Querying Relational Event Graphs using Colored Range Searching Data Structures*

Farah Chanchary, Anil Maheshwari, and Michiel Smid

School of Computer Science, Carleton University,
Ottawa, ON, K1S 5B6, Canada
farah.chanchary@carleton.ca,anil@scs.carleton.ca,michiel@scs.carleton.ca

Abstract. We present a general approach for analyzing structural parameters of a relational event graph within arbitrary query time intervals using colored range query data structures. Relational event graphs generally represent social network datasets, where each graph edge carries a timestamp. We provide data structures based on colored range searching to efficiently compute several graph parameters (e.g., density, neighborhood overlap, h-index).

Keywords: colored range searching, relational event graph, social network analysis, timestamp

1 Introduction

A *relational event (RE) graph* $G = (V, E)$ is defined to be an undirected graph with set of vertices V and a set of edges (or relational events) $E = \{e_k | 1 \leq k \leq m\}$ between pairs of vertices. We assume that each edge has a unique *timestamp*. We denote the timestamp of an edge $e_k \in E$ by $t(e_k)$. Without loss of generality, we assume that $t(e_1) < t(e_2) < \cdots < t(e_m)$. Given a relational event graph G, for a pair of integers $1 \leq i \leq j \leq m$, we define the *graph slice* $G_{i,j} = (V', E' = \{e_i \cup e_{i+1} \cup \cdots \cup e_j\})$, where V' is the set of vertices incident on edges of E'. In this paper, for a query time interval $[i, j]$, where $1 \leq i \leq j \leq m$, we are interested in answering questions about various graph parameters on the graph slice $G_{i,j}$.

A social network can naturally be represented by an RE graph, where each vertex of the graph represents an entity of the social network, and edges represent communication events between pair of entities occurred at some specific time. The RE graph model was first proposed by Bannister et al. [1]. They presented data structures to find the number of connected components, number of components containing cycles, number of vertices with some predetermined degree and number of reachable vertices on a time-increasing path within a query time window. Later, Chanchary and Maheshwari [6] presented data structures to solve subgraph counting problems for triangles, quadrangles and complete subgraphs in RE graphs. In this paper, we present a general approach to construct

* This research work was supported by NSERC and Ontario Graduate Scholarship.

data structures on a set of colored points in $\mathbb{R}^d$, ($d \geq 1$), that support colored (or generalized) range queries, and efficiently count and/or report various structural parameters of the underlying RE graph G within a query pair of indices $[i, j]$.

1.1 Preliminaries

We define some structural graph parameters that we want to compute using our data structures. One of the basic indicators for measuring graph structure is the *density* of a graph. It evaluates how close the graph is to a complete graph. The density of an undirected simple graph $G = (V, E)$ is defined as $D(G) = \frac{|E|}{\binom{|V|}{2}}$ [15].

In social networks, center vertices of k-stars are considered as *hubs*. In network analysis, hubs have been extensively studied as they are the basis of many tasks, for example web search and epidemic outbreak detection [2]. A *k-star* is defined to be a complete bipartite graph $K_{1,k}$, i.e., a tree with one internal node and k leaves. The *h-index* is the largest number h such that the graph contains h vertices of degree at least h. For any graph with m edges, $h = O(\sqrt{m})$ [1].

Embeddedness of an edge (u, v), denoted as $emb(u, v)$, in a network is the number of common neighbors the two endpoints u and v have, i.e., $emb(u, v) = |N(u) \cap N(v)|$[8]. Embeddedness of an edge (u, v) in a network represents the trustworthiness of its neighbors, and the confidence level in the integrity of the transactions that take place between two vertices u and v [8]. A *local bridge* in a graph G is an edge whose endpoints have no common neighbour. *Neighborhood overlap* of an edge (u, v), denoted as $NOver(u, v)$, is the ratio of the number of vertices who are neighbors of both u and v, and the number of vertices who are neighbors of only one of them [8].

$$NOver(u, v) = \frac{emb(u, v)}{|N(u) \cup N(v)| - emb(u, v) - 2} \tag{1}$$

In the denominator, u or v are not counted as the neighbor of one another. Neighborhood overlap of an edge represents the strength (in terms of connectivity) of that edge in its neighborhood. The neighborhood overlap of an entire graph G is defined as the average of the neighborhood overlap values of all the edges of G, i.e., $NOver(G) = \frac{1}{|E|} \sum_{k=1}^{|E|} NOver(e_k)$ [14]. Suppose $G = (V = A \cup B, E)$ is a bipartite graph. Neighborhood overlap of a pair of vertices $u, v \in A$ of G is defined as the following ratio [8].

$$NOver(u, v) = \frac{emb(u, v)}{|N(u) \cup N(v)| - emb(u, v)} \tag{2}$$

In social network analysis, this bipartite graph is known as the *affiliation network*. An affiliation network represents how entities of a network (i.e., vertices in A) are affiliated with other groups or activities (i.e., vertices in B).

Modularity of a network is a measure that quantifies the quality of a given network division into disjoint partitions or communities [15]. Many real world

complex networks, e.g., the world wide web, biological or social networks, demonstrate some forms of community structures that are important for various topological studies. The *modularity of a vertex pair* (u, v) is defined as $\frac{d(u)\times d(v)}{2|E|}$, where $d(u)$ is the degree of vertex u [15].

Analysis of the *diffusion phenomena* in graphs is a very widely studied research area in many communities including computer science, social sciences and epidemiology. Although the basic model of diffusion shares common properties across various disciplines, it is highly context dependent [3, 5, 9]. In particular, the spread of information in any social network requires some influence from a set of designated agents (vertices) and depends on the connectivity of the network. We want to solve the problem of counting and reporting influenced vertices in an RE graph slice using the following model.

Suppose, $G = (V, E)$ is an RE graph with n vertices and m edges with a fixed set of influential vertices $V' \subseteq V$. Let $f : V \to N$ be a function assigning thresholds values to vertices such that, (a) $f(v) = 0$ if v is an influential vertex; (b) $1 \leq f(v) \leq d(v)$ otherwise, where $d(v)$ is the degree of $v \in V$. A vertex can be influenced only if it is on a path of influence. A simpler model for counting influential vertices has been presented in [1].

Definition 1. *For a pair of vertices u and v, and a positive integer r, a path $\pi = (u = v_1, v_2, v_3, \cdots, v_k, v_{k+1} = v)$ is a path of influence with parameter r, if the following holds:*

1. *u is an influential vertex and v is a non-influential vertex*
2. *$v_2, v_3, \cdots, v_k$ are influenced vertices*
3. *$t(v_i, v_{i+1}) < t(v_{i+1}, v_{i+2})$*
4. *$t(v_k, v_{k+1}) - t(v_1, v_2) \leq r$.*

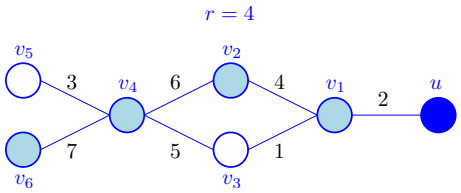

Fig. 1. A path of influence $\pi = (u, v_1, v_2, v_4, v_6)$ starting from an influential vertex u (shaded dark blue) to vertices v_1, v_2, v_4 and v_6 (shaded light blue) given $r = 4$. Vertices v_3 and v_5 (opaque) are not influenced by u since there is no path with increasing timestamps from u to these vertices.

Definition 2. *A vertex v is influenced with respect to r if either v is an influential vertex, i.e., $v \in V'$ or v is a non influential vertex and there are at least $f(v)$ edge-disjoint paths of influence with parameter r by which v can be reached from some influential vertices.*

We assume an influential vertex u can influence other vertices $v_1, v_2, \cdots, v_{k+1}$ on a path of influence π such that the time difference $t(v_k, v_{k+1}) - t(v_1, v_2) \leq r$, where $r > 0$. After r rounds, v_i's influence becomes inactive on other vertices on this particular path of influence. This is known as the degradation of influence in social networks. A similar model has been proposed by Gargano et al. [9] for non-temporal graphs.

1.2 New Results

Let G be a relational event graph consisting of m edges and n vertices, and let $q = [i, j]$ be an arbitrary query time interval, where $1 \leq i \leq j \leq m$. Our data structures can efficiently answer queries on computing the graph density, number of vertices, degrees of all vertices, k-stars, h-index, embeddedness of edges, average neighborhood overlap (for general and bipartite graphs) and the number of influenced vertices in the graph slice $G_{i,j}$. The contributions of this paper are summarized in Table 1. Some of the queries mentioned above are obtained by reducing the problem to a new colored range query problem stated as follows: Given a set of n colored points on the real line and a fixed threshold value k, we can construct an insertion-only persistent data structure in $O(n \log n)$ time using $O(n)$ space that can report those colors that have at least k elements within any query interval $q = [a, b]$ in $O(\log n + w)$ time, where w is the number of reported colors (Theorem 1).

Table 1. Summary of Results. Here, w is the size of the output, h is the h-index, s is the number of edges having neighboring edges, t is the number of edges with positive embeddedness, p is the number of vertex pairs having some common neighbors, k is the number of vertices having some neighbors in $G_{i,j}$, $a(G)$ is the arboricity of G, and $n = |V|$, $m = |E|$.

Problems	**Preprocessing time**	**Query time**	
$\|V_{i,j}\|$	$O(m+n)$	$O(\log n)$	(Theorem 2)
$D(G_{i,j})$	$O(m+n)$	$O(\log n)$	(Theorem 2)
$d(v \in V_{i,j})$	$O(m+n)$	$O(\log^2 n + w)$	(Theorem 3)
k-stars	$O(m+n)$	$O(\log n + w)$	(Theorem 4)
h-index (approx.)	$O(m+n)$	$O(\log^2 n + h \log n)$	(Theorem 5)
Embeddedness	$O(a(G)m)$	$O(\log^2 n + t \log n)$	(Theorem 6)
$NOver(G_{i,j})$	$O(mn)$	$O(\log^2 n + (t+s) \log n)$	(Theorem 6)
$NOver(G_{i,j})$-bipartite	$O(a(G)m)$	$O(\log^2 n + p \log n + k)$	(Theorem 7)
Influenced vertices	$O(m \log n)$	$O(\log n + w)$	(Theorem 8)

1.3 Organization

The rest of this paper is organized as follows. Section 2 presents some existing and new results on colored range query data structures. Section 3 presents the general approach for solving problems in RE graphs using colored range queries. In Section 4, we provide algorithms and their analysis for solving queries. Section 5 concludes this paper.

2 Colored Range Searching Data Structures

Colored range searching problems are variations of the standard range searching problems, where a set of colored objects (e.g., points, lines or rectangles) is given. Typically, the color of an object represents its category. In the generic instance of the range searching problems, a set of objects S is to be preprocessed into a data structure so that given a query range q, it can efficiently answer counting or reporting queries based on the intersection of q with the elements in S. In the colored version, the query should efficiently report those colors that have at least one object intersecting the query range. For example, we have a set of points $S \in \mathbb{R}^2$ and each point is colored by one of the four different colors, i.e., $C = \{c_1, c_2, c_3, c_4\}$ (see Figure 2). Suppose a query rectangle $q = [a, b] \times [c, d]$ is given. The standard colored range counting (reporting) query will count (report) the number of colors that have at least one point inside q. In this example, there are three colors (red, green and blue) that have points inside q, hence they will be reported. A different version namely *Type-2 counting problem* reports the number of points for each color intersected by q as a pair of values (c_k, $\#c_k$-colored points intersected by q). Following our example, the type-2 counting query will generate the output as $(red, 2), (green, 2), (blue, 1)$. A variety of colored range searching problems have been studied extensively, see for example [4, 10, 11, 13].

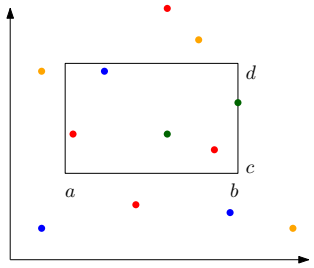

Fig. 2. Example of colored range queries.

2.1 *k*-threshold color queries

We wish to preprocess a set S of colored points on a horizontal line so that given a threshold k, where $k \geq 0$, an integer, and a query interval $q = [a, b]$, we can quickly report the colors that have at least k points in $q \cap S$. To solve this problem, we construct an insertion-only persistence based data structure. [1]

[1] In the conference version of this paper (CALDAM'17), we presented a data structure based on a chaining technique to solve this problem. The persistence based data structure presented here improves both total space and the query time by a factor of $O(\log n)$.

The Static Counting Problem: At first, we assume k is fixed and solve a similar k-threshold color counting problem where the query interval is of the form $q = [a, \infty)$. Suppose, S is a set of n colored points. For each color c, we sort all points of color c by non-decreasing x co-ordinate and let S_c be this sorted list. Let X be another sorted list, where each element X_c of X is the k-th largest element in S_c. For a query interval $[a, \infty)$, we start from the right end of list X, walk up till we reach a and report all points in between. Total space required for this structure is $O(n)$. Queries can be answered in time $O(w)$, where w is the number of reported colors.

Lemma 1. *Let $k \geq 1$ be a fixed integer. A set S of n colored points on the real line can be preprocessed into a data structure of size $O(n)$ such that the number of colors that have at least k points contained in any query interval $q = [a, \infty)$ can be reported in $O(w)$ time, where w is the number of reported colors. Total preprocessing time is $O(n \log n)$.*

The Insertion-only Persistent Structure: Now we solve the problem of k-threshold color queries for a query interval $q = [a, b]$. The idea is as follows. We first sort the list S of n colored points. Then we start with an empty list L. We insert the points of S, according to their sorted order, into L. During this sequence of insertions, we maintain the data structure of Lemma 1. A persistent version of this list will allow us to answer queries for arbitrary intervals $[a, b]$.

During the insertions, we maintain the following information: For each color c, L_c is a sorted list containing the k largest elements among all points having color c that have been inserted so far. In case fewer than k elements of color c have been inserted, L_c stores all these elements having color c. We also maintain the number $size(L_c)$ of elements in the list L_c. Note that $size(L_c) \leq k$ at any moment. We maintain a list L that stores the following elements (in sorted order): For each color c with $size(L_c) = k$, the list L stores the k-th largest element having color c. Finally, we maintain a balanced binary search tree T whose leaves store, in sorted order, the elements of the current list L. This tree will allow us to search and update the list L.

Initially, the list L, the tree T, and all lists L_c are empty. Moreover, $size(L_c)$ is zero for each color c.

When we insert a point p of color c, we add p at the end of L_c and increase $size(L_c)$ by one. In case $size(L_c) = k$, we insert p into T and L. In case $size(L_c) = k + 1$, we delete, from T, L, and L_c, the smallest element of L_c, we insert p into T and L, and we decrease $size(L_c)$ by one. See Algorithm 1 for details.

Sorting S takes $O(n \log n)$ time. Total preprocessing time required for inserting n points into tree T is at most $O(n \log n)$. The amount of memory changes in L, per insertion, is $O(1)$. Total space required to maintain T is $O(n)$. Given a query interval $[a, b]$, we do a binary search on b in the sorted sequence S. The result of this search gives us the version number of L in which we report all colors that occur at least k times in the query interval $[a, \infty)$. This gives a query time of $O(\log n + w)$, where w is the number of reported colors.

Algorithm 1: k-threshold(S, k)

```
Input : Set S of n colored points, a positive integer k.
Sort S in the non-decreasing order of the x-coordinate values of n colored points.
Set T, L and all L_c empty.
Set all size(L_c) = 0.
for each point p in S do
    Let c be the color of p.
    Add p at the end of L_c.
    Set size(L_c) = size(L_c) + 1.
    if size(L_c) = k then
        Insert p into T and L.
    if size(L_c) = k + 1 then
        Set q = the first element of L_c.
        Remove q from T, L and L_c.
        Insert p into T and L.
        Set size(L_c) = size(L_c) − 1.
```

Theorem 1. *Let $k \geq 1$ be a fixed integer. A set S of n colored points on the real line can be preprocessed into a persistent data structure of size $O(n)$ in time $O(n \log n)$ such that the number of colors that have at least k points contained in any query interval $q = [a, b]$ can be reported in $O(\log n + w)$ time, where w is the number of reported colors.*

3 General Approach for Modeling Problems

In this paper we present the following general approach to solve problems in RE graphs. Suppose an RE graph $G = (V, E)$ is given and we want to answer queries about some structural parameters of G. To solve each problem, we will define a set of colors $C = \{c_1, c_2, \cdots, c_p\}$, where each color $c_k \in C$ is encoded as an integer in the range $[1, p]$ for some integer p. We process each edge $e = (u, v)$ of G according to the timestamps of the edges in increasing order and scan either each adjacent edge of e or each neighboring vertex of both u and v. Depending on the problem in hand, our algorithm associates C either to the set of vertices V (i.e., $|C| = n$) or to the set of edges E (i.e., $|C| = m$). When each vertex $v_k \in V$ is associated with a color c_k, the algorithm scans each neighbor N_i of v_k, where $1 \leq i \leq d(v_k)$, generates a c_k colored point p, and assigns the timestamp $t(v_k, N_i)$ as p's coordinate value (see Figure 3(a)). Similarly, when each edge $e_k \in E$ is associated with a color c_k, the algorithm scans each adjacent edge M_i of e_k, and generates a c_k colored point p with the timestamp of M_i assigned as p's coordinate value (see Figure 4(b)). This general approach will be extended when we report influenced vertices or preprocess bipartite graphs. We explain these in later sections.

Now, we summarize the components of our general approach as follows. To solve any problem with our model, we need to specify the following components.

1. The set of colors C and the corresponding graph element (i.e., vertices, edges);
2. The representation of points in $\mathbb{R}^d$, where $d \geq 1$;
3. Appropriate colored range searching data structure to answer queries.

4 Algorithms for Answering Window Queries

4.1 Counting Vertices, Density, and Degrees of Vertices

Given an RE graph $G = (V, E)$ and a query time slice $[i, j]$, we want to find the number of vertices $|V_{i,j}|$, the density $D(G_{i,j})$, and report all the vertices with non-zero degrees in $V_{i,j}$.

Counting number of vertices $|V_{i,j}|$: We use a set $C = \{c_1, c_2, \cdots, c_n\}$ of n colors, where each c_k is associated with a vertex v_k, for $1 \leq k \leq n$. For each vertex $v_k \in V$, we maintain a linked list $adj[v_k]$, where each node of the list contains its neighboring vertex N_i, where $1 \leq i \leq d(v_k)$, and the timestamp of the edge between them, i.e., $t(v_k, N_i)$. We associate color c_k with all timestamps stored in $adj[v_k]$, for all $1 \leq k \leq n$. Now we have exactly $2|E|$ colored points on the real line (see Figure 3 for an example). Let this set of colored points be P. Using 1-dimensional colored range counting data structure on P with query interval $q = [i, j]$, we can find all distinct colors intersected by q. Each of these distinct colors represents a vertex having adjacent edges in $G_{i,j}$. Thus, we can report $|V_{i,j}|$ in $O(\log m) = O(\log n)$ time (by [11], Theorem 3.2) using a data structure of size $O(m \log n)$.

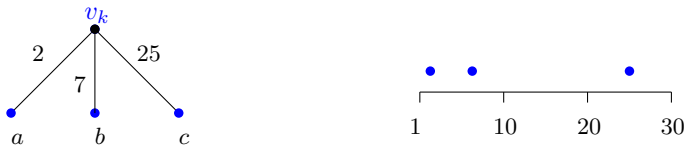

Fig. 3. (a) A vertex v_k and its three neighbors a, b and c. Since the timestamp $t(v_k, a)$ of the edge (v_k, a) is 2, (v_k, a) is shown as a c_k-colored point on the line with x-coordinate value = 2. (b) The set of all c_k colored points on the line.

Computing density $D(G_{i,j})$: This step requires computing the value of $|E_{i,j}|$ in addition to $|V_{i,j}|$. For any graph slice $G_{i,j}$, the value of $|E_{i,j}|$ can be computed in constant time since $|E_{i,j}| = j - i + 1$. We summarize these results as follows.

Theorem 2. *Given an RE graph G with m edges and n vertices, and a query time interval $[i, j]$, the problem of computing $|V_{i,j}|$ and the density $D(G_{i,j})$ of*

the graph slice can be reduced to 1-dimensional colored range queries in linear time. Queries can be answered in $O(\log n)$ time using $O(m \log n)$ space.

Reporting degrees of vertices $d(v)$: We want to report the degree of each vertex $v_k \in V$ such that v_k has some neighboring edges in $G_{i,j}$. So, we construct a 1-dimensional type-2 color counting data structure on P. This query will report all vertices that have non-zero degrees in $G_{i,j}$ in $O(\log n + w)$ time, where w is the number of reported vertices.

Theorem 3. *Given an RE graph G with m edges and n vertices, and a query time interval $[i, j]$, the problem of computing degrees of all vertices $v_k \in V_{i,j}$ can be reduced to 1-dimensional type-2 colored range queries in linear time. Queries can be answered in $O(\log n + w)$ time, where w is the number of reported vertices.*

4.2 Counting *k*-stars and *h*-index

Given an RE graph G with m edges and n vertices, a fixed threshold k and a query time slice $[i, j]$, we want to count all k-stars in $G_{i,j}$. We use a similar model as described in the previous section (see Figure 3). Each edge adjacent to a vertex v_k will have the color c_k. Thus we again have a set of colored points on the real line. By applying k-threshold color queries (Theorem 1) with query interval $q = [i, j]$ we can report all vertices that have at least k neighbors in $G_{i,j}$, hence are at the center of k-stars in $G_{i,j}$.

Theorem 4. *Given an RE graph G with m edges and n vertices, a query time slice $[i, j]$, and a fixed threshold value k, the problem of finding all k-stars in $G_{i,j}$ can be reduced to k-threshold color queries in linear time. Queries can be answered in $O(\log n + w)$ time, where w is the number of k-stars.*

Given an RE graph G with m edges and n vertices, and a query time slice $[i, j]$, we want to compute the h-index of $(G_{i,j})$. We use the same model and build the parameterized k-threshold color counting data structures to answer this query. A k-threshold color counting data structure reports the number of vertices (w) that have at least k neighbors in $G_{i,j}$. Therefore, if the number of reported vertices $w \geq k$, then the h-index of $G_{i,j}$ is at least k. Now we perform a set of decision problems to compute the h-index$(G_{i,j})$, i.e., *'Are there at least k vertices of degree k in $G_{i,j}$?'*

We query the k-threshold color counting data structure with parameter $k = 1$ and compare the number of colors (w) reported by this query with the value of k. We perform the following steps for $k = 1, (1+\epsilon), (1+\epsilon)^2, \cdots$ for up to $\log_{1+\epsilon} n$ blocks, where ϵ is an arbitrarily small positive constant:

1. If $w = k$, we return k as the h-index of $G_{i,j}$.
2. If $w > k$, we search using the data structure with the next value of k in the sequence.
3. Otherwise, if $k' < w < k$, where k' was the previous block size, we return w as the approximation to h-index.

In case we do not find the exact answer for h, we report an approximate h-index$(G_{i,j})$ with approximation ratio of $(1+\epsilon)$. Total preprocessing time required to build this data structure will be $O(n \log^2 n)$. The following theorem summarizes this result.

Theorem 5. *Given an RE graph G with m edges and n vertices, and a query time slice $[i, j]$, the problem of computing the h-index of $G_{i,j}$ can be reduced to the parameterized k-threshold color counting problem in linear time. Queries can be answered in $O(\log^2 n + h \log n)$ time with an approximation ratio of $(1+\epsilon)$, where ϵ is a small positive constant and h is the h-index of $G_{i,j}$.*

4.3 Computing Neighborhood Overlap and Embeddedness

We notice, from the definition of neighborhood overlap of an edge (u, v) that the numerator term represents the embeddedness of the edge (u, v) (i.e., the number of triangles containing the edge (u, v)) and the denominator is the number of edges adjacent to edge (u, v) minus the common edges (triangles). So, to answer the main query we need to solve two subproblems; i.e., for each edge $e_k \in G_{i,j}$, where $i \leq k \leq j$, we count (a) the number of edges adjacent to e_k, and (b) the number of triangles containing e_k. We describe below the preprocessing steps for these subproblems.

Counting the number of adjacent edges: We use a set of m colors, $C = \{c_1, c_2, \cdots, c_m\}$, where each color c_k will be associated with an edge e_k, for $1 \leq k \leq m$. Now, for each edge e_k we associate each adjacent edge of e_k with a color $c_k \in C$. Suppose, e_k has a set of adjacent edges $A(e_k) = \{e', e'', \cdots\}$. We create a c_k-colored point in $\mathbb{R}^2$ for each edge $e' \in A(e_k)$ with coordinate values $(t(e_k), t(e'))$ (see Figure 4). The total number of colored points is $\sum_{e=(u,v)}(d(u) + d(v)) = \sum_u d(u)^2 \leq (n-1)\sum_u d(u) = 2m(n-1) = O(mn)$. Using 2-dimensional type-2 counting query data structure (Theorem 1.4, [10]) with query interval $[i, j]$ the number of edges adjacent to each $e_k \in E_{i,j}$ can be found in $O(\log^2 mn + s \log mn) = O(\log^2 n + s \log n)$ time using $O(mn)$ space, where s is the total number of edges in $E_{i,j}$ having some neighboring edges.

Counting the number of triangles: We observe that we can preprocess G in $O(a(G)m)$ time to count the number of triangles in $G_{i,j}$ ([6], Theorem 2), where $a(G)$ is the arboricity of G. Arboricity is defined as the minimum number of edge-disjoint spanning forests into which G can be partitioned [12]. For a general connected graph G, $a(G) = O(\sqrt{m})$ [7]. Algorithm-1 presented in [6] represents each triangle of G as a point $p = (high, low) \in \mathbb{R}^2$, where $high$ (low) is the maximum (minimum) timestamp of the participating edges of that triangle. Now we modify the algorithm in [6] so that each point p in the original algorithm representing a triangle will now have three copies of itself and each copy will be associated with the color of each participating edge of the triangle (see Figure 5). There will be exactly $3\mathcal{K}$ colored points in $\mathbb{R}^2$, where $\mathcal{K}$ is the number of triangles in G. Using 2-dimensional type-2 range counting query data structure

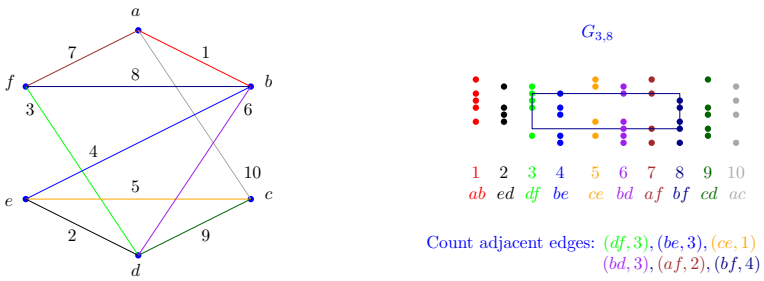

Fig. 4. (a) Relational event graph G, and (b) Number of adjacent edges in the graph slice $G_{3,8}$.

([4], Theorem 6.2) with query rectangle $[i, j] \times [i, j]$, the number of triangles containing edges e_k, where $(i \leq k \leq j)$, can be found in $O(\log^2 \mathcal{K} + t \log \mathcal{K})$ time, where t is the number of output edges. The maximum number of triangles in any graph G can be $O(n^3)$. So the query time can be re-defined as $O(\log^2 n + t \log n)$. From this result, we can also report (i) total number of local bridges $\#LB(G_{i,j}) = (j - i - t + 1)$ and (ii) value of embeddedness of any edge in $G_{i,j}$.

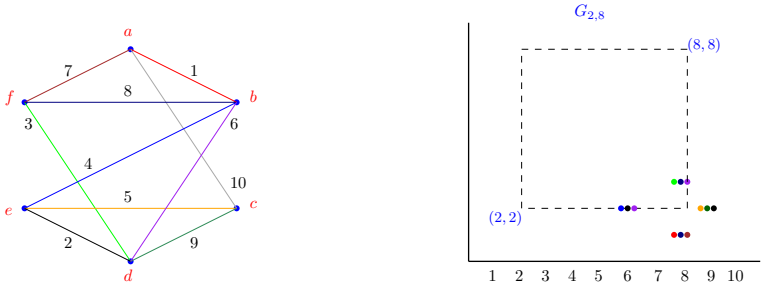

Fig. 5. (a) Relational event graph G, and (b) Number of triangles adjacent to edges in the graph slice $G_{2,8}$.

Analysis: For part (a), there are overall $O(mn)$ points representing all edge adjacencies in G. For part (b), identifying all triangles in G requires $O(a(G)m)$ preprocessing time. Thus, the total time required to reduce the problem of computing neighborhood overlap to colored range queries takes $O(a(G)m + mn) = O(mn)$ time. Total query time is $O(\log^2 n + t \log n) + O(\log^2 n + s \log n) = O(\log^2 n + (t + s) \log n)$. The following theorem summarizes the results.

Theorem 6. *Given an RE graph G with m edges and n vertices, and a query time slice $[i, j]$, the problem of computing the average neighborhood overlap of $G_{i,j}$ can be reduced to the colored range counting in $O(mn)$ time. $NOver(G_{i,j})$ can be computed in $O(\log^2 n + (t + s) \log n)$ time, where t is the number of edges*

in $G_{i,j}$ with positive embeddedness and s is the number of edges having some neighboring edges in $E_{i,j}$.

4.4 Neighborhood Overlap in Bipartite Graphs

Given a bipartite RE graph $G = (V = \{A \cup B\}, E)$ with m edges and n vertices, and a query time interval $[i, j]$, we now want to compute the neighborhood overlap of $G_{i,j}$. Recall from Definition 2 that for every pair of vertices $(u_1, u_2) \in A$, we need to compute (a) total number of common neighbors and (b) the total number of neighbors of the vertex pair.

Counting common neighbors of vertex pairs: To count all common neighbors of a pair of vertices $(u_1, u_2) \in A$ of a bipartite graph G, we use the edge searching technique applied in the quadrangle counting algorithm in [6]. The original algorithm represents a set of quadrangles sharing a common pair of vertices (two opposite corners of a rectangle) (u_1, u_2) as a tuple $(u_1, u_2, X = \{v_1, v_2, \cdots\})$ such that every pair of elements in X is connected to u_1 and u_2. In our case, for every tuple $u_1, u_2 \in A$, $X = \{v_1, v_2, \cdots\} \in B$, and we want to count $|X|$. Therefore, we modify the original algorithm presented in [6] such that the following holds.

1. The algorithm always starts searching with the vertices of set A.
2. Once a tuple $(u_1, u_2, X = \{v_1, v_2, \cdots\})$ is identified, (a) the pair of vertices (u_1, u_2) will correspond to a color $c_{u_1u_2}$, which is then added to the set of colors C, and (b) for each element $v_i \in X$ we store a point $(t(u_1, v_i), t(u_2, v_i)) \in R^2$ of color $c_{u_1u_2}$, where $t(u_1, v_i)$ is the timestamp of the edge (u_1, v_i).

Analysis: The modified algorithm identifies all neighbors of all pairs of vertices of A in $O(a(G)m)$ time [6]. For a complete bipartite RE graph G, $|C|$ will be at most $O(n^2)$, and there can be at most $O(n^3)$ colored points identified using this algorithm. Now given a query time slice $[i, j]$, we can find the total number of neighbors of each pair of vertices in the bipartite graph slice $G_{i,j}$ using 2-dimensional type-2 color counting data structure (Theorem 1.4, [10]). This query can be answered in $O(\log^2 n + p \log n)$ time, where p is the total number of vertex pairs having some common neighbors.

Counting all neighbors of vertex pairs: To count the total number of neighbors of each vertex pair, we count total degrees of all vertices $u_i \in A$ using 1-dimensional Type-2 colored range counting data structure in $O(\log n + k)$ time, where k is the total number of vertices having some neighbors in $G_{i,j}$ (see Theorem 3, Section 4.1). Thus the average neighborhood overlap of a bipartite graph slice $G_{i,j}$ can be computed in $O(\log^2 n + p \log n + k)$ time. The preprocessing time for the whole process is dominated by the process of counting common neighbors of all vertex pairs. Hence, total time required for preprocessing is $O(a(G)m)$. Thus we have the following results for bipartite RE graphs.

Theorem 7. *Given a bipartite RE graph G with m edges and n vertices, and a query time slice $[i, j]$, the problem of computing the average neighborhood overlap*

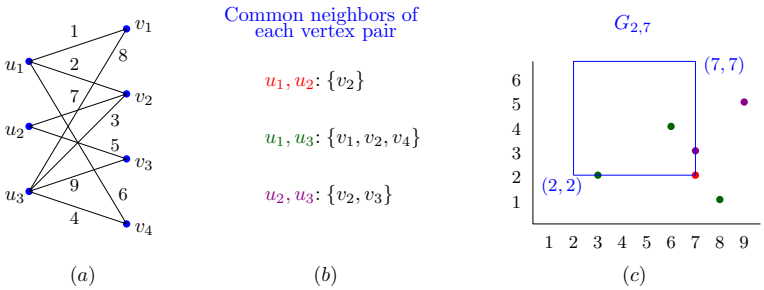

Fig. 6. (a) A bipartite RE graph (b) Set of neighbors adjacent to each vertex pairs, (c) Common neighbors for vertex pairs within query interval $q = [2, 7]$.

of a bipartite graph slice $G_{i,j}$ can be reduced to type-2 colored range searching problem in $O(a(G)m)$ time. $NOver(G_{i,j})$ can be computed in $O(\log^2 n + p \log n + k)$ time, where p is the total number of vertex pairs having some common neighbors in $G_{i,j}$ and k is the total number of vertices having some neighbors in $G_{i,j}$.

4.5 Reporting Influenced Vertices

Given an RE graph $G = (V, E)$ with a fixed set of influential vertices $V' \subseteq V$, a positive integer r, and a query time slice $[i, j]$, we want to find the total number of influenced vertices in $G_{i,j}$. We associate each vertex $v_k \in V$ with a color $c_k \in C = \{c_1, c_2, \cdots, c_n\}$. Each point representing an influenced vertex v_k will be colored with c_k.

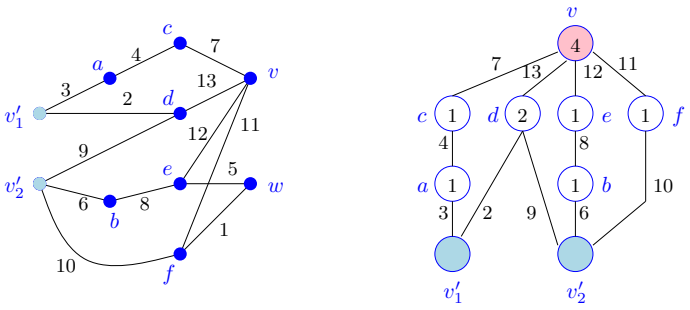

Fig. 7. (a) An RE graph with two influential vertices v'_1 and v'_2, (b) An influenced vertex v with $r = 3$ and $f(v) = 3$. The influence threshold $f(v)$ is mentioned inside the circle of each vertex v.

Preprocessing Step: Recall Definitions 1 and 2. For each vertex $v \in V$ having thresholds $f(v) \geq 1$, we associate it with a queue $Q[v]$ of size $f(v)$, initially empty. Queues are maintained by balanced binary search trees, so that the minimum

element of the queue can be retrieved when required. In addition, we maintain $\alpha(v)$, for all $v \in V$, where $\alpha(v)$ is the largest timestamp of an edge e such that e is adjacent to an influential vertex v' and there is a path of influence from v' to v. Initially $\alpha(v)$ is set to zero, for all $v \in V$. We process each edge $e_k = (u_k, v_k)$ according to the sequence of their timestamps, i.e., $t(e_1) < t(e_2) < \cdots < t(e_m)$, and set $\alpha(u_k)$ and $\alpha(v_k)$. We explain the process as follows. At time k, let $e_k = (u_k, v_k)$.

Case 1: Both u_k and v_k are influential vertices. Then u_k and v_k are already influenced. We set both $\alpha(u_k)$ and $\alpha(v_k)$ to k. We also store two points (k, k) of colors c_{u_k} and c_{v_k} respectively.

Case 2: u_k is an influential vertex and v_k is a non-influential vertex. We store a point (k, k) of color c_{u_k} and set $\alpha(v_k)$ to k.

Case 3: u_k is already an influenced non-influential vertex and $\alpha(u_k) = l$ has been set. If $k - l \leq r$, u_l influences v_k. If $\alpha(v_k) < \alpha(u_k)$ and we set $\alpha(v_k) = \alpha(u_k) = l$. Otherwise, we keep $\alpha(v_k)$ unchanged.

Each time we set $\alpha(v_k) = l$ for some vertex v_k, we add l to $Q[v_k]$. In case $Q[v_k]$ is full after inserting l, we remove the smallest element s from the queue, and store (k, s) as a point in $\mathbb{R}^2$ with color c_{v_k} (see Algorithm 2 for details). The c_{v_k}-colored point (k, s) implies that v_k is influenced by $f(v_k)$ paths of influence in the graph slice $G_{s,k}$. Vertices v_s and v_l are respectively the first and the last influential vertex that influence v_k in $G_{s,k}$. See Figure 7 for an example.

Preprocessing Analysis: All initializations take linear time (lines 1-4, Algorithm 2). There will be n queues, one for each vertex and each queue can have size at most equal to the degree of the vertex $d(v)$. Maintaining a queue for each vertex using balanced binary search tree requires $O(d(v) \log d(v))$ time, and both insertion and removal of a queue element requires $O(\log d(v))$ time. We visit each edge exactly once to set the values of α (lines 7 and 19), and to perform operations on queues (lines 13 and 20 for insertions, lines 15 and 22 for deletions). Thus, the total preprocessing time will be $\sum_v O(d(v) \log d(v)) = O(m \log n)$.

Now, we can reduce this problem to the colored range searching problem. In the worst case, every vertex can be influenced by each of its neighboring vertices. Thus, we have a set of at most $O(n^2)$ colored points in plane. Now our problem of reporting all influenced vertices within $G_{i,j}$ reduces to the problem of reporting the number of distinct colors in plane. By using 2-dimensional range reporting data structure with query rectangle $[0, j] \times [i, \infty]$, we can report all influenced vertices in $G_{i,j}$ in $O(\log n + w)$ time, where w is the number of influenced vertices ([10], Theorem 1.12).

Theorem 8. *Given an RE graph G with m edges and n vertices, and a query time slice $[i, j]$, the problem of reporting the total number of influenced vertices can be reduced to 2-dimensional colored range reporting problem in $O(m \log n)$ time. Queries can be answered in $O(\log n + w)$ time, where w is the number of influenced vertices in $G_{i,j}$.*

Algorithm 2: Influence(G, r)

Input : An RE graph G, a positive integer r.
Output: A colored point set $P \in R^2$.

for *each vertex* $v_k \in V$ *for* $k = 1$ *to* n **do**
  Set $\alpha(v_k) \leftarrow 0$.
  if $f(v_k) > 0$ **then**
    Set queue $Q[v_k] \leftarrow \emptyset$ with size $f(v_k)$.
for *each edge* $e_k = (u_k, v_k) \in E$ *for* $k = 1$ *to* m **do**
  if u_k *is an influential vertex* **then**
    Set $\alpha(u_k) \leftarrow k$ and $\alpha[v_k] \leftarrow k$.
    Store a point $(k, k) \in R^2$ of color c_{u_k}.
    if v_k *is a non-influential vertex* **then**
      if $f(v_k) = 1$ **then**
        Store a point $(k, k) \in R^2$ of color c_{v_k}.
      else
        Add $\alpha(u_k)$ to $Q[v_k]$
      if *the queue* $Q[v_k]$ *is full* **then**
        Remove the minimum element s from $Q[v_k]$.
        Store a point $(k, s) \in R^2$ of color c_{v_k}.
  if u_k *is an influenced vertex* **then**
    if $\alpha(u_k)(= l) > \alpha(v_k)$ *and* $(k - l) \leq r$ **then**
      Set $\alpha[v_k] \leftarrow l$.
      Add l to $Q[v_k]$.
      if *the queue* $Q[v_k]$ *is full* **then**
        Remove the minimum element s from $Q[v_k]$.
        Store a point $(k, s) \in R^2$ of color c_{v_k}.

5 Conclusion

In this paper, we have presented a general approach for solving queries regarding various structural parameters of relational event graphs in an arbitrary query time slice using colored range query data structures. Our approach models the reachability relationships between vertices and edges of a given RE graph by transforming them into colored points in $\mathbb{R}^d$, where $d \geq 1$. Subsequently, we reduce original problems into colored range searching problems to efficiently answer the queries. Following our model, we showed (a) how to compute the value of a specific structural parameter of any graph slice $G_{i,j}$; e.g., density, embeddedness, neighborhood overlap, h-index; and (b) how to count and report vertices in any $G_{i,j}$ with some specific properties; e.g., being influenced vertices or k-stars. We also presented a new persistence based k-threshold colored range searching data structure, where k is a fixed positive integer. However, this problem is still open if the value of k is given during the query time.

References

1. Bannister, M.J., DuBois, C., Eppstein, D., Smyth, P.: Windows into relational events: Data structures for contiguous subsequences of edges. In: Proceedings of the 24th ACM-SIAM SODA. pp. 856–864. SIAM (2013)
2. Berlingerio, M., Coscia, M., Giannotti, F., Monreale, A., Pedreschi, D.: The pursuit of hubbiness: Analysis of hubs in large multidimensional networks. Journal of Computational Science 2(3), 223–237 (2011)
3. Bettencourt, L.M., Cintrón-Arias, A., Kaiser, D.I., Castillo-Chávez, C.: The power of a good idea: Quantitative modeling of the spread of ideas from epidemiological models. Physica A: Statistical Mechanics and its Applications 364, 513–536 (2006)
4. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: New upper bounds for generalized intersection searching problems. In: ICALP, pp. 464–474. Springer (1995)
5. Centola, D., Macy, M.: Complex contagions and the weakness of long ties. American Journal of Sociology 113(3), 702–734 (2007)
6. Chanchary, F., Maheshwari, A.: Counting subgraphs in relational event graphs. In: WALCOM: Algorithms and Computation, vol. 9627, pp. 194–206. Springer (2016)
7. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM Journal on Computing 14(1), 210–223 (1985)
8. Easley, D., Kleinberg, J.: Networks, crowds, and markets: Reasoning about a highly connected world. Cambridge University Press (2010)
9. Gargano, L., Hell, P., Peters, J.G., Vaccaro, U.: Influence diffusion in social networks under time window constraints. Theoretical Computer Sc. 584, 53–66 (2015)
10. Gupta, P., Janardan, R., Rahul, S., Smid, M.: Computational geometry: Generalized (or colored) intersection searching. In: Handbook of Data Structures and Applications, 2nd Edition. In Press.
11. Gupta, P., Janardan, R., Smid, M.: Further results on generalized intersection searching problems: Counting, reporting, and dynamization. Journal of Algorithms 19(2), 282–317 (1995)
12. Harary, F.: Graph theory. Addison-Wesley, Reading, MA (1969)
13. Janardan, R., Lopez, M.: Generalized intersection searching problems. International Journal of Computational Geometry & Applications 3(01), 39–69 (1993)
14. Meghanathan, N.: A greedy algorithm for neighborhood overlap-based community detection. Algorithms 9(1), 8 (2016)
15. Santoro, N., Quattrociocchi, W., Flocchini, P., Casteigts, A., Amblard, F.: Time-varying graphs and social network analysis: Temporal indicators and metrics. In: 3rd AISB SNAMAS. pp. 32–38 (2011)