3(1–2):276–284, 2010.

[60] C. Yu, T. Ren, W. Li, H. Liu, H. Ma, and Y. Zhao. BL: an efficient index for reachability queries on large graphs. *IEEE Trans. Big Data*, 10(2):108–121, 2024.

[61] J. Yu, X. Zhang, H. Wang, X. Wang, W. Zhang, and Y. Zhang. FPGN: follower prediction framework for infectious disease prevention. *WWW*, 26(6):3795–3814, 2023.

[62] J. X. Yu and J. Cheng. Graph reachability queries: A survey. In *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 181–215. 2010.

[63] P. Yuan, Y. You, S. Zhou, H. Jin, and L. Liu. Providing fast reachability query services with mgtag: A multi-dimensional graph labeling method. *IEEE Trans. Serv. Comput.*, 15(2):1000–1011, 2022.

[64] C. Zhang, A. Bonifati, and M. T. Özsu. An overview of reachability indexes on graphs. In *SIGMOD conference Companion*, SIGMOD '23, page 61–68, 2023.

[65] T. Zhang, Y. Gao, L. Chen, W. Guo, S. Pu, B. Zheng, and C. S. Jensen. Efficient distributed reachability querying of massive temporal graphs. *VLDB J.*, 28(6):871–896, 2019.

[66] R. Zhao and Q. Liu. Dynamical behavior and optimal control of a vector-borne diseases model on bipartite networks. *Applied Mathematical Modelling*, 102:540–563, 2022.

[67] J. Zhou, S. Zhou, J. X. Yu, H. Wei, Z. Chen, and X. Tang. DAG reduction: Fast answering reachability queries. In *SIGMOD*, pages 375–390, 2017.

APPENDIX

A1. Proof of Theorem 2

($\Rightarrow$) Assume that vertex u can reach v in G in time interval $[t_s, t_e]$. Then there exists a temporal path $\rho = u_1v_1u_2v_2\ldots u_kv_ku_{k+1}$ with $u = u_1$ and $v = u_{k+1}$ such that the starting time and ending time of ρ fall in $[t_s, t_e]$. Moreover, for each $i \in [1, k-1]$ there exist two consecutive connections $\langle u_i, e_1^i, v_i, e_2^i, u_{i+1}\rangle$ and $\langle u_{i+1}, e_1^{i+1}, v_{i+1}, e_2^{i+1}, u_{i+2}\rangle$ in G satisfying that (1) the starting time of two consecutive connections increase, *i.e.,* $\Theta(e_1^i, e_2^i).t_s \leq \Theta(e_1^{i+1}, e_2^{i+1}).t_s$; and (2) the timestamps in two adjacent edges of top vertex u_{i+1} increase; that is, (a) when $v_i \neq v_{i+1}$, then $e_2^i.t_e \leq e_1^{i+1}.t_s$; and (b) when $v_i = v_{i+1}$, then $e_2^i.t_s \leq e_1^{i+1}.t_s$.

Given the temporal path ρ, we construct a path from src_u to dest_v in the traject graph $\mathcal{G}_G$ as follows: for any two consecutive connections $\langle u_i, e_1^i, v_i, e_2^i, u_{i+1}\rangle$ and $\langle u_{i+1}, e_1^{i+1}, v_{i+1}, e_2^{i+1}, u_{i+2}\rangle$, there exist four snapshots constructed from the starting time and ending time of these connections. That is, (a) $s_1 = [\Theta(e_1^i, e_2^i).t_s, \Theta(e_1^i, e_2^i).t_s]_{v_i}$ and $s_2 = [\Theta(e_1^i, e_2^i).t_e, \Theta(e_1^i, e_2^i).t_e]_{v_i}$ from the first connection, and (b) $s_3 = [\Theta(e_1^{i+1}, e_2^{i+1}).t_s, \Theta(e_1^{i+1}, e_2^{i+1}).t_s]_{v_{i+1}}$ and $s_4 = [\Theta(e_1^{i+1}, e_2^{i+1}).t_e, \Theta(e_1^{i+1}, e_2^{i+1}).t_e]_{v_{i+1}}$ from the second connection. Moreover, there exists a path $s_1s_2s_3s_4$ in $\mathcal{G}_G$.

(1) We start with the path from s_1 to s_2. Because both s_1 and s_2 are constructed from the connection $\langle u_i, e_1^i, v_i, e_2^i, u_{i+1}\rangle$, the timestamps of s_1 and s_2 are covered by both edges e_1^i and e_2^i. Then there exist a path from s_1 and s_2 constructed from e_1^i and e_2^i, due to the construction of $\mathcal{G}_G$. Observe that there may exist multiple snapshots constructed from one connection (see Example 7). Similarly for the path from s_3 to s_4.

(2) We next construct the path from s_2 to s_3. Since snapshots s_2 and s_3 are constructed from two edges of the same top vertex u_{i+1}, we can collect all edges $(u_{i+1}, v_1, t_s^{12}, t_e^{12}), (u_{i+1}, v_2, t_s^{22}, t_e^{22}), \ldots, (u_{i+1}, v_k, t_s^{k2}, t_e^{k2})$ of top vertex u_{i+1} in the time interval $[\Theta(e_1^i, e_2^i).t_e, \Theta(e_1^{i+1}, e_2^{i+1}).t_s]$. Let $t_s^{12} \leq \ldots \leq t_s^{k2}$. Then there exists a path from snapshot $[t_s^{12}, t_s^{12}]_{v_1}$ to snapshot $[t_s^{k2}, t_s^{k2}]_{v_k}$, to simulate the movements of a top vertex u_{i+1}. Moreover, there exist an edge from s_2 to $[t_s^{12}, t_s^{12}]_{v_1}$, and an edge from $[t_s^{k2}, t_s^{k2}]_{v_k}$ to s_3, since both of these snapshots are constructed from edges of top vertex u_{i+1}. Therefore, there is a path from s_2 to s_3 in $\mathcal{G}_G$.

Therefore, $([t_s^1, t_e^1]_{v_1}, \ldots, [t_s^n, t_e^n]_{v_n})$ is a path in $\mathcal{G}_G$, with $\mathsf{src}_u = [t_s^1, t_e^1]_{v_1}$ and $\mathsf{dest}_v = [t_s^n, t_e^n]_{v_n}$.

($\Leftarrow$) Assume that snapshot src_u can reach snapshot dest_v in $\mathcal{G}_G$. Let $P_{\mathcal{G}} = (s_1, s_2, \ldots, s_{n-1}, s_n)$ be such a path with $s_1 = \mathsf{src}_u$ and $s_n = \mathsf{dest}_v$ in $\mathcal{G}_G$. For each edge from s_i to s_{i+1} ($i \in [1, n-1]$) in $\mathcal{G}_G$, we construct multiple connections forming a temporal path in G.

(1) We start with i=1. There exists an edge from s_1=$[t_s^1, t_e^1]_{v_1}$ to $s_2 = [t_s^2, t_e^2]_{v_2}$. Let (e_s^1, e_e^1) (resp. (e_s^2, e_e^2)) be a pair of edges whose timestamps are used to construct snapshot s_1 (resp. s_2). Assume that $e_s^1 = (u_2, v_1, t_s^{11}, t_e^{11})$, $e_e^1 = (u_3, v_1, t_s^{12}, t_e^{12})$, $e_s^2 = (u_3, v_2, t_s^{21}, t_e^{21})$ and $e_e^2 = (u_4, v_2, t_s^{22}, t_e^{22})$. Observe that (a) the first edge e_s^1 may not be an edge of u_1, since u_1 can link to v_1 before timestamp t_s^1 (see the definition of the span-reachability problem); here u_1 is the given top vertex u; (b) the edge from s_1 to s_2 are constructed due to top vertex u_3; (c) the edges e_e^1 and e_s^2 may be the same edge of u_3 (see Cases b.1 and b.2 in Figure 2); and (d) from the definition of edges in $\mathcal{G}_G$, the following condition holds: $s_1.t_e \leq s_2.t_s$.

(I) We start with the connections from u_2 to u_4. We construct the following path: $u_2v_1u_3v_2u_4$. This is a valid temporal path, since $\Theta(e_s^1, e_e^1).t_s \leq \Theta(e_s^1, e_e^1).t_e = s_1.t_e \leq s_2.t_s = \Theta(e_s^2, e_e^2).t_s$ (*i.e.,* the starting time of the sequential connections increase), and $e_e^1.t_e$=$\Theta(e_s^1, e_e^1).t_e$=$s_1.t_e$$\leq$$s_2.t_s$ = $\Theta(e_s^2, e_e^2).t_s$$\leq$$e_s^2$ (*i.e.,* timestamps of e_e^1 and e_s^2 of u_3 increase).

It remains to construct a connection between u_1 and u_2. From $s_1 = \mathsf{src}_u$, we know that u_1 and u_2 link to v_1 during interval $[t_s^1, t_e^1]$, and there exists an edge $e_s^0 = (u_1, v_1, t_s^{01}, t_e^{01})$ such that $\langle u_1, e_s^0, v_1, e_s^1, u_2\rangle$ is a connection with $t_s^{01} \leq t_s^{11}$ and $t_s^{11} \leq t_e^{01}$, *i.e.,* the timestamps of e_s^0 and e_s^1 are overlap.

Therefore, we construct the temporal path: $u_1, v_1, u_2, v_1, u_3, v_2, u_4$ to replace the first edge in $P_{\mathcal{G}}$.

(II) We can similarly construct other connections for the edge from s_i to s_{i+1} with $i \in [2, n-1]$. Therefore, we can deduce a temporal path from u to v. we omit the details here. □