A Contraction Framework for Reachability Queries on Temporal Bipartite Graphs

Lijun Sun[1,2], Junlong Liao[1], Ting Deng[1], Ping Lu[1], Jianxin Li[1,2], Xiangping Huang[3], Zhongyi Liu[3], Richong Zhang[1,2]
[1]Beihang University [2]Zhongguancun Laboratory [3]TravelSky Technology Limited
{sunlijun1,liaojunlong,dengting,luping,lijx,zhangrichong}@buaa.edu.cn,{xphuang,liuzy}@travelsky.com.cn

ABSTRACT

Temporal bipartite graphs model interactions between two distinct entity types over time, facilitating applications in disease control, supply chain management, and transportation system optimization, etc. This paper focuses on the span-reachability problem. It is hard to exactly characterize the reachability in temporal bipartite graphs. Existing definitions suffer from insufficient temporal resolution, and may miss some solutions. To address this, we propose a new span-reachability queries based on a refined notion of temporal path, and a contraction-based indexing framework CBSR to efficiently answer such queries. CBSR transforms temporal bipartite graphs into directed acyclic graphs (DAGs) while preserving the reachability relation between vertices, partitions the DAG into fragments for space optimization, and constructs indices for these fragments via a contraction strategy. Utilizing real-life data, we experimentally show that on average CBSR outperforms the state-of-the-art algorithms by 1245.32× in constructing indices, and 1.99× in querying.

1 INTRODUCTION

Bipartite graphs (bigraphs) are specialized graphs in which the vertices are divided into two disjoint sets, commonly referred to as top and bottom vertices. Edges only connect vertices from different sets. This structure represents interactions between entities of distinct types, and can be extended with temporal information to capture the time of these interactions.

Various reachability problems have been studied on temporal graphs, including temporal reachability [10, 12, 45], restless temporal reachability [11, 16, 46], shortest path distance [30, 55, 56] and span-reachability [12, 53, 57, 64]. In this work, we focus on the span-reachability query (*a.k.a.* temporal reachability query in [64]), which determines whether there exists a temporal path between two vertices within a given time interval. This fundamental problem has critical applications in diverse domains, including disease prevention and contact tracing [12, 17, 22, 29, 65], stock market supervision [50], recommendation systems [38], transport system optimization [24, 26] and fake news detection [37, 44].

For example, (1) in contact tracing, temporal bigraphs can model movements of individuals across locations, and span-reachability analysis can identify potential infectors (see Example 1.1). (2) In stock trading networks, temporal bigraphs model trades between investors and stocks [50], and reachability analysis can detect price ramping [5], where multiple accounts controlled by the same trader purchase shares of the same broker within a short time to intentionally inflates its price [4, 34]. Span-reachability reveals these manipulations by detecting that one account can reach another in a short time. (3) In recommendation systems, temporal bigraphs encode user-item interactions [38], and reachability analysis can identify customers with similar temporal interests or habits [25, 39].

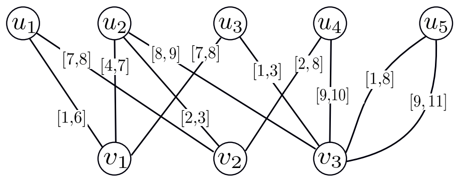

Figure 1: A temporal bipartite graph G

However, it is not easy to correctly characterize the span-reachability in temporal bigraphs. Consider the following example.

Example 1.1. Figure 1 shows a temporal bigraph, which models a disease contact tracing scenario. The top vertices u_1-u_5 correspond to five individuals, and the bottom vertices v_1-v_3 represent three locations. Each edge is labeled with a time interval $[t_1, t_2]$, indicating the period during which an individual visits a location.

Consider a disease transmission scenario where u_2 becomes infected at timestamp 2. The infection can propagate along the following temporal contact chain: (1) During interval [2,3], both u_2 and u_4 are present at location v_2, allowing potential transmission from u_2 to u_4. (2) After that both u_4 and u_1 are present at location v_2 during interval [7,8], enabling transmission from u_4 to u_1. These two interactions form a transmission chain: $u_2 \rightarrow u_4 \rightarrow u_1$.

However, existing algorithms such as TBP [12] fail to identify this transmission chain due to their coarse-grained definition of "contact". They define the duration of the contact between two individuals at a location as the union of time intervals during which the individuals visit the location, *i.e.,* both the contact between u_2 and u_4, and the contact between u_4 and u_1, occurring at location v_2 are within the same time interval $[2, 8]$. Under the time constraint for a temporal path defined in TBP (*i.e.,* the end time of the former contact is no larger than the start time of the latter contact), these two contacts cannot form a transmission chain, *i.e.,* such coarse-grained definition deviates from real-world transmission scenarios. □

The above example illustrates that existing methods for span-reachability problem suffer from insufficient temporal resolution, which motivates us to propose a new definition of span-reachability query based on a refined notion of temporal contact. This new definition preserves fine-grained temporal information, enabling more accurate span-reachability analysis in temporal bigraphs.

Contributions. Our contributions are summarized as follows.

(1) *Fine-grained definition for span-reachability* (Section 3). We propose a fine-grained definition for temporal paths to correctly answer the span-reachability in temporal bigraphs. Unlike existing

temporal path definition, we partition the time interval of each edge into multiple segments, and ensure that the time intervals of these segments increase along the path. Two consecutive edges in the the path connect to the same bottom vertex, the time interval is defined as the intersection of their intervals instead of the union. Based on such paths, we define the span-reachability problem, which allows us to correctly identify temporal path such as $u_2 \rightarrow u_4 \rightarrow u_1$ in Example 1.1 (see Example 3.2 for details).

(2) Graph transformation (Section 5). We propose a transformation method to convert temporal bigraphs into directed acyclic graphs (DAGs). We introduce the concept of snapshots to compactly capture reachability information, such that the DAG maintains all critical information to answer the span-reachability queries. Moreover, such transformation may also reduce the graph sizes, which in turn reduces the index sizes and accelerates the queries (see Section 8).

(3) Index construction (Section 6). We introduce a contraction-based indexing approach for the constructed DAG. In contrast to existing indices for reachability, we partition the DAG into multiple fragments, and constructs both intra- and inter-fragment indices. In this way, we can reduce the size of the indices, and process large-scale graphs. Moreover, we develop an algorithm to determine the appropriate number of fragments for the span-reachability queries.

(4) Query algorithm (Section 7). We show that existing algorithms for reachability on DAGs can be adapted with minimum changes to answer span-reachability. We develop an algorithm SRPQ for span-reachability by extending algorithm TopChain [56] as a case study, and show that SRPQ retains the same complexity of TopChain.

(5) Experimental study (Section 8). Using real-life datasets, we experimentally find the following. (1) Using real-life queries such as contact tracing [17], CBSR correctly answers all queries due to the fine-grained definition, while on average the accuracy of TBP achieves only 75.48%. (2) On average the graph transformation method can generate a DAG that is only 29.52% the size of the original graph and preserves the reachability relations among vertices. (3) CBSR is efficient in index construction. On average, CBSR is 1245.32× faster than TBP. On a graph with 6.83M vertices and 129.88M edges, CBSR constructs the index in less than 600s, while the state-of-the-art algorithm TBP [12] fails to process the graph. (4) On average, SRPQ is 1.99× faster than the existing algorithm TopChain [56].

Organization. This paper is organized as follows. We introduce the necessary notations in Section 3 and give an overview of the solution in Section 4. Then we present algorithms for graph transformation, index construction and querying in Sections 5, 6 and 7, respectively. Experimental results are presented in Section 8.

2 RELATED WORK

We categorize the related work as follows.

General graph. The reachability problem in general graphs has been extensively studied. It is to determine whether there is a path between two vertices in a graph. Classical algorithms for this problem include depth-first search (DFS) and breadth-first search (BFS). However, these methods do not scale well with large-scale graphs [33], motivating the development of various indexing methods to accelerate the computation: (1) tree-cover based methods (*e.g.,* path-tree [31], gripp [47] and grail [58]), which construct (a) a tree to capture most reachability information, and (b) a supplementary index table for the remaining information; (2) hop-based methods (*e.g.,* 2-hop [14], 3-hop [32], HHL [3], O'Reach [27], BL [59] and TF-Label [13]), which pick a set of vertices as the landmarks and answer reachability queries using the landmarks as bridges; (3) pruned-based methods (*e.g.,* IP [52], BFL [42], PReaCH [36] and ELF [43]), which compute some conditions on vertices to filter out unreachable queries; and (4) contraction-based methods (*e.g.,* [66]), which convert a graph into a DAG in which each vertex represents a strongly connected component. See [61, 63] for more discussion.

Closer to our work, MGTag [62] and TDS [28] support reachability queries on DAGs. MGTag is a graph labeling method that recursively partitions graphs into multiple fragments and computes four-dimensional labels for indexing. TDS further extends MGTag with more reasonable dimension selection.

Discussion. This work differs from prior work as follows. (1) We target the reachability problem on temporal bigraphs, which is more challenging than the reachability problem on general graphs [28, 62, 66]. (2) Using contracted graphs, we can exploit existing algorithms to conduct queries, rather than developing a new one. (3) To construct inter-fragment indices, we merge vertices connecting to the same set of border vertices and use trie data structure to reduce the number of border vertices, which are not exploited in [28, 62].

Temporal graphs. The reachability problem has also been studied on temporal graphs [41, 51, 53, 54, 56, 64]. ReachGraph [41] transforms temporal graphs into DAGs by materializing connected components at each timestamp. TTL [51] extends hop-based methods to temporal graphs and adopts an index partitioning strategy to handle consecutive time intervals. TopChain [56] also transforms temporal graphs into DAGs, decomposes the DAG into a set of chains, and extends the hop-based methods to construct indices. TVL [64] proposes TVL indices for the reachability queries. TILL [53, 54] extends the 2-hop method with timestamps to answer the span-reachability.

Closer to this work are TBP [12], WTB [37] and RQ [45]. TBP focuses on the span-reachability problem (*i.e.,* the single-pair reachability problem) on temporal bigraphs and devises TBP-indices to accelerate query processing. WTB inherits the definition of temporal path from TBP, but targets the global approximate reachability problem. RQ adopts a partition-based strategy to construct indices for reachability queries under budget consraints.

Discussion. In contrast to prior work, (1) we adopt a different definition of temporal paths, which can answer the reachability problem more accurate than TBP, WTB and RQ (see Section 3 and **Exp-1** in Section 8). (2) We compute equivalence relations to reduce the size of graphs, and exploit the trie data structure to reduce used memory, which is not studied in TBP, WTB or RQ. (3) We partition the DAG into multiple fragments to process large-scale graphs, and design an algorithm to determine an appropriate number of fragments for span-reachability queries, which is not considered in TBP, WTB or RQ. (4) We develop an algorithm to convert temporal bigraphs into DAGs. Unlike ReachGraph [41] and TopChain [56], where the resulting DAGs remain similar in size to the temporal graphs, our DAGs can be significantly smaller, since a single edge in the DAG may represent multiple edges in the temporal graphs (*e.g.,* a group of people moving simultaneous from one place to another).

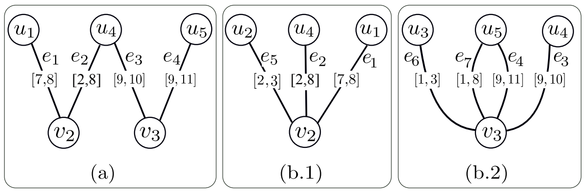

Figure 2: Examples of temporal paths of graph G in Figure 1

3 PRELIMINARY

Assume a countably infinite set Ω of timestamps with a discrete total order $\leq$. A time interval is a range $[t_1, t_2]$ with $t_1, t_2 \in \Omega$ and $t_1 \leq t_2$. Let $\mathcal{T} \subseteq \Omega \times \Omega$ denote the set of all time intervals.

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 referred to as 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 $[t_s, t_e]$. Denote the *start time* and *end time* of e by $e.t_s$ and $e.t_e$, respectively.

In this paper, we consider multi-bigraphs, *i.e.,* there may exist multiple edges between a top vertex in U and a bottom vertex in L; *e.g.,* a person visit the same location at different time intervals.

Intersection. Given two edges e_1 and e_2, we define the *intersection* of their time intervals, representing, for example, the temporal contact between two individuals. Specifically, two edges $e_1 = (u_1, v_1, t_s^1, t_e^1)$ and $e_2 = (u_2, v_2, t_s^2, t_e^2)$ in E are said to intersect if $\min(t_e^1, t_e^2) \geq \max(t_s^1, t_s^2)$. The intersection of their time intervals, denoted by $\Theta(e_1, e_2)$, is defined as $\Theta(e_1, e_2) = [\max(t_s^1, t_s^2), \min(t_e^1, t_e^2)]$. If e_1 and e_2 do not intersect, we denote $\Theta(e_1, e_2) = \emptyset$.

Example 3.1. For the temporal bigraph G in Figure 1, edges $e_1=(u_1, v_2, 7, 8)$ and $e_2=(u_4, v_2, 2, 8)$ intersect with $\theta(e_1, e_2) = [7, 8]$, while e_1 and $e_3 = (u_4, v_3, 9, 10)$ do not intersect and $\theta(e_1, e_3) = \emptyset$.

Temporal Path. Given a temporal bigraph $G = (U \cup L, E)$ and two top vertices u and u' in U, a *temporal path* ρ from u to u' of length $2k$ is defined as a sequence of edges $\rho = e_1 e_2 \ldots e_{2k}$ where $e_i = (u_i, v_i, t_s^i, t_e^i)$, which satisfy the following four conditions:

(1) $u_1 = u$ and $u_{2k} = u'$;

(2) for each two consecutive edges $e_{2i+1}=(u_{2i+1}, v_{2i+1}, t_s^{2i+1}, t_e^{2i+1})$ and $e_{2i+2}=(u_{2i+2}, v_{2i+2}, t_s^{2i+2}, t_e^{2i+2})$ of ρ with $i \in [0, k-1]$, they share the same bottom vertex (*i.e.,* $v_{2i+1} = v_{2i+2}$) and their time intervals intersect, *i.e.,* $\Theta(e_{2i}, e_{2i+2}) \neq \emptyset$ (see e_1 and e_2 in Fig. 2(a));

(3) for each two consecutive edges $e_{2i} = (u_{2i}, v_{2i}, t_s^{2i}, t_e^{2i})$ and $e_{2i+1} = (u_{2i+1}, v_{2i+1}, t_s^{2i+1}, t_e^{2i+1})$ with $i \in [1, k-1]$, they are incident to the same top vertex (*i.e.,* $u_{2i} = u_{2i+1}$), and the timestamps of e_{2i} and e_{2i+1} increases; more specifically, (a) if $v_{2i} \neq v_{2i+1}$, then $e_{2i}.t_e \leq e_{2i+1}.t_s$ (*e.g.,* top vertex u_{2i} traverses from one bottom vertex v_{2i} to another bottom vertex v_{2i+1}; see e_2 and e_3 in Figure 2(a) for an example); otherwise, (b) either $e_{2i} = e_{2i+1}$ (*e.g.,* top vertex u_{2i} "stays at" the same bottom vertex v_i; see e_2 in Figure 2(b.1)), or $e_{2i}.t_e \leq e_{2i+1}.t_s$ (*i.e.,* top vertex u_{2i} "leaves" and "reenters" the same bottom vertex v_{2i}; see e_7 and e_4 in Figure 2(b.2)); and

(4) for each $i \in [1, 2k-4]$ and any four consecutive edges $e_i, e_{i+1}, e_{i+2}, e_{i+3}$, where $e_{i+j}=(u_{i+j}, v_{i+j}, t_s^{i+j}, t_e^{i+j})$ with $j \in [0, 3]$, if they share the same bottom vertex (*i.e.,* $v_i=v_{i+1}=v_{i+2}=v_{i+3}$), and e_{i+1} and e_{i+2} are the same edge (see Figure 2(b.1)), the end time of the intersection $\Theta(e_i, e_{i+1})$ is no larger than the start time of $\Theta(e_{i+2}, e_{i+3})$.

Intuitively, condition (4) requires that u_i first meets u_{i+1} at "location" v_i during the interval $\Theta(e_i, e_{i+1})$, and then u_{i+1} (*i.e.,* u_{i+2}) meets u_{i+3} at the same "location" v_i during a *subsequent interval* $\Theta(e_{i+2}, e_{i+3})$. That is, u_{i+1} must meet u_i before meeting u_{i+3}. Without this condition, one could construct a path $e_{i+3}e_{i+2}e_{i+1}e_i$, implying that u_{i+3} meets u_{i+2} (*i.e.,* u_{i+1}) at v_i during $\Theta(e_{i+3}, e_{i+2})$, and *later* u_{i+1} (*i.e.,* u_{i+2}) meets u_i at v_i during $\Theta(e_i, e_{i+1})$. This leads to a contradiction because the end time of the intersection $\Theta(e_i, e_{i+1})$ is no larger than the start time of $\Theta(e_{i+2}, e_{i+3})$. We can verify that condition (4) is not covered by conditions (1)-(3) above.

Example 3.2. Figure 2 shows temporal paths ρ_1, ρ_2, ρ_3 in Figure 1.

(i) $\rho_1=e_1e_2e_3e_4$ is a temporal path from u_1 to u_5, since (a) edges $e_1=(u_1, v_2, 7, 8)$ and $e_2=(u_4, v_2, 2, 8)$ intersect with $\Theta(e_1, e_2) = [7, 8]$; (b) edges e_2 and $e_3=(u_4, v_3, 9, 10)$ share the same top vertex u_4, and links to two bottom vertices v_2 and v_3, and $e_2.t_e \leq e_3.t_s$ (*i.e.,* $8 < 9$); and (c) edges e_3 and $e_4=(u_5, v_3, 9, 11)$ intersect and $\Theta(e_3, e_4)=[9, 10]$.

(ii) $\rho_2 = e_5e_2e_2e_1$ is a temporal path from u_2 to u_1, which satisfies that (a) edges $e_5 = (u_2, v_2, 2, 3)$ and $e_2 = (u_4, v_2, 2, 8)$ intersect with $\Theta(e_5, e_2) = [2, 3]$, and (b) edges e_2 and e_1 intersect with $\Theta(e_2, e_1) = [7, 8]$; and (c) these edges share the same bottom vertex v_2, edge e_2 appears twice in the path, and the end time of $\Theta(e_5, e_2)$ is not larger than the start time of $\Theta(e_2, e_1)$, *i.e.,* $3 \leq 7$.

(iii) $\rho_3=e_6e_7e_4e_3$ is a path from u_3 to u_4, such that (a) $e_6=(u_3, v_3, 1, 3)$ and $e_7=(u_5, v_3, 1, 8)$ intersect with $\Theta(e_6, e_7)=[1, 3]$; (b) e_7 and e_4 share the same top and bottom vertices, and $e_7.t_e \leq e_4.t_s$ (*i.e.,* $8 \leq 9$); (c) e_4 and e_3 intersect with $\Theta(e_3, e_4)=[9, 10]$; and (d) these edges share the same bottom vertex v_3, u_5 has two edges to v_3, and the end time of $\Theta(e_6, e_7)$ is less than the start time of $\Theta(e_4, e_3)$, *i.e.,* $3 \leq 9$.

Without condition (4), some edge sequences may be mistakenly identified as temporal paths, thus misrepresenting reachability. For example, the edge sequence $e_1e_2e_2e_5$ with edges e_1, e_2 and e_5 given in Figure 2(b.1), satisfies conditions (1)-(3), but not condition (4). It implies that u_1 is in contact with u_4 during $[7, 8]$, and then u_4 is in contact with u_2 during $[2, 3]$, which contradicts the required temporal order and cannot be regarded as a valid temporal path. □

The temporal paths are asymmetric, since the timestamps of two consecutive edges in a temporal path increases. Consequently, the existence of a temporal path from u to u' during interval I does not imply the existence of a path from u' to u within the same interval.

Span-Reachability. We say that one top vertex u can reach another top vertex u' in G during time interval $I = [t_s, t_e]$ if there exists a temporal path $\rho = e_1e_2 \ldots\ldots e_{2k}$ from u to u' such that the end time of $\Theta(e_1, e_2)$ and the start time of $\Theta(e_{2k-1}, e_{2k})$ fall in the time interval I. That is, (1) the end time of the intersection $\Theta(e_1, e_2)$ is no less than the start time of I, and (2) the start time of $\Theta(e_{2k-1}, e_{2k})$ is no larger than the end time of I. We use the end time of the first two edges and the start time of the last two edges to ensure that all edges in ρ are within interval I, which result in more temporal paths between vertices, and enables more precise contact tracing.

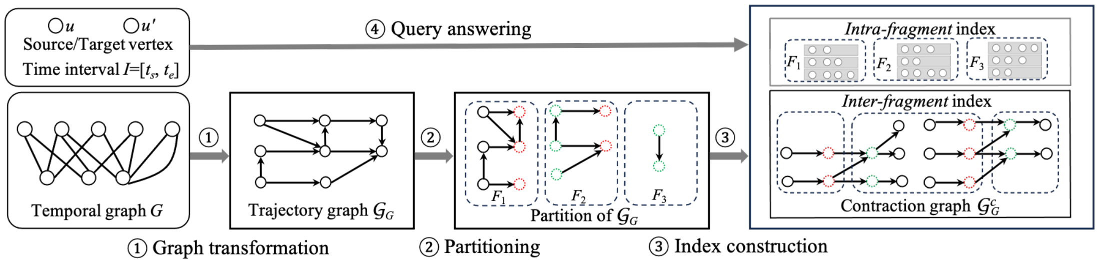

Figure 3: The framework of CBSR. (1) Construct a DAG $\mathcal{G}_G$ from temporal bipartite G, to transform the span-reachability problem on G into a reachability problem on $\mathcal{G}_G$; (2) partition $\mathcal{G}_G$ into multiple fragments; (3) build intra-fragement indices for each fragment and inter-fragment indices on the contraction graph $\mathcal{G}_G^c$ formed by border vertices; and (4) given source vertex u, target vertex u' and interval I, use intra-fragment indices if they are in the same fragment, and inter-fragment indices otherwise.

Table 1: Notations

Notations	Definitions
$G = (U \cup L, E)$	an undirected temporal bigraph
$Q(u, u', G, I)$	the span-reachability query
$\mathcal{G}_G/\mathcal{G}_G^c$	trajectory/contraction graph of temporal bigraph G
$\Theta(e_1, e_2)$	intersection of time intervals of e_1 and e_2
$[t_1, t_2]_v$	a snapshot of a bottom vertice v
$\mathcal{S}(G)/\mathcal{S}(u)$	the set of snapshots constructed in G/from vertex u

Example 3.3. Consider graph G in Fig. 1. Top vertex u_1 can reach u_5 during interval $I = [8, 9]$ via path ρ_1=$e_1e_2e_3e_4$ given in Example 3.2, since (1) $\Theta(e_1, e_2) = [7, 8]$ and $\Theta(e_3, e_4) = [9, 10]$, (2) the end time of $\Theta(e_1, e_2)$ is no less than the start time of I (*i.e.,* $8 \geq 8$), and (3) the start time of $\Theta(e_3, e_4)$ is no larger than the end time of I (*i.e.,* $9 \leq 9$). In fact, u_1 can reach u_5 during any interval that contains I. □

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 on the temporal bigraph G in interval I, where u and u' refer to the *source vertex* and the *target vertex*, respectively.

Remark. Our definition differs from the one in TBP [12] and WTB [37]. Specifically, for consecutive edges e_1 and e_2 incident to the same bottom vertex in a path, we define their time interval as the intersection of their individual intervals, rather than the union as in TBP and WTB. This intersection forms a subinterval common to e_1 and e_2, allowing the same edge to appear multiple times in a path. In contrast, under the union-based semantics in TBP and WTB, 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 TBP and WTB (see the following example).

Example 3.4. Consider path $\rho_2 = e_5e_2e_2e_1$ from Example 3.2.

(1) Path ρ_2 is not a valid path under the definitions of TBP and WTB [12, 37]. (a) Subpath e_5e_2, where e_5=($u_2, v_2, 2, 3$) and e_2=($u_4, v_2, 2, 8$), forms a wedge $\mathcal{W}_1 = (e_5, e_2)$ with start time 2 (*i.e.,* min($e_5.t_s, e_2.t_s$)=2) and end time 8 (*i.e.,* max($e_5.t_e, e_2.t_e$)=8) [12]. (b) Similarly, path e_2e_1 forms another wedge $\mathcal{W}_2$=(e_2e_1) with start time 2 and end time 8, where $e_1 = (u_1, v_2, 7, 8)$. (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 start time of $\mathcal{W}_2$ (*i.e.,* 2) is earlier than the end time of $\mathcal{W}_1$ (*i.e.,* 8).

(2) However, path ρ_2 is valid in the context of disease tracing. After being infected by u_2 during $[2, 3]$ at location v_2, u_4 remains at v_2, and infects u_1 during $[7, 8]$, though u_2 has already left v_2 at timestamp 3. Under our definition, such indirect propagation is permitted, and thus u_2 can reach u_1 during $[2, 8]$. Moreover, edge e_2 appears twice in path ρ_2, which is not allowed in TBP or WTB. □

Graph Partition. Given a number k and a graph $G = (V, E)$, we can partition G into k fragments $F_1, F_2, \ldots, F_k$ such that (1) $F_i = (V_i, E_i, \mathsf{IN}_i, \mathsf{OUT}_i)$ is a fragment; (2) $V = \cup_{i\in\{1,\ldots,k\}} V_i$; (3) $V_i \cap V_j = \emptyset$ for $i \neq j$; (4) IN_i is the set of vertices $v \in V_i$ with incoming edges from a vertex in $V \setminus V_i$; (5) OUT_i is the set of vertices $v \in V \setminus V_i$ with incoming edges from a vertex in V_i; and (6) E_i consists of all edges between vertices in $V_i \cup \mathsf{IN}_i \cup \mathsf{OUT}_i$. Such partitions are called edge-cut [8], *i.e.,* each vertex in V_i has all its adjacent edges in F_i as in 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.

A partition is ε-balanced, if $|V_i| \leq (1 + \varepsilon)\frac{|V|}{k}$ for each fragment F_i=($V_i, E_i, \mathsf{IN}_i, \mathsf{OUT}_i$), where $\varepsilon \geq 0$ is a threshold for the partitions.

The notations used in the paper are listed in Table 1.

4 SOLUTION OVERVIEW

In this section, we present an overview of a contraction-based framework for the SRP problem, denoted by CBSR (see Figure 3).

Architecture. Given a temporal bigraph G, CBSR first converts it into a DAG, denoted by $\mathcal{G}_G$. It then partitions $\mathcal{G}_G$ into multiple fragments, and constructs both intra- and inter-fragment indices for $\mathcal{G}_G$. Finally, these indices are used to design algorithm SRPQ to check whether one top vertex can reach another top vertex.

Step 1: Graph transformation. Given a temporal bigraph G, CBSR first constructs a DAG $\mathcal{G}_G$ from G, called the trajectory graph of G, such that an edge in $\mathcal{G}_G$ corresponds to multiple edges in G (Section 5). This transformation reduces the span-reachability problem on G to the standard reachability problem on $\mathcal{G}_G$. Most existing approaches for the reachability problem convert the graph into a DAG, where each vertex represents a strongly connected component (*e.g.,* [66]). Different from these approaches, CBSR partitions time intervals on edges to construct the DAG, and reduces the DAG sizes to optimize the reachability analysis.

Step 2: Graph partitioning. To handle large-scale graphs, CBSR partitions $\mathcal{G}_G$ into multiple fragments using a greedy strategy guided by temporal order of timestamps (Section 6.1). The number of fragments is refined via a binary search such that each fragment has a size comparable to the graph built on border vertices (see step 3), ensuring that answering queries have comparable performance for vertex pairs in the same fragment and across distinct fragments.

Step 3: Index construction. We construct both intra-fragement and inter-fragment indices to answer span-reachability queries (Section 6.2). For intra-fragment indexing, CBSR leverages existing algorithms (*e.g.,* [27, 31]). For inter-fragment indices, CBSR applies a graph contraction strategy to reduce graphs [20].

Step 4: Answering reachability queries on DAG. We develop a query algorithm SRPQ for SRP (Section 7). Given two top vertices u and u' in G as the source and target vertices, it first identifies the corresponding vertices $\mathsf{src_u}$ and $\mathsf{tgt_{u'}}$ in $\mathcal{G}_G$, and checks whether $\mathsf{src_u}$ and $\mathsf{tgt_{u'}}$ are in the same fragment. If so, intra-fragment indices are used to answer the query; otherwise inter-fragment ones are employed.

Property. (1) After reducing the span-reachability problem on temporal graphs G to the standard reachability problem on DAGs, we can exploit the existing algorithms to conduct the queries. Moreover, The DAGs can be much smaller than the original temporal graph G, which accelerates the computation (see **Exp-3** in Section 8).

(2) For large-scale graphs, on which existing methods [12, 37] fail to build indices due to memory limits, CBSR constructs both intra-fragment and inter-fragment indices after partitioning the graph into multiple fragments, which avoids building indices for vertices far from each other and reduces memory usage. Inter-fragment indices are used to check reachability for such vertices, to balance query efficiency and space consumption (See Sections 6 and 7).

(3) After partitioning the DAGs into multiple fragments, the algorithm SRPQ uses either intra-fragment or inter-fragment indices, to answer the queries. Since intra-fragment indices are smaller than the ones constructed directly on the whole DAGs, SRPQ get faster when the number of fragments increases (see **Exp-5** in Section 8). However, such number cannot be too large, otherwise the inter-fragment indices would be larger. Thus, we develop a strategy to determine the appropriate number of partitions for SRPQ (Section 6).

5 GRAPH TRANSFORMATION

In this section, we transform temporal bigraph G into a trajectory graph $\mathcal{G}_G$, which is a DAG and preserves the reachability relations between vertices. Here $\mathcal{G}_G$ is constructed based on top vertices, capturing their "movements" across different bottom vertices.

Challenge. To construct a DAG from a temporal bigraph, we need to address the following challenges: the temporal bigraph is undirected, contains timestamps, and may include cycles (see Figure 1). How can we derive a DAG from such a temporal graph, while preserving the reachability relationships between top vertices?

To solve these, we adopt the following strategies. (1) We introduce a data structure called snapshots, to capture time intervals during which a top vertex is connected to a bottom vertex. (2) Snapshots are then sorted in ascending chronological order, and linked according to the adjacent edges of top vertices, ensuring that the resulting graph is a DAG. Consequently, the reachability between top vertices can be determined by traversing paths in this DAG. Intuitively, if a top vertex u can reach another top vertex u' in the original temporal graph G, this reachability relation can be verified by tracing paths of u's corresponding vertex through the DAG.

Snapshots. We start with the definition of snapshots.

Definition. Given a temporal bigraph G=$(U \cup L, E)$, let $\mathcal{T}_G$ be the set of timestamps in G. For a bottom vertex $v \in L$ and any two timestamps t_1 and t_2 in $\mathcal{T}_G$, a *snapshot* of a bottom vertex v over interval $[t_1, t_2]$, denoted by $[t_1, t_2]_v$, is defined as a time interval during which there exist two adjacent edges e_1 and e_2 of v satisfying that their intersection $\Theta(e_1, e_2)$ contains $[t_1, t_2]$. The two top vertices incident to e_1 and e_2 are referred to as top vertices of the snapshot.

Construction. We can construct snapshots from each pair of adjacent edges of a bottom vertex. However, this method may introduce redundant edges in the resulting DAG, since multiple top vertices may be simultaneously connected to the same bottom vertex.

To address this, we construct snapshots based on the sorted timestamps of edges incident to each bottom vertex v. Let $t_1 < t_2 < \ldots < t_n$ be all timestamps of v. For any two consecutive timestamps t_i and t_{i+1}, we generate *candidate* snapshots $[t_i, t_i]_v$, $[t_i+1, t_{i+1}-1]_v$ and $[t_{i+1}, t_{i+1}]_v$ when $t_{i+1} - t_i > 1$, and $[t_i, t_i]_v$ and $[t_{i+1}, t_{i+1}]_v$ when $t_{i+1} = t_i + 1$. We next remove useless snapshots $[t', t'']_v$ such that no pair (e_1, e_2) of edges incident to v with $\Theta(e_1, e_2)$ contains $[t_1, t_2]$.

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

Example 5.1. Bottom vertex v_2 of temporal bigraph G in Figure 1 has three edges e_5, e_2 and e_1 (see Figure 2(b.1)), whose timestamps are 2, 3, 7 and 8, which yield candidate snapshots $[2, 2]_{v_2}$, $[3, 3]_{v_2}$, $[4, 6]_{v_2}$, $[7, 7]_{v_2}$ and $[8, 8]_{v_2}$. Among these edges in G, e_5 and e_2 intersect with $\Theta(e_5, e_2)$=$[2, 3]$, and e_2 and e_1 intersect with $\Theta(e_2, e_1)$=$[7, 8]$. Then we get snapshots $[2, 2]_{v_2}$ and $[3, 3]_{v_2}$ from edges e_5 and e_2, and $[7, 7]_{v_2}$ and $[8, 8]_{v_2}$ from e_2 and e_1. Therefore, candidate snapshot $[4, 6]_{v_2}$ is removed. Similarly, we get snapshots $[4, 4]_{v_1}$, $[5, 5]_{v_1}$, $[6, 6]_{v_1}$ and $[7, 7]_{v_1}$ of 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 $[10, 10]_{v_3}$ of v_3. □

Remark. (1) A snapshot may be constructed from multiple edge pairs. In Example 5.1, $[9, 9]_{v_3}$ is obtained from either (a) $e_8 = (u_2, v_3, 8, 9)$ and $e_4 = (u_5, v_3, 9, 11)$, or (b) e_8 and $e_3 = (u_4, v_3, 9, 10)$.

(2) The intersection of two edges can be split into multiple snapshots, to reduce the size of $\mathcal{G}_G$. Consider a graph G with k top vertices $u'_1, u'_2, \ldots, u'_k$, one bottom vertex v and k edges $e'_j = (u'_j, v, j, k + j)$, $j \in [1, k]$, *i.e.,* each top vertex u'_j has a single edge connected to the bottom vertex v with time interval $[j, k + j]$. There exist $k(k-1)/2$ many edge pairs (e'_i, e'_j) with $i < j$ and $\Theta(e'_i, e'_j) = [j, k+i]$. If we construct $\mathcal{G}_G$ using any edge pair, when k is large, the resulting DAG would be large. Instead, we construct $2k$ snapshots $[l, l + 1]_v$ ($l \in [1, 2k - 1]$), and link these snapshots to represent reachability relation among these top vertices, leading to smaller DAGs (see below).

Trajectory Graphs. We now define the trajectory graph for a temporal bigraph based on snapshots. For a temporal bigraph G=$(U \cup L, E)$ and snapshot set $\mathcal{S}(G)$, we define the trajectory graph

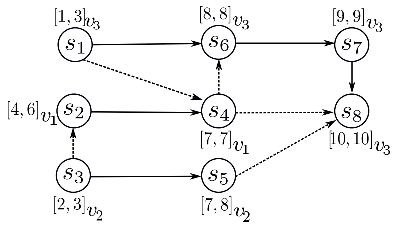

Figure 4: The trajectory graph $\mathcal{G}_G$ of graph G in Figure 1

$\mathcal{G}_G$=$(\mathcal{V}, \mathcal{E})$, where (1) $\mathcal{V}$=$\mathcal{S}(G)$, *i.e.,* each snapshot in $\mathcal{S}(G)$ is represented by a vertex in $\mathcal{G}_G$; and (2) each edge in $\mathcal{E}$ connects two snapshots constructed from the same edge or two consecutive edges incident to the same top vertex. Intuitively, edges in $\mathcal{G}_G$ capture movements of top vertices from one bottom vertex to another.

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. We add a directed edge from snapshot s_i= $[t_s^i, t_e^i]_{v_i}$ to s_j=$[t_s^j, t_e^j]_{v_j}$, if one of the following conditions holds: (1) v_i=v_j, $t_e^i \leq t_s^j$ and there exists an edge e=(u, v_i, t_s, t_e) in G such that its time interval contains the time intervals of both s_i and s_j (*i.e.,* $t_s \leq t_s^i$ and $t_e^j \leq t_e$); this edge represents that top vertex u keeps linking to bottom vertex v_i during $[t_s^i, t_e^j]$, although other vertices leave; or (2) $v_i \neq v_j$, $t_e^i \leq t_s^j$ 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 the same top vertex u such that $t_e^1 = t_e^i$ and $t_s^2 = t_s^j$; that is, top vertex u moves from bottom vertex v_i to v_j.

Optimization. Connecting all snapshots that satisfy the above conditions may produce redundant edges in the trajectory 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 avoid such redundancy, we only add an edge from s_1=$[t_s^i, t_e^i]_{v_i}$ to s_2=$[t_s^j, t_e^j]_{v_j}$, if $s_1, s_2 \in \mathcal{S}(u)$ and there exists no snapshot $s_3 = [t_s^l, t_e^l]_{v_l}$ in $\mathcal{S}(u)$ whose time interval lies between the end time of s_1 and the start 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 $\mathcal{G}_G$ such that each intermediate snapshot $[t_i{+}1, t_{i+1}]_v$ ($i \in [2, n-2]$) has exactly one incoming edge and one outgoing edge, while the first snapshot $[t_1, t_2]_v$ and last snapshot $[t_{n-1}+1, t_n]_v$ have only one outgoing and one incoming edges, respectively, we can merge the chain into a single snapshot $[t_1, t_n]_v$. This merging preserves the reachability relationships in $\mathcal{G}_G$.

Example 5.2. Fig. 4 shows the trajectory graph $\mathcal{G}_G$ of G given in Fig. 1, where solid and dashed arrows indicate edges added under conditions (1) and (2), respectively. Consider $\mathcal{S}(u_2)$=$\{[4,6]_{v_1}, [7,7]_{v_1}, [2,3]_{v_2}, [8,8]_{v_3}, [9,9]_{v_3}\}$ (see below). A solid edge exists from s_2=$[4,6]_{v_1}$ to s_4=$[7,7]_{v_1}$ due to edge $(u_2, v_1, 4, 7)$, whose interval $[4,7]$ covers $[4,6]$ and $[7,7]$; a dashed edge connects s_3=$[2,3]_{v_2}$ to s_2=$[4,6]_{v_1}$ due to edges $(u_2, v_2, 2, 3)$ and $(u_2, v_1, 4, 7)$ in G.

Redundant edges of types (a) and (b) have been removed from $\mathcal{G}_G$ in Fig. 4. For instance, although an edge from s_3 to s_6 could be added due to edges $(u_2, v_2, 2, 3)$ and $(u_2, v_1, 8, 9)$, we omit this edge since s_3 can reach s_6 via path $s_3s_2s_4s_6$. Furthermore, s_2=$[4,6]_{v_1}$ results from merging $[4,4]_{v_1}$, $[5,5]_{v_1}$ and $[6,6]_{v_1}$, s_3=$[2,3]_{v_2}$ results from merging $[2,2]_{v_2}$ and $[3,3]_{v_2}$; similar for s_1=$[1,3]_{v_1}$. □

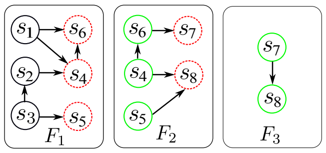

Figure 5: A partition $\{F_1, F_2, F_3\}$ of $\mathcal{G}_G$ given in Figure 4.

Properties. We next present properties of trajectory graphs.

Sizes of trajectory graphs. We have the following result.

THEOREM 5.3. *Given a bigraph $G = (U \cup L, E)$, the trajectory 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 Sketch: (1) We show that $|\mathcal{V}| \leq 4|G|$. For each bottom vertex v, there exist 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 exist 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 show that $|\mathcal{E}| \leq 4d_{\max}|G|$. Observe that (a) there exist $O(4|G|)$ many snapshots; and (b) each snapshot $[t_1, t_2]_v$ has $O(d_v)$ many edges, since these edges link snapshots constructed from (a) the same edges, *e.g.,* a top vertex remains at v, or (b) edges whose end time are t_2, *i.e.,* some top vertex leaves v at t_2. We assume that each top vertex is connected to at most one bottom vertex at timestamp t, as usually found in disease prevention [35, 49, 60]. Then the number of edges in case (b) is bound by $O(d_v)$. Therefore, the number of edges is bounded by $\Sigma_{v \in L} 4d_v^2 \leq 4d_{\max}|G|$. □

The theorem only provides an upper bound for the sizes of trajectory graphs. However, in practice, trajectory graphs are much smaller than original temporal bigraphs (see **Exp-3** in Section 8).

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

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

THEOREM 5.4. *Given a temporal bigraph G=$(U \cup L, E)$, top vertices u and u', and a time interval $[t_s, t_e]$, u can reach u' in G in $[t_s, t_e]$ if and only if src_u can reach $\text{tgt}_{u'}$ in $\mathcal{G}_G$.*

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

($\Rightarrow$) Assume that u can reach u' in G in time interval $[t_s, t_e]$. Then there is a temporal path $\rho = e_1 e_2 \ldots\ldots e_{2k}$, where e_i=(u_i, v_i, t_s^i, t_e^i) for each $i \in [1, 2k-1]$, and $u = u_1$ and $u' = u_{2k}$, such that the end time of $\Theta(e_1, e_2)$ and the start time of $\Theta(e_{2k-1}, e_{2k})$ fall in the time interval $[t_s, t_e]$. We can construct a path in $\mathcal{G}_G$ from src_u to $\text{tgt}_{u'}$ by linking snapshots representing each pair of consecutive edges in ρ.

($\Leftarrow$) Assume that src_u can reach $\text{tgt}_{u'}$ in $\mathcal{G}_G$. Let $P_{\mathcal{G}}$=$(s_1$=$[t_s^1, t_e^1]_{v_1}$, $s_2 = [t_s^2, t_e^2]_{v_2}, \ldots, s_n = [t_s^n, t_e^n]_{v_n})$ be a path witnessing the reachability. Starting from the first edge in $P_{\mathcal{G}}$, we construct a path p_i

in G to encode the edge from s_i to s_{i+1} ($i \in [1, n-1]$), and the constructed paths can be catenated to form a temporal path in G. □

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, since (a) $\mathcal{G}_G$=$(\mathcal{V}, \mathcal{E})$ has at most $4|G|$ vertices and $4d_{\max}|G|$ edges; and (b) it takes $O(\log |G|)$ time to sort the edges.

6 INDEX CONSTRUCTION

We present a contraction-based index schema for SRP. We first partition the trajectory graph into fragments (Section 6.1). Then we construct both intra-fragment and inter-fragment indices (Section 6.2).

6.1 Graph Partitioning

In this section, we partition the trajectory graph $\mathcal{G}_G$ into k fragments F_i=$(\mathcal{V}_i, \mathcal{E}_i, \mathsf{IN}_i, \mathsf{OUT}_i)$ ($i \in [1, k]$). Multiple algorithms have been developed for edge-cut partitions [6, 8, 48]. Leveraging the temporal information in $\mathcal{G}_G$ and the order of timestamps, we adopt a greedy strategy to partition vertices in $\mathcal{G}_G$ [18]. More specifically, we compute a topological order of vertices in $\mathcal{G}_G$. Following this order, each vertex v is assigned to the currenet fragment F_i if $|\mathcal{V}_i| \leq (1+\varepsilon)\frac{|\mathcal{V}|}{k}$; otherwise, a new fragment F_{i+1} is initialized with v. The procedure proceeds until all vertices are assigned.

Example 6.1. Figure 5 shows a partition of $\mathcal{G}_G$ in Figure 4, which consists of fragments F_1, F_2 and F_3, having vertex 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 in the figure. □

Remark. (1) Due to the greedy partitioning strategy, all fragments have similar numbers of vertices, resulting in similar index sizes for each fragment. Moreover, (2) once a path leaves a fragment, it cannot re-enter the fragment, as the trajectory graph $\mathcal{G}$ is a DAG and the partition follows the topological order of its vertices.

The number k of fragments. We partition the trajectory graph into multiple fragments such that intra-fragment indices and inter-fragment indices have the similar size. This ensures that each query has comparable query time, no matter whether the source and target vertices are in the same fragment or not. Specifically, we partition the trajectory graph $\mathcal{G}_G$ such that each fragment F_i=$(V_i, E_i, \mathsf{IN}_i, \mathsf{OUT}_i)$ has similar size as the graph $\mathcal{G}^c_G$=$(\mathcal{V}^c, \mathcal{E}^c)$ constructed by border vertices, which is used to answer queries with source and target vertices in different fragments (see Section 6.2).

We adopt a binary search strategy to determine k. We start with k=256, since $\mathcal{G}^c_G$ is usually larger than F_i when k is large. Given a partition, it is costly to directly construct $\mathcal{G}^c_G$ (see below). To alleviate this, we estimate the size of $\mathcal{G}^c_G$ as the production of the number of border vertices in all fragments and a factor α=8. Then we partition $\mathcal{G}_G$ as follows: (1) partition $\mathcal{G}_G$ into k=256 fragments; (2) estimate the size of the graph $\mathcal{G}^c_G$ as described above, and check whether $\mathcal{G}^c_G$ and F_i have comparable size, *i.e.,* $||\mathcal{V}^c|-|V_i|| \leq \tau$ for a balance factor τ; if $|\mathcal{V}^c|>|V_i|+\tau$, we decrease k; if $|\mathcal{V}^c|<|V_i|-\tau$, increase k; and (3) re-partition graph $\mathcal{G}_G$ and repeat step (2). The process terminates once $||\mathcal{V}^c|-|V_i|| \leq \tau$. Clearly, the process terminates after at most eight iterations from the initial k=256. In practice, on average the binary search procedure accounts for only about 16% of the indexing time (see **Exp-3** in Section 8).

Queries on partitioned trajectory graphs. Given a partitioned trajectory graph, there are two kinds of span-rechability queries:

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

Intra-fragment indices. To answer intra-fragment reachability queries, we construct indices for each fragment using existing strategies, *e.g.,* TopChain [56], PathTree [31], grail [58] and ferrari [40].

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

6.2 Inter-fragment indexing

We build inter-fragment indices in the following two steps: (1) first construct a contraction graph from trajectory graph by merging vertices connecting to the same set of border vertices [20]; and (2) construct the indices on the contraction graph.

Equivalent Sets. We first define two equivalence relations on vertices in the trajectory graph $\mathcal{G}_G$, which categorize vertices based on whether they connect to or are connected from border vertices.

Given a fragment F_i=$(\mathcal{V}_i, \mathcal{E}_i, \mathsf{IN}_i, \mathsf{OUT}_i)$, we define two equivalence 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* $[v]_{\mathsf{EOut}_i}$ consists of vertices in $\mathcal{V}_i$ that reach the same set of border vertices in OUT_i as v; and (2) its *in-equivalence class* $[v]_{\mathsf{EIn}_i}$ comprises is the set of all vertices in $\mathcal{V}_i$ reachable from the same set of vertices in IN_i that can reach v. Here, $[v]_{\mathsf{EIn}_i}$ is defined on vertices in IN_i, *i.e.,* vertices in the same fragment as v, which is used to connect fragments for reachability (see Section 7).

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 v (resp. can be reached from v). Let BIn_i (resp. BOut_i) be the collection of all $\mathsf{BIn}_i[v]$ (resp. $\mathsf{BOut}_i[v]$).

Example 6.2. Consider partition $\{F_1, F_2, F_3\}$ in Fig. 5. (1) In F_1, OUT_1=$\{s_4, s_5, s_6\}$, both s_1 and s_2 reach s_4 and s_6, but s_3 can reach all vertices in OUT_1, then $[s_1]_{\mathsf{EOut}_1}$=$[s_2]_{\mathsf{EOut}_1}$=$\{s_1, s_2\}$ and $[s_3]_{\mathsf{EOut}_1}$=$\{s_3\}$. Also, $[s_1]_{\mathsf{EIn}_1}$=$[s_2]_{\mathsf{EIn}_1}$=$[s_3]_{\mathsf{EIn}_1}$=$\{s_1, s_2, s_3\}$ as IN_1=$\emptyset$. (2) In F_2, IN_2=$\{s_4, s_5, s_6\}$, s_4 and s_5 are only reached by themselves, while s_6 is reached by s_4 and itself, then $[s_i]_{\mathsf{EIn}_2}$=$\{s_i\}$ for $i \in \{4, 5, 6\}$. Likewise, we get $[s_j]_{\mathsf{EOut}_2}$=$\{s_j\}$ for $j \in \{4, 5, 6\}$. (3) In F_3, $[s_7]_{\mathsf{EIn}_3}$=$\{s_7\}$, $[s_8]_{\mathsf{EIn}_3}$=$\{s_8\}$ and $[s_7]_{\mathsf{EOut}_3}$=$[s_8]_{\mathsf{EOut}_3}$=$\{s_7, s_8\}$ since OUT_3=$\emptyset$. □

Construction. We can construct the equivalence relations EOut_i and EIn_i based on their definition as follows: (1) for each vertex v in $\mathcal{V}_i$, determine the subset of border vertices in OUT_i reachable from v and the subset of vertices in IN_i that can reach v; (2) cluster vertices in $\mathcal{V}_i$ based on these subsets, *i.e.,* two vertices are in the same cluster if they can reach or be reached from the same subsets 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 address this, we develop algorithm OutEquivalentSet (see Figure 6), to compute the out-equivalence relation EOut_i by iteratively removing vertices without outgoing edges. This is an extension of the topological sorting algorithm [15].

Given a fragment F_i, it first identifies the set B_i of border vertices in OUT_i that have no outgoing edges (line 1). Since F_i is a DAG, such vertices always exist. It then initializes $\mathsf{BOut}_i[v]$=$\{v\}$ for each border vertex $v \in B_i$ (line 2). After that, it iteratively

```
Input:  A 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 the set $B_i$ of border vertices in $\mathsf{OUT}_i$ without outgoing edges;
2.  set $\mathsf{BOut}_i[v] := \{v\}$ for all $v \in B_i$;
3.  while $B_i \neq \emptyset$ do
4.      $v := B_i$.pop();
5.      for each $u$ having an edge $e$ leading to $v$ do
6.          $\mathsf{BOut}_i[u] := \mathsf{BOut}_i[u] \cup \mathsf{BOut}_i[v]$;
7.          remove edge $e$ from $F_i$;
8.          if $u$ has no outgoing edge then
9.              push $u$ to $B_i$;
10. $\mathsf{EOut}_i := \mathsf{GroupEquiv}(\mathsf{BOut}_i)$;
11. return $\mathsf{EOut}_i$;
```

Figure 6: Algorithm OutEquivalentSet

computes EOut_i as follows (lines 3-9). More specifically, for each border vertex $v \in B_i$ and each incoming neighbor u of v, the algorithm adds set $\mathsf{BOut}_i[v]$ to the set $\mathsf{BOut}_i[u]$ (line 5-6) (*i.e.,* u can reach vertices in both $\mathsf{BOut}_i[v]$ and $\mathsf{BOut}_i[u]$), removes the edge e from u to v from the fragment F_i (line 7), and checks whether all outgoing 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. Finally, the algorithm 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 algorithm OutEquivalentSet. It runs in $O(|\mathcal{E}_i||\mathsf{OUT}_i|)$ time, since (1) each edge is accessed only once (lines 5 and 7), which takes $O(|\mathcal{E}_i|)$ time; (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), giving $O(|\mathcal{V}_i||\mathsf{OUT}_i|)$ time. In practices, $|\mathsf{OUT}_i|$ is much smaller than $|\mathcal{E}_i|$, making algorithm 6 much faster than the direct vertex comparisons.

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

For a trajectory graph $\mathcal{G}_G$=$(\mathcal{V}, \mathcal{E})$ and its partition $\{F_1, F_2, \ldots, F_k\}$, we define its *contraction graph* as $\mathcal{G}_G^c$=$(\mathcal{V}^c, \mathcal{E}^c)$, where 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. The edge set $\mathcal{E}^c$ consists of the following three types of edges: for each vertex $v \in \mathcal{V}_i$,

(a) add edges from the out-equivalence class $[v]_{\mathsf{EOut}_i}$ to $\mathsf{BOut}_i[v]$ (*i.e.,* the set of border vertices reachable from v), and from $\mathsf{BIn}_i[v]$ (*i.e.,* border vertices that can reach v in F_i) to in-equivalence class $[v]_{\mathsf{EIn}_i}$; these edges (*e.g.,* black arrows in Figure 7) connect each equivalence class to border vertices that it can reach or be reached from, preserving reachability within a fragment;

(b) add an edge from $\mathsf{BOut}_i[v]$ to $\mathsf{BIn}_j[v']$ for $i \neq j$, if $\mathsf{BOut}_i[v] \cap \mathsf{BIn}_j[v'] \neq \emptyset$, *i.e.,* there exists a vertex u in F_j that is also a border vertex in F_i, and v can reach u in F_i, and in turn reach v' in F_j via u; hence different fragments are connected through shared border vertices (*e.g.,* blue arrows in Figure 7);

(c) add an edge from $\mathsf{BOut}_i[v]$ to $\mathsf{BOut}_j[v']$ for $i \neq j$, if v' is in $\mathsf{BOut}_i[v]$; in this case, v can reach v' (a vertex in fragment F_j) and all border vertices reachable from v' in fragment F_j (*i.e.,* vertices

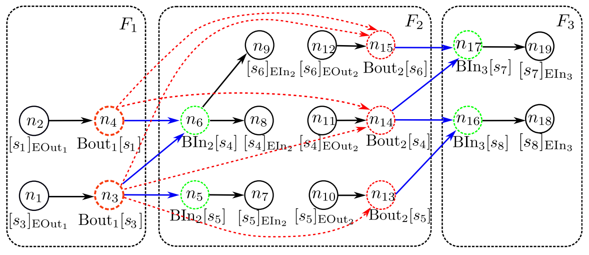

Figure 7: Contraction Graph $\mathcal{G}_G^c$

in $\mathsf{BOut}_j[v']$); such edges propagate reachability across fragments by chains of border vertices (*e.g.,* red dashed arrows in Figure 7).

Example 6.3. We construct contraction graph $\mathcal{G}_G^c$ (see Figure 7) based on the partition in Figure 5, where edges of type (a)-(c) defined above are shown as black, blue, and red dashed edges, respectively. (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 $\mathsf{BOut}_1[s_3] \cap \mathsf{BIn}_2[s_5] = \{s_5\}$; (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. □

Optimization with trie. The contraction graph may contain redundant edges, *e.g.,* when sets BIn_i and BOut_j share border vertices. Consider a vertex u appearing in both fragments F_1 and F_2. Suppose u occurs in m sets $\mathsf{BOut}_1[u_1], \ldots, \mathsf{BOut}_1[u_m]$ in F_1 (*i.e.,* u is a border vertex reachable from u_i) and a set $\mathsf{BIn}_2[v]$ in F_2 (*i.e.,* u can reach v in F_2). This yields m edges in the contraction graph from each $\mathsf{BOut}_1[u_i]$ to $\mathsf{BIn}_2[v]$, *i.e.,* one for each appearance of u in BOut_i.

To address this, we employ the trie data structure [23] to reduce redundancy. Specifically, we assign each vertex in $\mathcal{G}_G^c$ a unique ID and sort them in ascending order by their appearance counts in BOut_i and BIn_i. Then all BOut_i and BIn_i sets are represented as paths in a prefix tree based on this order. When constructing contraction graphs, we connect vertices based on their appearances in the trie, rather than directly using BOut_i and BIn_i, thereby avoiding duplicated edges. In this way, the m edges above can be reduced to a single edge, when vertex u appears only once in the trie.

Analysis. 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 $\mathsf{BOut}_i[s_1]$ can reach $\mathsf{BIn}_j[s_2]$ in the contraction graph $\mathcal{G}_G^c$, where s_1 and s_2 are in F_i and F_j, respectively. Therefore, we can construct $\mathcal{G}_G^c$ in $O(\Sigma_{i\in[1,k]}|\mathcal{V}_i||\mathsf{Out}_i|)$.

Remark. To accelerate queries across different fragments, we adopt TopChain [56] to construct the chain cover on $\mathcal{G}_G^c$ [56]. More specifically, (a) we first decompose $\mathcal{G}_G^c$ into chains such that each vertex belongs to 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} for each $(i \in [1, n-1])$; and (b) we next compute the reachability labels for each vertex in $\mathcal{G}_G^c$, which record the chains that the vertex can reach and can be reached from. Unlike TopChain [56], which applies chain covers directly on temporal graphs, we construct the chain covers on both fragments and contraction graphs, which are smaller than the original graph, leading to better performance (see Section 8).

```
Input:  Trajectory graph $\mathcal{G}_G = (\mathcal{V}, \mathcal{E})$ and vertices $v_1$, $v_2$.
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 Query($v_1$, $v_2$, $F_i$);
3.  else
4.      ($F_1$, $F_2$) := Locate($v_1$, $v_2$, $\mathcal{G}_G$);
5.      $v'_1$ := Border$_{\text{out}}$($v_1$, $F_1$, $\mathcal{G}_G$);
6.      $v'_2$ := Border$_{\text{in}}$($v_2$, $F_2$, $\mathcal{G}_G$);
7.      bReach := CheckReach($v'_1$, $v'_2$);
8.      return bReach;
```

Figure 8: Algorithm SRPQ

7 QUERY ON CONTRACTION GRAPHS

We develop algorithm SRPQ for the span-reachability problem using indices presented in Section 6.

Overview. To answer the reachability on partitioned trajectory graphs, we must tackle the following challenges: (1) How to convert the reachability problem on a trajectory graph to the one on its contraction graph, and preserve the reachability. (2) How to conduct reachability queries on the contraction graphs, especially when the given two vertices are in different fragments.

We solve these problems as follows.

(1) Given two top vertices u and v in a temporal bigraph G and time interval $[t_s, t_e]$, we first identify snapshots src_u and tgt_v in trajectory graph $\mathcal{G}_G$ 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 tgt_v (see Section 5). To accelerate the computation, we first group snapshots based on bottom vertices and then sort them based on their time intervals. Then src_u and tgt_v can be identified using binary search.

(2) Then we check whether src_u and tgt_v are in the same fragment of $\mathcal{G}_G$. 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 [56], PathTree [31], grail [58] and ferrari [40]. Here, we adopt TopChain [56] as a case study. When they are in different fragments, we answer the queries using inter-fragment indices, which is also computed using TopChain [56].

Algorithm. We present algorithm SRPQ in Figure 8. Given vertices v_1 and v_2, and trajectory graph $\mathcal{G}_G$, it first checks whether v_1 and v_2 are in the same fragment; if so, it exploits the intra-partition indices to check whether v_1 can reach v_2 (line 1); otherwise, it checks whether v_1 can reach v_2 using the chain covers [56] (lines 4-8). (1) It first identifies fragments F_1 and F_2 containing v_1 and v_2, respectively (line 4). Recall that $\mathcal{G}_G$ is partitioned via edge-cut, and each vertex is in only one fragment. (2) Then it fetches the set $\text{BOut}_1[v_1]$ of vertices that v_1 can reach in F_1, by accessing equivalence class $[v_1]_{\text{EOut}_1}$ (line 5). Similarly, it collects $\text{BIn}_2[v_2]$ of vertices that can reach v_2 in F_2 (line 6). (3) Finally, it checks whether $\text{BOut}_1[v_1]$ can reach $\text{BIn}_2[v_2]$ via CheckReach using chain covers defined on $\mathcal{G}_G$, *i.e.,* the TopChain algorithm (line 7), which recursively checks reachability following the chain covers.

Example 7.1. Consider temporal bigraph G and its trajectory graph, partition and contraction graph in Figures 1, 4, 5 and 7, respectively. (1) For query $Q(u_4, u_1, G, [1, 7])$, we get $\text{src}_{u_4}=[2, 3]_{v_2}=s_2$ and $\text{tgt}_{u_1}=[4, 6]_{v_2}=s_3$, which are all in fragment F_1. Then the query is answered via the intra-fragment indices. (2) For query $Q(u_2, u_5, G, [1, 9])$, we identify that $\text{src}_{u_2}=[2, 3]_{v_2}=s_2$ and $\text{tgt}_{u_5}=[9, 9]_{v_3}=s_7$, which are in F_1 and F_3, respectively. Then the inter-fragment index are used. Since (a) s_2 is in $[s_1]_{\text{EOut}_1}$ and s_7 is in $[s_7]_{\text{EIn}_3}$, and (b) $[s_1]_{\text{EOut}_1}$ can reach $[s_7]_{\text{EIn}_3}$ in $\mathcal{G}^c_G$ via vertices $\text{BOut}_1[s_1]$, $\text{BOut}_2[s_6]$ and $\text{BIn}_3[s_7]$, u_2 reach u_5 in G in [1, 9]. □

Analysis. We next analyze the complexity of SRPQ. Denote by $O(f(|G|))$ the complexity of TopChain [56] to answer queries. Here, $f(|G|)$ is a polynomial on $|G|$, 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 $O(\max(f(\max_i |F_i|), f(|\mathcal{G}^c_G|)))$ to answer a query, and is faster than TopChain, since F_i and $\mathcal{G}^c_G$ are much smaller than G, when the number of fragments is large (see Section 8).

Remark. The contraction graphs can also be used to answer both single-source reachability and earliest-arrival reachability.

(1) Single-Source Reachability (SSR) identifies all vertices reachable from a source vertex u within a given time interval on a bigraph. SSR can be answered by performing a Breadth-First Search starting from vertex u using the in-equivalence and out-equivalence classes in each fragment and the edges between fragments.

(2) The Earliest-Arrival Reachability (EAR) problem is to identify the earliest time at which a source vertex u can reach a target vertex v within a given time interval $[t_s, t_e]$. Since the interval contains $t_e - t_s$ distinct timestamps, EAR can be solved using a binary search strategy, requiring at most $\log_2(t_e - t_s)$ calls of SPRQ.

8 EXPERIMENTS

Using real-life, we experimentally evaluated CBSR for its (1) correctness, (2) efficiency (3) effectiveness, (4) parameter sensitive and (5) scalability. The source code is publicly available in [2].

Experimental setting. We start with the setting.

Datasets. We used eleven real-life temporal bigraphs, summarized in Table 2, where Name denotes dataset abbreviation, $|E|$ is the number of edges, and $|U|$ and $|L|$ are the numbers of top and bottom vertices, respectively; Δ is the time span (in days) of the graph, *i.e.,* the difference between the maximum and minimum timestamps in the graph.

Table 2: Dataset summary

Name	Dataset	$\lvert E\rvert$	$\lvert U\rvert$	$\lvert L\rvert$	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

All graphs are from Konect [1]. Since each edge carries only a single timestamp, we generated time intervals by sampling an interval length Δt from a power-law distribution $p(\Delta t) = C\tau^{-\alpha}$ [7], with

Table 3: Average positive and negative query time (μs)

Dataset	Positive queries				Negative queries			
	TC	PT	TBP	SRPQ	TC	PT	TBP	SRPQ
WP(5)	22.94	18.56	**0.21**	17.74	3.59	3.38	**0.52**	3.53
SO(2)	67.14	18.37	**2.03**	30.03	1.07	3.27	3.37	**0.82**
LK(7)	40.34	32.38	**7.97**	12.82	1.27	2.22	19.26	**1.16**
CU(3)	63.40	112.93	OT	**35.18**	**0.20**	7.33	OT	0.32
BS(11)	194.13	82.01	**5.81**	75.97	18.59	22.62	**12.95**	19.81
TW(3)	1,396.67	313.72	OT	**205.17**	3.33	13.21	OT	**0.58**
AM(3)	52.28	8.57	**1.82**	30.75	0.99	1.57	1.86	**0.93**
EP(6)	272.06	OM	OT	**186.46**	2.98	OM	OT	**2.50**
LF(5)	105.36	64.63	**6.14**	64.08	11.00	10.97	11.99	**10.24**
EJ(7)	90.46	204.47	**13.70**	46.16	5.19	13.78	41.70	**4.80**
ED(13)	678.53	OM	OT	**242.34**	14.67	OM	OT	**10.96**

⋆ Best results are shown in **bold** and the second best in underline.

Table 4: Original *vs.* transformed graph sizes

Dataset	Temporal bigraphs G			Trajectory graphs $\mathcal{G}_G$	
	$\lvert E\rvert$	$\lvert U\rvert$	$\lvert L\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{V}\rvert$
WP	378,979	4,312	49,500	12,746	10,399
SO	1,301,943	545,195	96,678	556,044	520,871
LK	1,565,684	42,045	337,509	506,037	398,311
CU	2,411,820	153,277	731,769	2,262,206	1,381,283
BS	2,555,081	5,794	204,673	111,783	98,884
TW	4,664,606	175,214	530,418	3,445,596	2,684,630
AM	5,838,042	2,146,057	1,230,915	476,363	529,769
EP	13,668,321	120,492	755,760	5,815,905	4,163,505
LF	19,150,869	992	1,084,620	112,935	85,873
EJ	41,998,341	444,563	2,963,672	1,830,929	1,474,038
ED	129,885,940	1,025,084	5,812,980	24,225,743	18,946,499

C ranging from 10 minutes to 8 hours and $\alpha = -2.5$ following [12].

Algorithms. We evaluated the following algorithms.

Indexing algorithms. We implemented (1) CBSR in C++, which comprises graph transformation (Section 5), index construction (Section 6) and query answering (*i.e.,* SRPQ in Section 7). In our comparison, we focus on the transformation and index construction phases, and the intra-fragment indices are constructed by TopChain [56].

We compared with (2) TBP [12], using an implementation from [37]. We also compared CBSR with algorithms for DAGs: (3) TopChain (TC) [56], a chain-based index for temporal DAGs; and (4) PathTree (PT) [31], a path-tree based index for DAGs.

Remark. (1) Since PathTree is designed for DAGs, we apply it on the trajectory graphs (*i.e.,* DAGs) constructed from the temporal bigraphs. (2) For TopChain on temporal DAGs, we assigned timestamps to the edges in the trajectory graphs based on the timestamps in the snapshots of the trajectory graphs. More specifically, for each edge from $[t_s^i, t_e^i]_v$ to $[t_s^j, t_e^j]_{v'}$ in the trajectory graph we assign a time interval $[t_e^i, t_s^j]$. Intuitively, for instance, a person stays at bottom vertex (*e.g.,* location) v during $[t_s^i, t_e^i]$ and then moves to another bottom vertex (*e.g.,* location) v' at time t_s^j; then the time interval on the edge represents the time used to move from v to v'. Consequently, this assignment preserves the original reachability and thus does not affect the correctness of reachability queries.

Query algorithms. We compared query algorithm SRPQ (Section 7) with two kinds of algorithms: (a) query algorithms on temporal bigraphs, and (b) query algorithms on DAGs.

(a) For queries on temporal bigraphs, we implemented in C++: (5) SRPQ, our query algorithm in Section 7. We compared with (6) query algorithm TBP from [12]. Recall that the problem defined in TBP [12] differs from ours (see Example 3.4). To ensure a fair comparison, we split each edge into multiple edges with smaller intervals. We can verify that SRPQ and TBP produce the same answers on the same query. Indeed, when using edges with smaller intervals, temporal paths defined by our definition are also temporal paths under the union semantics of TBP [12]. However, such processing results in much larger graphs, and degrades the performance of TBP (see **Exp-2** for the comparison).

(b) For queries on DAGs, we compared SRPQ with TopChain [56] and PathTree [31]. Since their original implementations do not support bigraphs, we evaluate them on DAGs obtained via the transformation strategy in Section 5.

Environment. We run experiments on a Linux server with an Intel Xeon Platinum 8358 CPU (2.6GHz, 32 cores, 64 threads), 1TB memory. Experiments were repeated 5 times and averages are reported.

Experimental results. We next report our findings.

Exp-1: Correctness. To verify the correctness of CBSR, we compared it with TBP in a real-life setting, *e.g.,* contact tracing [17], on all real-life graphs. The ground truth is obtained by performing a BFS search on real-life datasets with the following constraints on the visited paths: (1) adjacent edges of one bottom vertex have a common timestamp; and (2) the timestamps increase along the path. Denoted by Init-TBP the variant of TBP that takes the original temporal bigraph as input. We do not report the result of TopChain and PathTree, since they cannot directly process temporal bigraphs. We evaluated both CBSR and Init-TBP using 100k positive queries on each real-life graph. We found the following: (1) over all real-life graphs, CBSR correctly answers all these queries due to the fine-grained definition of the span-reachability (see Section 3). (2) Init-TBP failed to build indices on datasets CU, TW, EP, EJ and ED within 24 hours, and its accuracy ranges from only 50.04% (on AM) to 94.55% (on LF) on the remaining datasets, with an anverage 75.48%, due to its coarse definition of reachability.

We did not use BFS as baseline, as it is much slower than CBSR. Given 100k positive queries on WP, the BFS algorithm on average takes 90.68ms to finish a query, while CBSR takes only 17.74μs.

Exp-2: Query processing. We then evaluated the performance of the query algorithms. Queries are generated in the form of (u, u', t_s, t_e), where u and u' are top vertices. The start time t_s is randomly selected from the time intervals associated with edges of the temporal bigraph, and the interval length $t_e - t_s$ follows the power-law distribution as described above. Specifically, we generated 100k positive queries and 100k negative queries for each graph, and report the average query time in Table 3. Here, WP(6) denotes that dataset WP was partitioned into six fragments, and similarly for other datasets; OT indicates that the algorithm did not finish within 24 hours; and OM denotes out of memory error.

(1) On positive queries, SRPQ consistently outperforms TopChain (TC) across all datasets by a factor ranging from 1.29× (on WP) to

Table 5: Indexing time and memory usage

Dataset	Indexing Time (ms)				Memory Usage (GB)			
	TC	PT	TBP	CBSR	TC	PT	TBP	CBSR
WP(5)	**143**	184	78,169	160	0.80	0.81	**0.08**	0.80
SO(2)	**3,528**	40,427	3,673,724	4,899	1.88	9.07	1.32	**0.81**
LK(7)	**1,612**	118,218	530,398	2,789	0.81	19.83	**0.40**	0.81
CU(3)	**6,740**	1,885,985	OT	16,765	**4.10**	212.32	OT	4.57
BS(11)	**979**	2,456	3,094,355	1,081	8.99	0.79	**0.50**	1.44
TW(3)	**19,683**	4,100,630	OT	40,027	**6.67**	597.55	OT	6.77
AM(3)	**3,467**	12,626	7,186,620	5,713	**2.59**	4.90	2.74	2.60
EP(6)	**24,512**	OM	OT	76,409	**9.18**	OM	OT	11.20
LF(5)	10,118	10,964	11,349,943	**6,225**	8.81	4.32	**3.38**	4.32
EJ(7)	**18,687**	2,382,505	35,224,180	26,191	9.68	299.57	**7.12**	10.49
ED(13)	**134,457**	OM	OT	546,569	**50.85**	OM	OT	**50.85**

⋆ Best results are shown in **bold** and the second best in underline.

6.81× (on TW), with an average of 2.49×, since the graph partitioning can effectively reduce the search space. On negative queries, both SRPQ and TopChain are fast (5.05μs *vs* 5.72μs on average), due to the effectiveness of the pruning rules in TopChain. On all queries, SRPQ is on average 1.99× faster than TopChain.

(2) On the EP and ED datasets, PathTree (PT) failed to build indices within the memory limit. On the LK, CU, TW and EJ datasets, SRPQ is on average 2.55× faster than PT on positive queries. This is because these datasets contain a large number of vertices and edges in their trajectory graphs (see **Exp-3** below), enlarging the search space, which however can be mitigated by the contraction-based strategy in SRPQ [31]. While on the SO and AM datasets, SRPQ is slower than PT, since the trajectory graphs of these graphs are sparse, on which PT constructs fewer paths and labels, which can reduce the searching space.

(3) Algorithm TBP is faster than SRPQ on positive queries (5.38μs *vs* 39.65μs on average), since it constructs the minimal TBP-Index to accelerate query processing. However, constructing such index is time-consuming, thus TBP cannot process large-scale graphs: on CU, TW, EP and ED datasets, TBP cannot complete index construction within 24 hours. On negative queries, SRPQ outperforms TBP on most datasets, by a factor ranging from 1.07× (on LF) to 22.90× (on CU), with an average 6.59×, due to the pruning rules in TopChain. On WP and BS, SRPQ is slower than TBP, since the TBP-Index of TBP is built on top vertices and the number of which are small (see Table 2).

Exp-3: Index construction. We next evaluated the indexing time and memory usage of our framework CBSR.

Size of the trajectory graph. We first evaluated the sizes of trajectory graphs transformed from the original temporal bigraphs. As shown in Table 4, we found that $|\mathcal{E}|$ (*i.e.,* number of edges in trajectory graph) is consistently smaller than $|E|$ (*i.e.,* number of edges in temporal bigraph) across all datasets. Specifically, $|\mathcal{E}|$ ranges from 0.59% (on LF) to 93.79% (on CU) of $|E|$, with an average of 29.52%, demonstrating the effectiveness of snapshots and trajectory graphs in reducing graph sizes.

Indexing time. We next evaluated the indexing algorithms. As shown in Table 5, (1) CBSR outperforms PathTree on all datasets by an average factor of 40.43×, since PathTree computes additional reachability labels, which is time-consuming (2) CBSR outperforms TBP by a factor ranging from 190.18× (on LK) to 2, 862.49× (on BS), with an average of 1245.32×, showing the effectiveness of the contraction-based method. Moreover, TBP cannot handle fine-grained reachability queries as CBSR. To ensure that TBP returns correct answers, we split edges, which enlarges graphs and degrades its performance. (3) CBSR is slower than TopChain (TC) in index construction, since CBSR incurs extra cost to build inter-fragment indices, which accelerate the reachability analysis (see **Exp-2**).

Table 6: Cost breakdown of indexing time (s)

Dataset	Trans	BiSearch	Part	EqSet	Intra	Inter
WP(5)	**0.124**	0.013	0.005	0.003	0.012	<0.001
SO(2)	0.773	**1.592**	0.596	1.000	0.792	0.143
LK(7)	0.666	**0.700**	0.325	0.536	0.494	0.066
CU(3)	2.066	3.735	1.697	**6.213**	2.520	0.532
BS(11)	**0.787**	0.102	0.048	0.038	0.098	0.005
TW(3)	3.944	8.000	3.815	**17.923**	5.561	0.781
AM(3)	**2.037**	1.566	0.643	0.756	0.616	0.093
EP(6)	8.414	12.958	6.396	**37.901**	8.205	2.532
LF(5)	**5.851**	0.114	0.062	0.068	0.102	0.026
EJ(7)	**13.856**	3.896	1.713	4.194	2.217	0.312
ED(13)	61.331	56.726	31.281	**358.859**	35.510	2.859

⋆ The longest time among the six phases is shown in **bold**.

Memory usage. As shown in Table 5, (1) compared with PathTree (PT), CBSR uses much less memory on most datasets, as PathTree computes additional reachability labels, which dramatically enlarge the graphs. (2) CBSR incurs slightly higher memory overhead than TBP, since TBP constructs the minimal TBP-Index. However, due to this reason TBP cannot process large-scale graphs like CU, TW, EP or ED. That is, CBSR strikes a balance between query efficiency and space consumption. (3) CBSR consumes more memory than TopChain, as CBSR takes extra space for inter-fragment indices.

Cost breakdown. We conducted a cost breakdown analysis on the indexing time in CBSR, which is divided into six phases: graph transformation (Trans), binary search for fragment n number (BiSearch), graph partitioning (Part), equivalent set construction (EqSet), inter-index construction (Inter), and intra-index construction (Intra). As shown in Table 6, (1) on the CU, TW, EP and ED datasets, computing equivalent classes (*i.e.,* EqSet) dominates the total cost, consuming 37%, 45%, 50% and 66% of the indexing time in CBSR, respectively. This is because these datasets have large trajectory graphs. (2) On WP, BS, AM, LF and EJ datasets, graph transformation (Trans) accounts for most of the indexing time, contributing 79%, 73%, 36%, 94% and 53%, respectively. The trajectory graphs of these datasets are small, while their temporal bigraphs are much larger than trajectory graphs. Then CBSR takes less time to compute equivalence classes, and Trans dominates the cost. (3) On SO and LK, BiSearch dominates the cost (*i.e.,* 32% and 25%, respectively), since the trajectory graphs of SO and LK are moderate, all steps take similar time, and BiSearch is repeated for multiple times. (4) Across all datasets, BiSearch on average takes only 16% of indexing time.

Exp-4: Parameter sensitivity. We evaluated the impacts of partition numbers and connection latency (see below) on CBSR.

Partition numbers. We varied the partition number N_f (*i.e.,* the number of fragments) from 2 to 16 to evaluate the effectiveness of our binary search strategy that determines the partition number k. For datasets CU, TW, EP, EJ and ED, each of which contains more than 1M vertices in its trajectory graphs, we generated 1k intra-fragment and 1k inter-fragment queries. To eliminate the impact of

Table 7: The query time of SRPQ using different partition numbers (μs)

Dataset	Number of Fragments														
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CU(3)	5395	**4632**	4666	4711	5240	5369	6102	5914	6324	6166	6784	6646	7011	7017	7592
TW(3)	27711	16155	13553	**12316**	12745	14077	14446	15645	15963	16702	17368	18656	17978	19229	19010
EP(6)	28308	20320	16715	14469	13169	14098	13935	13098	**12564**	13149	14391	13969	13519	13806	14057
EJ(7)	6484	4556	3538	3331	2931	2944	2876	2845	**2721**	2759	2953	2878	3018	2941	2979
ED(13)	157174	100585	75641	59208	52592	48768	44325	41128	38962	37978	35583	35832	35583	34750	**33902**

★ Best results are shown in **bold**; the number in parentheses (*e.g.,* **CU(3)**) indicates the partition number estimated by our binary search.

Table 8: The impact of δ on CBSR

Dataset	Metric	δ(day)					
		1	2	4	8	16	32
EJ	IT(ms)	21,269	21,836	22,349	22,984	23,339	24,206
	PQT(μs)	27.33	31.04	31.57	31.19	31.52	33.73
	NQT(μs)	5.03	5.10	5.01	5.03	4.93	5.01
CU	IT(ms)	8,701	9,234	10,107	10,894	11,655	12,726
	PQT(μs)	2.26	3.63	4.97	6.46	9.78	13.81
	NQT(μs)	0.16	0.17	0.17	0.18	0.20	0.18
TW	IT(ms)	26,522	28,784	32,147	37,635	39,270	44,248
	PQT(μs)	50.37	59.45	94.11	117.24	167.94	168.34
	NQT(μs)	2.94	3.02	3.01	3.06	3.18	3.11

Table 9: Scalability

Scale factor	0.20	0.40	0.60	0.80	1.00
IT (s)	45.18	199.75	616.64	1,695.30	4,788.91
MU (GB)	25.60	55.71	92.57	138.38	183.10
QT (μs)	39.00	86.96	191.75	112.81	1068.80

the pruning rules in TopChain, we used its original DFS-based query procedure, to evaluate the impact of the partition strategy only.

The average query times are reported in Table 7. (1) When N_f increases, the average query time first decreases and then increases, as expected, since when N_f gets larger, fragment size decreases but the contraction graph gets larger. For dataset ED, the optimal performance is achieved when $N_f = 16$, since ED is large and the time for intra-fragment query dominates the overall time when N_f is small. (2) Partition numbers estimated by binary search (next to datasets) yield the performance of SRPQ comparable to its best performance (in bold), confirming the effectiveness of our approach. (3) The best performance (in bold) occurs when the sizes of each fragment and the contraction graph are similar (not shown).

Connection latency. We next studied the impact of a parameter, namely connection latency, on the performance of CBSR. 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 (Section 5), if the ending time of s_1 is not larger than the starting time of s_2 (*i.e.,* $t_e^1 \leq t_s^2$) and there exists a top vertex u linked to v_1 and v_2 at time t_e^1 and t_s^2, respectively. However, if t_s^2 is sufficiently larger than t_e^1 (*e.g.,* one year apart), such edge is unlikely to be useful for reachability queries, especially in applications like disease tracing, where the latency of infection is much shorter.

Therefore, we define the connection latency δ as the maximum time span within which two snapshots can be linked. Formally, s_1 and s_2 cannot have an edge if $|t_e^1 - t_s^2| \geq \delta$ or $|t_e^2 - t_s^1| \geq \delta$.

Table 8 reports the results on indexing time (IT), query time on positive queries (PQT) and negative queries (NQT). (1) Increasing δ increases the indexing time, as expected, since larger δ leads to more edges in trajectory graphs. (2) Increasing δ slows down CBSR on positive queries, as expected, since increasing δ enlarges the number of edges in trajectory graphs, which leads to a larger search space. In contrast, the negative query time is insensitive to δ, because the pruning rules in query algorithm of TopChain, *i.e.,* procedure CheckReach in SRPQ, can identify negative queries in the early stage. (3) The impact of δ on CBSR and SRPQ varies across datasets. (a) On TW, when δ varies from 32 to 1, CBSR is 1.67× faster and SRPQ is 3.36× faster on positive queries, since TW is a user-tag relations dataset, users often use tags for a long time, and when δ is small there exist few edges in trajectory graphs. (b) While on EJ, a wiki-editing dataset, CBSR is only 1.14× faster, and SRPQ is 1.23× faster, as wiki texts are frequently updated in a short time, *i.e.,* the number of edges in trajectory graphs is less sensitive to δ.

Exp-5: Scalability. We use the edit-enwiki dataset (with 8.1M top vertices, 42.5M bottom vertices and 572M edges) [1] to evaluate the scalability of CBSR on the indexing time (IT), memory usage (MU) and query time (QT) (see Table 9), by varying the scale factor from 0.2 to 1. (1) When $|G|$ gets larger, CBSR takes longer for indexing and conducting queries. And CBSR also uses more memory, as expected. (2) CBSR scales well. It takes $4,788.91$s and 183.10GB to construct indices, and 1068.80μs to process a query when the scale factor is 1, *i.e.,* the entire edit-enwiki dataset.

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 6.83M vertices and 129.88M edges, it can construct the indices in 546.57s while TBP fails to construct the indices in 24 hours. And over all datasets, it is on average 1245.32× faster than TBP. (3) CBSR is effective. Using the indices constructed by CBSR, SRPQ is on average 1.99× faster than TopChain over all datasets.

9 CONCLUSION

This paper proposes algorithms to answer the span-reachability problem on temporal bigraphs. Our contributions include: (1) a fine-grained definition for the span-reachability; (2) a graph transformation method that converts the span-reachability problem on temporal bigraphs to the reachability problem on DAGs; (3) a contraction-based indexing schema to process large-scale graphs; and (4) a query algorithm for the span-reachability queries on temporal bigraphs. Experiments show that the methods are promising in practice.

One topic for future work is to develop effective partition algorithms for trajectory graphs, to reduce border vertices, and better strategies to determine the number of fragments.

REFERENCES

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

[2] Full version, data and code, 2025. *https://github.com/SunboTax/CBSR.git.*

[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] C. Alexander and D. Cumming. *Corruption and Fraud in financial markets: Malpractice, Misconduct and Manipulation.* John Wiley & Sons, 2022.

[5] S. Ali. Controlling serious financial crime: A jamaican perspective. *Economic Affairs*, 27(1):14–17, 2007.

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

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

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

[9] J. D. Brunner and N. Chia. Confidence in the dynamic spread of epidemics under biased sampling conditions. *PeerJ*, 8, 2020.

[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] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

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

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

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

[22] 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.

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

[24] E. Frejinger and M. Hewitt. Perspectives on optimizing transport systems with supply-dependent demand. *INFOR*, 2025. Publisher Copyright: © 2025 Canadian Operational Research Society (CORS).

[25] Z. Guan, L. Wu, H. Zhao, M. He, and J. Fan. Enhancing collaborative semantics of language model-driven recommendations via graph-aware learning. *TKDE*, 37(9):5188–5200, 2025.

[26] V. Hajipour, M. Tavana, D. Di Caprio, M. Akhgar, and Y. Jabbari. An optimization model for traceable closed-loop supply chain networks. *Applied Mathematical Modelling*, 71:673–699, 2019.

[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] S. Khodabandehlou and A. H. Golpayegani. Fifraud: Unsupervised financial fraud detection in dynamic graph streams. *ACM Trans. Knowl. Discov. Data*, 18(5):111:1–111:29, 2024.

[35] 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.

[36] 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.

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

[38] B. J. Mirza, B. J. Keller, and N. Ramakrishnan. Studying recommendation algorithms by graph analysis. *J. Intell. Inf. Syst.*, 20(2):131–160, 2003.

[39] S. M. Rahimi, B. H. Far, and X. Wang. Behavior-based location recommendation on location-based social networks. *GeoInformatica*, 24(3):477–504, 2020.

[40] 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.

[41] 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.

[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. *VLDB J.*, 33(5):1399–1426, 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] 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.

[56] 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.

[57] 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.

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

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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

A. Proof of Theorem 5.4

($\Rightarrow$) Assume that vertex u can reach v in G in time interval $[t_s, t_e]$. There is a temporal path $\rho = e_1 e_2 \ldots e_{2k-1} e_{2k}$ with $u = u_1$ and $v = u_{k+1}$ such that (1) e_1 and e_{2k} are adjacent edges of u and v, respectively, and (2) the end time of $\Theta(e_1, e_2)$ and the start time of $\Theta(e_{2k-1}, e_{2k})$ fall in the time interval $[t_s, t_e]$.

Given the temporal path ρ, we construct a path from src_u to tgt_v in the traject graph $\mathcal{G}_G$ as follows: for each four consecutive edges $e_1^i e_2^i e_1^{i+1} e_2^{i+1}$, there exist four snapshots constructed from the starting times and ending times of these edges. That is, (a) snapshots $s_1 = [t_s^i, t_s^i]_{v_i}$ and $s_2 = [t_e^i, t_e^i]_{v_i}$ are constructed from the first two consecutive edges e_1^i and e_2^i with $\Theta(e_1^i, e_2^i) = [t_s^i, t_e^i]$; (b) snapshots $s_3 = [t_s^{i+1}, t_s^{i+1}]_{v_{i+1}}$ and $s_4 = [t_e^{i+1}, t_e^{i+1}]_{v_{i+1}}$ are constructed from the last two edges e_1^{i+1} and e_2^{i+1} with $\Theta(e_1^i, e_2^i) = [t_s^{i+1}, t_e^{i+1}]$; and (c) there is a path from s_2 to s_3 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 two consecutive edges e_1^i and e_2^i, the timestamps of s_1 and s_2 are covered by both edges e_1^i and e_2^i. Then there exists a path from s_1 and s_2 in $\mathcal{G}_G$ due to edges e_1^i and e_2^i. Similarly, there exists a 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} due to the definition of temporal paths, we can collect all its adjacent edges $(u_{i+1}, v_1, t_s^{12}, t_e^{12}), (u_{i+1}, v_2, t_s^{22}, t_e^{22}), \ldots, (u_{i+1}, v_k, t_s^{k2}, t_e^{k2})$ during the time interval $[t_e^i, t_s^{i+1}]$. 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 u_{i+1}. Moreover, there exist an edge from s_2 to snapshot $[t_s^{12}, t_s^{12}]_{v_1}$, and an edge from snapshot $[t_s^{k2}, t_s^{k2}]_{v_k}$ to s_3, since both of these snapshots are constructed from edges of top vertex u_{i+1} during the time interval $[t_e^i, t_s^{i+1}]$. Therefore, there is a path from s_2 to s_3 in $\mathcal{G}_G$.

Given two consecutive two edges e_1^{i+1} and e_2^{i+1} in the temporal path ρ, we construct the same two snapshots s_1 and s_2, no matter we consider the four consecutive edges $e_1^i e_2^i e_1^{i+1} e_2^{i+1}$ or $e_1^{i+1} e_2^{i+1} e_1^{i+2} e_2^{i+2}$. Then we connected all paths of length four constructed above to form a path from src_u to tgt_v in traject graph $\mathcal{G}_G$.

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

($\Leftarrow$) Assume that snapshot src_u can reach snapshot tgt_v in $\mathcal{G}_G$. Let $P_{\mathcal{G}} = (s_1, s_2, \ldots, s_{n-1}, s_n)$ be such a path with $s_1 = \text{src}_u$ and $s_n = \text{tgt}_v$ in $\mathcal{G}_G$. For each edge from s_i to s_{i+1} ($i \in [1, n-1]$) in $\mathcal{G}_G$, we identify multiple consecutive edges in G to form a temporal path witnessing vertex u can reach v in G in time interval $[t_s, t_e]$.

(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 the pair of edges 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 $e_e^2 = (u_4, v_2, t_s^{22}, t_e^{22})$. From the construction of the traject graph, we have that (a) the intersection of e_s^1 and e_e^1 is not empty (*i.e.,* $\Theta(e_1, e_2) \neq \emptyset$); similar for e_s^2 and e_e^2; (b) the starting time of the intersection $\Theta(e_s^1, e_e^1)$ is not larger than the starting time of $\Theta(e_s^2, e_e^2)$; (c) timestamps of consecutive edges e_e^1 and e_s^2 of top vertex u_3 increase; and (d) $t_e^1 \leq t_s^2$, where $s_1 = [t_s^1, t_e^1]_{v_1}$ and $s_2 = [t_s^2, t_e^2]_{v_2}$.

We construct the temporal path in G for u based on $s_1 = [t_s^1, t_e^1]_{v_1}$ and $s_2 = [t_s^2, t_e^2]_{v_2}$ as follows.

(I) We start with the two consecutive edges from u_2 to u_4. We construct the following path: $u_2 v_1 u_3 v_2 u_4$. This is a valid temporal path, since (a) $t_e^1 \leq t_s^2$, (b) the starting time of the intersection $\Theta(e_s^1, e_e^1)$ is not larger than then starting time of $\Theta(e_s^2, e_e^2)$; and (c) timestamps of consecutive edges e_e^1 and e_s^2 of top vertex u_3 increase.

It remains to construct two consecutive edges between u_1 and u_2. From $s_1 = \text{src}_u$, 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 $e_s^0 e_s^1$ are two consecutive edges 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 intersect. Note that 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; Section 3); here u_1 is the given source vertex u.

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

(II) We can similarly construct other consecutive edges 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 in G. we omit the details here. $\square$