Effective Community Detection Over Streaming Bipartite Networks (Technical Report)

Nan Zhang
East China Normal University
Shanghai, China
51255902058@stu.ecnu.edu.cn

Yutong Ye
East China Normal University
Shanghai, China
52205902007@stu.ecnu.edu.cn

Yuyang Wang
East China Normal University
Shanghai, China
51265902008@stu.ecnu.edu.cn

Xiang Lian
Kent State University
Kent, Ohio, USA
xlian@kent.edu

Mingsong Chen
East China Normal University
Shanghai, China
mschen@sei.ecnu.edu.cn

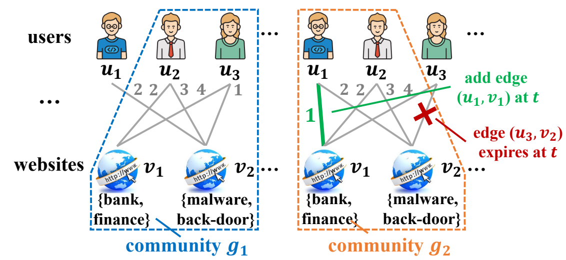
ABSTRACT

The streaming bipartite graph is extensively used to model the dynamic relationship between two types of entities in many real-world applications, such as movie recommendations, location-based services, and online shopping. Since it contains abundant information, discovering the dense subgraph with high structural cohesiveness (i.e., *community detection*) in the bipartite streaming graph is becoming a valuable problem. Inspired by this, in this paper, we study the structure of community on the butterfly motif in the bipartite graph. We propose a novel problem, named *Community Detection over Streaming Bipartite Network* (CD-SBN), which aims to retrieve qualified communities with user-specific query keywords and high structural cohesiveness at *snapshot* and *continuous* scenarios. In particular, we formulate the user relationship score in the weighted bipartite network via the butterfly pattern and define a novel (k, r, σ)-bitruss as the community structure. To efficiently tackle the CD-SBN problem, we design effective pruning strategies to rule out false alarms of (k, r, σ)-bitruss and propose a hierarchical synopsis to facilitate the CD-SBN processing. Due to the dynamic of streaming bipartite networks, we devise an efficient procedure for incremental graph maintenance. We develop an efficient algorithm to answer the snapshot and continuous CD-SBN query by traversing the synopsis and applying the pruning strategies. With extensive experiments, we demonstrate the efficiency and effectiveness of our proposed CD-SBN processing approach over real/synthetic streaming bipartite networks.

PVLDB Reference Format:
Nan Zhang, Yutong Ye, Yuyang Wang, Xiang Lian, and Mingsong Chen. Effective Community Detection Over Streaming Bipartite Networks (Technical Report). PVLDB, 14(1): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/L1ANLab/CD-SBN.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX



(a) G_{t-1} at timestamp $(t-1)$ (b) G_t at timestamp t

Figure 1: An example of community detection in streaming click bipartite networks G_t.

1 INTRODUCTION

Nowadays, the management of bipartite graphs that involve two distinct types of nodes (e.g., customers and products, visitors and check-in locations, users and clicked websites) has been extensively studied in many real applications such as Online Recommendation from user-product transaction graphs [11, 20, 37], location-based services with visitor-location check-in graphs [16, 19, 36, 44], and behavior analysis in user-Webpage click graphs [23, 29].

The *community detection* (CD) over such bipartite graphs has recently become increasingly important for bipartite graph analysis and mining [4, 6, 41, 46], which uncovers hidden and structurally cohesive bipartite subgraphs (or groups). The CD problem over bipartite graphs has many useful real applications, such as online product recommendation and marketing, user behavior analysis, malicious user group identification, and so on.

Although many prior works [1, 34, 35, 43] studied the CD problem in *static* bipartite graphs, real-world bipartite graphs are usually dynamically changing over time (e.g., updates of purchase transactions, check-ins, or website browsing). Therefore, in this paper, we will consider the *Community Detection over Streaming Bipartite Network* (CD-SBN), upon graph updates (e.g., edge insertions/deletions or edge weight changes).

Below, we give a motivating example of our CD-SBN problem in the application of detecting malicious user groups (communities) over a user-website click bipartite graph.

Example 1. **(Anomaly Detection for the Cybersecurity)** *Figure 1(a) illustrates an example of a clickstream bipartite network G_{t-1} at timestamp $(t-1)$, which contains 3 user vertices, $u_1 \sim u_3$, 2 website vertices, v_1 and v_2, and 5 edges (u_i, v_a) between two types of nodes, user u_i and website v_a (for $1 \le i \le 3$ and $1 \le a \le 2$). Here, each website vertex v_a (for $a = 1, 2$) has a set of keywords that represent* *the website's features (e.g., {bank, finance} for a bank website v_1). Moreover, each edge (u_i, v_a) (for $i = 1 \sim 3$ and $a = 1, 2$) is associated with an integer weight, indicating the frequency that user u_i accessed website v_a for the past five minutes.*

As shown in Figure 1(a), to detect malicious groups of users accessing sensitive/suspicious websites, the network security officer may want to specify some query keywords of websites (e.g., "bank" and "malware"), and identify some group (community), g_1, of users (e.g., u_2 and u_3) who frequently visit some common websites with query keywords (e.g., bank website v_1 and hacking website v_2). The officer will warn those suspicious users in the resulting community g_1 and/or take immediate actions (e.g., recording the evidence for the police).

Figure 1(b) shows the streaming bipartite network G_t at timestamp t, with a newly inserted edge (u_1, v_1) and an expired edge (u_3, v_2) (i.e., user u_3 has not visited website v_2 for the past five minutes). In this case, at timestamp t, the community in G_t is changed to g_2 (rather than g_1 at timestamp $(t-1)$), as circled by the orange dashed line. ■

In addition to the example above, the CD-SBN problem has many other real applications. For example, in the customer-product transaction bipartite graph, we can use the CD-SBN results to find communities of customers who recently have a purchasing behaviors for online advertising and marketing. Similarly, in the visitor-location check-in bipartite graph (e.g., from Yelp [18]), we can identify a group of visitors who frequently check in at some common *points of interest* (POIs) and provide them with group buying coupons/discounts (e.g., Groupon [14]).

In this paper, we formulate and tackle the CD-SBN problem, which obtains all bipartite communities with the user-specified query keywords (e.g., POIs' features) and high structural cohesiveness (e.g., small distance between any two users and high structural score of the community). In particular, we formally define the community semantics in the context of bipartite graphs, that is, a keyword-aware and structurally dense bipartite subgraph (i.e., a so-called (k, r, σ)-bitruss that contains query keywords, as will be described in Section 2.4). We consider both *snapshot* and *continuous* scenarios of our CD-SBN problem, which detect communities over a snapshot of streaming bipartite network G_t at timestamp t, or continuously monitor the CD-SBN answer set for each registered community constraint upon streaming graph updates, respectively.

Due to the large scale of bipartite graphs and rapid streaming graph changes, efficient processing of the snapshot and continuous CD-SBN is rather challenging. In order to tackle the challenges, in this paper, we propose effective pruning strategies with respect to the community constraints (e.g., query keywords, community radius, edge supports, and community scores) to significantly reduce the CD-SBN problem space. We design a hierarchical synopsis to effectively facilitate the candidate community search, and develop efficient snapshot and continuous algorithms to retrieve or incrementally maintain actual community answers (via our proposed synopsis and pruning methods), respectively.

In this paper, we make the following major contributions.

(1) We formally define the problem of the *community detection over streaming bipartite network* (CD-SBN) containing snapshot and continuous queries in Section 2.
(2) We present a framework to efficiently process the two types of queries with incremental updates of the bipartite streaming graph in Section 3.
(3) We design effective community-level pruning strategies to reduce the search space of the CD-SBN problem in Section 4.
(4) We propose a novel hierarchical synopsis to facilitate CD-SBN query processing and devise an efficient procedure for incremental graph maintenance in Section 5.
(5) We develop an efficient algorithm with effective synopsis-level pruning strategies to answer the snapshot CD-SBN query and another algorithm to maintain the result set of a continuous CD-SBN query with low computational cost in Section 6.
(6) We demonstrate the efficiency and effectiveness of our CD-SBN processing approach through extensive experiments over synthetic/real-world graphs in Section 7.

Section 8 reviews the related work on community search/ detection on unipartite graphs and static/streaming bipartite graphs. Finally, Section 9 concludes this paper.

2 PROBLEM DEFINITION

Section 2.1 provides the data model for streaming bipartite networks. Section 2.2 defines the basic units that indicate the relationship between two users. Section 2.3 introduces the *relationship score* and the (k, r, σ)-bitruss to measure the cohesiveness of a subgraph. Section 2.4 formally defines our problem of the *community detection over streaming bipartite networks* (CD-SBN).

2.1 Streaming Bipartite Networks

Bipartite Graphs: We first define an undirected and weighted bipartite graph as follows.

Definition 1. ***(Bipartite Graph,** G) A bipartite graph, G, is represented by a quadruple $(U(G), L(G), E(G), \Phi(G))$, where $U(G)$ and $L(G)$ are two disjoint sets (types) of vertices, $E(G)$ is a set of edges, $e_{u,v}$, between vertices $u \in U(G)$ and $v \in L(G)$, and $\Phi(G)$ is a mapping function: $U(G) \times L(G) \rightarrow E(G)$.*

Each vertex $v \in L(G)$ has a set, $v.K$, of keywords, and each edge $e_{u,v} \in E(G)$ is associated with a weight $w_{u,v}$.

Bipartite graphs have been widely used in many real applications, such as movie recommendations for users [15], recommending *places of interest* (POIs) to tourists [21], or online product shopping recommendation for customers [7, 27, 42].

For simplicity, this paper considers vertices u in the upper vertex layer $U(G)$ as users in real-world applications, and vertices v in the lower vertex layer $L(G)$ as items that are associated with keyword sets $v.K$ such as movie types, POI features, and product descriptions. Moreover, each edge $e_{u,v}$ from user u to item v is associated with a weight $w_{u,v}$, indicating the user-item interaction frequency such as # of movie views for movie recommendations, # of POI visits by users, # of product purchases for online shopping, and so on.

Streaming Bipartite Networks: Below, we introduce the bipartite network in the streaming scenario.

Definition 2. ***(Streaming Bipartite Network,** G_t) A streaming bipartite network G_t consists of an initial bipartite graph $G_0 = (U(G_0), L(G_0), E(G_0), \Phi(G_0))$, and an ordered sequence of update items $S = \{p_1, p_2, \ldots, p_t, \ldots\}$, where each item $p_t = (e_{u,v}, t)$ indicates an update operator of weight $w_{u,v}$ for an inserted edge $e_{u,v}$ arriving at timestamp t. A sliding window, W_t, of size s in a streaming bipartite network G_t contains the most recent s update items p_{t-s+1}, p_{t-s+2}, ..., and p_t, over bipartite graph G_t, where t is the current timestamp.*

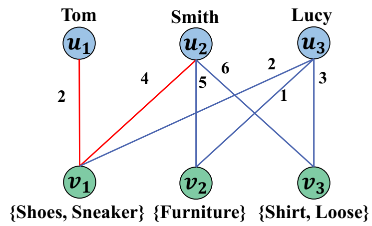


Figure 2: An example of basic patterns over bipartite network.

In Definition 2, we consider a sliding window, W_t, of s update items over a streaming bipartite network G_t. Each update item can be the insertion of a new edge/interaction between users and items (or equivalently the increase of the edge weight $w_{u,v}$).

After applying update items in sliding window W_t to the bipartite graph G, we can obtain the latest snapshot, G_t, of the streaming bipartite network at timestamp t.

Sliding Window Maintenance of Streaming Bipartite Networks: We maintain the sliding window W_t upon dynamic updates in G_t. When a new item p_{t+1} arrives at timestamp $(t+1)$, we will add p_{t+1} to W_t and remove the expired item p_{t-s+1} from W_t, which yields a new sliding window W_{t+1}.

Insertion of a New Item p_{t+1}. For the insertion of a new item p_{t+1} = $(e_{u,v}, t+1)$, we consider two cases:

- When both vertices u and v exist in G_t (i.e., $u \in U(G_t)$ and $v \in L(G_t)$), we add edge $e_{u,v}$ to G_t and increment edge weight $w_{u,v}$ by 1 (i.e., $w_{u,v} = w_{u,v} + 1$), and;
- When either of vertices u and v does not exist in G_t (i.e., $u \notin U(G_t)$ and/or $v \notin L(G_t)$), we add new vertex(es) u and/or v to $U(G_t)$ and/or $L(G_t)$, respectively, insert a new edge $e_{u,v}$ into G_t, and set edge weight $w_{u,v}$ to 1 (i.e., $w_{u,v} = 1$).

Expiration of an Old Item p_{t-s+1}. When an old item p_{t-s+1} = $(e_{u,v}, t-s+1)$ expires, we decrement the edge weight $w_{u,v}$ by 1 (i.e., $w_{u,v} = w_{u,v} - 1$). If $w_{u,v} = 0$ holds, we remove the edge $e_{u,v}$ from G_t (keeping two ending vertices u and v).

2.2 Basic Patterns in the Bipartite Graph

In this subsection, we give the definitions of basic patterns, *wedge* and *butterfly* [13], in the bipartite graph G.

Wedge: A wedge is formally defined as follows:

Definition 3. ***(Wedge,** $\angle(u_i, v, u_j)$) Given two user vertices $u_i, u_j \in U(G)$ and an item vertex $v \in L(G)$ in a bipartite graph G, a wedge [13], $\angle(u_i, v, u_j)$, is a path $u_i \to v \to u_j$, where u_i, v, and u_j are called start-, middle-, and end-vertex of the wedge, respectively. The weight of a wedge $\angle(u_i, v, u_j)$ is given by:*

$$w_{\angle(u_i,v,u_j)} = \min\{w_{u_i,v}, w_{u_j,v}\}. \tag{1}$$

Based on Definition 3, a wedge $\angle(u_i, v, u_j)$ in the bipartite graph can be two audiences watching the same movie, two travelers visiting the same site, or two customers purchasing the same product.

The weight of the wedge $\angle(u_i, v, u_j)$ takes the minimum edge weights between $w_{u_i,v}$ and $w_{u_j,v}$, indicating the intensity (or frequency) of *maximally common* interest/behavior v (e.g., movie, POI site, or product) between two users u_i and u_j.

In the example of Figure 2, $\angle(u_1, v_1, u_2)$ is one of the wedges, with weight $w_{\angle(u_1,v_1,u_2)} = \min\{2, 4\} = 2$.

Butterfly: Two wedges form a *butterfly*, as defined below:

Definition 4. ***(Butterfly [13],** $\bowtie (u_i, u_j, v_a, v_b)$) Given two user vertices $u_i, u_j \in U(G)$ and two item vertices $v_a, v_b \in L(G)$ in a bipartite graph G, a butterfly, $\bowtie (u_i, u_j, v_a, v_b)$, is a complete bipartite subgraph containing wedges $\angle(u_i, v_a, u_j)$ and $\angle(u_i, v_b, u_j)$.*

The butterfly structure (as given in Definition 4) has been widely used in many algorithms and applications [25, 38, 45] to detect communities with high cohesiveness in bipartite graphs. In Figure 2, $\bowtie (u_2, u_3, v_2, v_3)$ is an example of a butterfly.

The Score of a Butterfly: The butterfly is a basic graphlet pattern of bipartite graphs. Below, we define a *butterfly score* to measure the relationship among user and item vertices in the butterfly.

Definition 5. ***(The Butterfly Score)** The score, $w_{\bowtie(u_i,u_j,v_a,v_b)}$, of a butterfly $\bowtie (u_i, u_j, v_a, v_b)$ is given by:*

$$\begin{aligned} w_{\bowtie(u_i,u_j,v_a,v_b)} &= w_{\angle(u_i,v_a,u_j)} \cdot w_{\angle(u_i,v_b,u_j)} \\ &= \min\{w_{u_i,v_a}, w_{u_j,v_a}\} \cdot \min\{w_{u_i,v_b}, w_{u_j,v_b}\}. \end{aligned} \tag{2}$$

In Definition 5, we consider two wedges $\angle(u_i, v_a, u_j)$ and $\angle(u_i, v_b, u_j)$ in the butterfly $\bowtie (u_i, u_j, v_a, v_b)$, and define the score of the butterfly by multiplying their corresponding weights, that is, $\min\{w_{u_i,v_a}, w_{u_j,v_a}\} \cdot \min\{w_{u_i,v_b}, w_{u_j,v_b}\}$. Intuitively, we treat the weight of each wedge as the frequency of duplicate wedges, and the score of the butterfly is given by the count of duplicate butterflies (i.e., the multiplication of these two wedge weights).

For the previous example in Figure 2, the score, $w_{\bowtie(u_2,u_3,v_2,v_3)}$, of the butterfly $\bowtie (u_2, u_3, v_2, v_3)$ can be calculated by: $\min\{5, 1\} \cdot \min\{6, 3\} = 1 \times 3 = 3$.

2.3 The (k, r, σ)-Bitruss Community in a Bipartite Graph

In this subsection, we first define the *user relationship score* between any two user vertices u_1 and u_2 in a bipartite graph G and then formalize the (k, r, σ)-bitruss community with high structural cohesiveness in G.

The Relationship Score of a User Vertex Pair: Based on the score of a butterfly (as given in Eq. (2)), we can define the *user relationship score* of a user vertex pair (u_1, u_2) as follows.

Definition 6. ***(User Relationship Score,** $score_{u_i,u_j}$) Given two vertices $u_i, u_j \in U(G_t)$ in a bipartite graph G_t, the user relationship score, $score_{u_i,u_j}$, between u_i and u_j in G_t can be calculated by:*

$$score_{u_i,u_j}(G_t) = \sum_{\forall v_a, v_b \in L(G_t)} w_{\bowtie(u_i,u_j,v_a,v_b)}. \tag{3}$$

In Definition 6, the user relationship score $score_{u_i,u_j}$ between two users u_i and u_j is defined as the summed butterfly score for all butterflies containing vertices u_i and u_j. A higher relationship score indicates a stronger relationship between users u_i and u_j.

As illustrated in Figure 2, the relationship score, $123score_{u_2,u_3}$, between users u_2 and u_3 is given by: $w_{\bowtie(u_2,u_3,v_1,v_2)} + w_{\bowtie(u_2,u_3,v_1,v_3)} + w_{\bowtie(u_2,u_3,v_2,v_3)} = \min\{4, 2\} \cdot \min\{5, 1\} + \min\{4, 2\} \cdot \min\{6, 3\} + \min\{5, 1\} \cdot \min\{6, 3\} = 2 \times 1 + 2 \times 3 + 1 \times 3 = 11$.

The $(\mathbf{k}, \mathbf{r}, \sigma)$-Bitruss: We next define a (k, r, σ)-bitruss community in the bipartite graph G with high structural and user relationship cohesiveness.

Definition 7. *((k, r, σ)**-Bitruss**) Given a bipartite graph $G_t = (U(G_t), L(G_t), E(G_t), \Phi(G_t))$, a center vertex $u_c \in U(G_t)$, a threshold k of butterfly number, a maximum radius r, and a threshold, σ, of the user relationship score, a (k, r, σ)-bitruss is a connected subgraph, g, of G_t (denoted as $g \subseteq G_t$), such that:*

- *(Support) for each edge $e_{u,v} \in E(g)$, the edge support $sup(e_{u,v})$ (defined as the number of butterflies containing edge $e_{u,v}$) is not smaller than k;*
- *(Radius) for any user vertex $u_i \in U(g)$, we have $dist(u_c, u_i) \leq 2r$, and;*
- *(Score) for any two user vertices $u_i, u_j \in \angle(u_i, v, u_j)$, we have $score_{u_i,u_j}(g) \geq \sigma$,*

where $dist(u_c, u_i)$ is the shortest path distance between u_c and u_i in bipartite subgraph g.

2.4 The Problem of the Community Detection Over Streaming Bipartite Network (CD-SBN)

The CD-SBN Problem: We are now ready to define the problem of detection for communities over a streaming bipartite network upon dynamic updates.

Definition 8. ***(Community Detection Over Streaming Bipartite Network, CD-SBN)*** *Given a streaming bipartite network G_t, a support threshold k, a maximum radius r, a threshold, σ, of the user relationship score, and a set, Q, of query keywords, the problem of the community detection over streaming bipartite network (CD-SBN) retrieves a result set, R, of subgraphs g_i of G_t (i.e., $g_i \subseteq G_t$), such that:*

- *(Keyword Relevance) for any item vertex $v_i \in L(g_i)$, its keyword set $v_i.K$ must contain at least one query keyword in Q (i.e., $v_i.K \cap Q \neq \emptyset$), and;*
- *(Structural and User Relationship Cohesiveness) bipartite subgraph g_i is a (k, r, σ)-bitruss (as given in Definition 7).*

Note that, the CD-SBN problem over streaming bipartite network in Definition 8 has two scenarios, *one-time snapshot* and *continuous CD-SBN* versions. In particular, the one-time snapshot CD-SBN problem finds CD-SBN communities over a snapshot, G_t, of a streaming bipartite graph at timestamp t, whereas the continuous CD-SBN continuously monitors CD-SBN communities upon edge updates (i.e., S) in G_t, for some registered community constraints in Definition 8. The former version can be used to obtain initial CD-SBN community answers when CD-SBN predicates are online specified, and the latter one dynamically maintains CD-SBN community results upon graph updates.

Challenges: The processing of snapshot/continuous CD-SBN queries is quite challenging. One straightforward method to tackle this problem is to enumerate all community candidates and check whether or not these candidates satisfy the CD-SBN constraints. However, this method is quite inefficient. First, the checking of CD-SBN community predicates is very costly and involves the constraints of keywords, graph structural cohesiveness, and user relationship scores. Second, there are many possible CD-SBN community candidates, which incurs high computation costs to enumerate and refine. Third, upon graph updates over a sliding window W_t, it is also challenging to incrementally maintain the dynamically changing graph G_t and efficiently update the CD-SBN query answers. Table 1 shows the commonly used symbols and their descriptions in this paper.

3 THE CD-SBN PROCESSING FRAMEWORK

Figure 1 illustrates a CD-SBN processing framework for answering snapshot/continuous CD-SBN queries, which consists of three components, *initialization*, *graph incremental maintenance*, and *CD-SBN query processing*.

Table 1: Symbols and descriptions.

Symbol	Description
G	a bipartite graph
G_t	a streaming bipartite network at timestamp t
$U(G)$	a set of user vertices u in the upper layer of bipartite graph G
$L(G)$	a set of item vertices v in the lower layer of bipartite graph G
$E(G)$	a set of edges $e_{u,v}$ for $u \in U(G)$ and $v \in L(G)$
S	an ordered sequence of update items $p_t = (e_{u,v}, t)$
u (or u_i, u_j)	a user vertex in $U(G)$ of bipartite graph G
v (or v_a, v_b)	an item vertex in $L(G)$ of bipartite graph G
W_t	a sliding window of size s at timestamp t
$w_{u,v}$	the edge weight of edge $e_{u,v}$
$\angle(u_i, v, u_j)$	a wedge with $u_i, u_j \in U(G)$ and $v \in L(G)$
$\bowtie(u_i, u_j, v_a, v_b)$	a butterfly with $u_i, u_j \in U(G)$ and $v_a, v_b \in L(G)$
$w_{\bowtie(u_i,u_j,v_a,v_b)}$	the score of a butterfly $\bowtie(u_i, u_j, v_a, v_b)$
k	the support threshold for a (k, r, σ)-bitruss
$sup(e)$	the number of butterflies containing edge e
r	the maximum radius for a (k, r, σ)-bitruss
σ	the relationship score threshold for a (k, r, σ)-bitruss
$v_i.K$	the keyword set for each vertex $v_i \in L(G)$

To support the CD-SBN stream processing, the *initialization component* prepares some auxiliary data over initial bipartite graph G_0 at timestamp 0, including a keyword bit vector $v.BV$ for hashing keywords in item vertices, wedge-score-related data X_{u_i,u_l} and Y_{u_i,u_l} for the butterfly score calculation, support upper bound $ub\_sup(e_{u,v})$, and synopses, Syn (lines 1-4).

At each timestamp t, the sliding window W_t includes a new item p_t (i.e., the edge insertion) and removes an expired item p_{t-s} (i.e., the edge deletion). Therefore, the *graph incremental maintenance component* will compute an incremental update of wedge-score-related data, $\Delta X_{u_i,u_j}$ and $\Delta Y_{u_i,u_j}$ (given by Eqs. (7) and (8), respectively), and obtain the updated wedge-score-related data X_{u_i,u_j} and Y_{u_i,u_j} at timestamp t (lines 5-6). Similarly, for those affected edges e' (due to edge insertion/deletion), we will also update support upper bounds $ub\_sup(e')$ (line 7) and synopsis Syn (line 8).

For the *CD-SBN query processing component*, the snapshot CD-SBN algorithm traverses the synopsis Syn to retrieve candidate communities g, by applying our proposed pruning strategies, and refines the resulting candidate communities g to obtain/return actual CD-SBN answers in R (lines 9-12). Moreover, the continuous CD-SBN algorithm incrementally maintains CD-SBN community answers in R by removing those communities (with the expired edge) and inserting new CD-SBN communities (due to the edge insertion) (lines 13-15).

4 PRUNING STRATEGIES

This section details the community-level pruning strategies (see Section 3) to reduce the search space of the CD-SBN problem.

4.1 Keyword Pruning

Based on the CD-SBN problem definition (as given Definition 8), any item vertex v_i in the lower layer of a candidate bipartite subgraph g should contain at least one query keyword in Q. In the sequel, we present an effective *keyword pruning*, which filters out item vertices v_i in subgraph g that do not contain any query keywords in Q.

Lemma 1. ***(Keyword Pruning)*** *Given a query keyword set Q and a candidate bipartite subgraph g, any item vertex $v_i \in L(g)$ can be safely pruned from the subgraph g, if $v_i.K \cap Q = \emptyset$ holds, where $v_i.K$ is the keyword set associated with item vertex v.*

Algorithm 1: The CD-SBN Query Process Framework

Input: i) a streaming bipartite network G_t; ii) a set, Q, of query keywords; iii) a support threshold k; iv) a maximum radius r; v) a threshold, σ, of user relationship score

Output: a set, R, of CD-SBN communities

```
// Initialization (t = 0)
hash keywords in v.K into a keyword bit vector v.BV for each item vertex v ∈ L(G_0)
calculate the wedge-score-related data X_{u_i,u_j} and Y_{u_i,u_j} (as given in Eq. (6)) for each pair of user vertices u_i, u_j ∈ U(G_0)
compute the support upper bound ub_sup(e_{u,v}) for each edge e_{u,v} ∈ E(G_0)
construct a hierarchical synopsis, Syn, over G_0
for each timestamp t (> 0) do
    // Graph Incremental Maintenance
    compute ΔX_{u_i,u_j} and ΔY_{u_i,u_j} (as given in Eqs. (7) and (8), resp.) and obtain the updated X_{u_i,u_j} and Y_{u_i,u_j}
    update support upper bounds ub_sup(e') for relevant edges e' (upon edge insertion of p_t and deletion of p_{t-s})
    update synopsis Syn
    // CD-SBN Query Processing
    for each snapshot CD-SBN query do
        traverse the synopsis Syn by applying keyword, support, and score upper bound pruning strategies to retrieve candidate communities g
        refine candidate communities g and obtain actual CD-SBN results R
        return R
    for each continuous CD-SBN query do
        remove the expired communities from R
        insert new CD-SBN communities into R
```

Proof. If $v_i.K \cap Q = \emptyset$ holds for an item vertex $v_i \in L(g)$, then vertex v_i does not contain any query keyword in Q, which violates the keyword relevance constraint (as given in Definition 8). Hence, the item vertex v_i cannot be in the CD-SBN subgraph answer and can thus be safely pruned, which completes the proof. □

4.2 Support Pruning

As mentioned in Definition 8, since the CD-SBN community should be a (k, r, σ)-bitruss, the support, $sup(e_{u,v})$, of edge $e_{u,v}$ should not be smaller than k (i.e., $sup(e_{u,v}) \geq k$). Assume that we can obtain an upper bound, $ub\_sup(e_{u,v})$, of support $sup(e_{u,v})$ for an edge $e_{u,v} \in E(g)$. We provide the following *support pruning* lemma to rule out edges in a candidate bipartite subgraph g with low supports.

Lemma 2. ***(Support Pruning)*** *Given a support threshold k and a candidate bipartite subgraph g, the edge $e_{u,v} \in E(g)$ can be safely pruned from subgraph g if it satisfies $ub\_sup(e_{u,v}) < k$.*

Proof. According to Definition 7, the support, $sup(e_{u,v})$, is defined as the number of butterflies containing a specific edge. Thus, each edge e must be reinforced by at least k butterflies in a (k, r, σ)-bitruss. Since $ub\_sup(e_{u,v})$ is an upper bound of support $sup(e_{u,v})$, we have $sup(e_{u,v}) \leq ub\_sup(e_{u,v})$. Moreover, from the assumption of the lemma that $ub\_sup(e_{u,v}) < k$ holds, by the inequality transition, we have $sup(e_{u,v}) \leq ub\_sup(e_{u,v}) < k$. Hence, the support $sup(e_{u,v})$ of the edge $e_{u,v}$ is below the threshold k, and can be safely pruned, which completes the proof. □

Discussions on How to Obtain the Upper Bound of Support $\mathbf{ub\_sup(e_{u,v})}$. In the offline phase, the upper bound of support needs to be computed to enable the support pruning. Since part of item vertices in data graph G_t will be filtered out by the *keyword pruning*, where the rest of the vertices induce a subgraph g, the number of butterflies containing the edge $e_{u,v}$ in g is smaller than or equal to that in G_t. Thus, the support of $e_{u,v}$ in G_t can be considered as a supper upper bound $ub\_sup(e_{u,v})$ in subgraph g defined below.

$$ub\_sup(e_{u,v}) = \sum_{\forall v_x \in N(u)-\{v\}} |N(v) \cap N(v_x) - \{u\}|, \tag{4}$$

where $N(v)$ is the neighbor vertices set of vertex v. With $u \in U(G_t)$ and $v \in L(g)$, the number of butterflies containing $e_{u,v}$ in G_t can be calculated as the sum of the number of common neighbors for v and each neighbor v_x of u.

4.3 Layer Size Pruning

Since the support of a bipartite subgraph is closely related to the subgraph size, we can utilize filter out a candidate community g with small sizes of upper/lower layers (i.e., $|U(g)|$ and $|L(g)|$), with respect to the support threshold k.

Lemma 3. ***(Layer Size Pruning)*** *Given a support threshold k and a candidate bipartite subgraph $g = (U(g), L(g), E(g), \Phi(g))$, subgraph g can be safely pruned, if it satisfies the condition that $(|U(g)| - 1) \cdot (|L(g)| - 1) < k$.*

Proof. According to Definition 7, the support of each edge e in $(k, r, \sigma) - bitruss$ must be higher than the support threshold k, which means that each edge e must be contained by at least k butterflies. For each edge, e_{u_i,v_a}, any pair of other user vertex and item vertex, (u_x, v_y), can form a butterfly, $\bowtie (u_i, u_x, v_a, v_y)$, with the two ending vertices of edge e_{u_i,v_a}. Assume that g is a fully connected bipartite graph, the edge e_{u_i,v_a} is supported by pairs of vertices $|U - \{u_i\}| \times |L - \{v_a\}|$ so that its support is $sup(e_{u_i,v_a}) = (|U(g)| - 1) \cdot (|L(g)| - 1)$. However, not every pair of vertices is connected to u_i or v_a in general, hence it holds that $sup(e_{u_i,v_a}) \leq (|U(g)| - 1) \cdot (|L(g)| - 1)$. If $(|U(g)| - 1) \cdot (|L(g)| - 1) < k$ holds, we have $sup(e_{u_i,v_a}) \leq (|U(g)| - 1) \cdot (|L(g)| - 1) < k$. Thus, edge e_{u_i,v_a} can be safely pruned, which completes the proof. □

4.4 Radius Pruning

In the definition of (k, r, σ)-bitruss, the radius parameter r ensures that the distances between the center vertex and other user vertices are smaller than or equal to $2r$. Thus, we devise a radius pruning strategy to discard a user vertex u_i whose distance, $dist(u_i, u_q)$, to the center vertex u_q is greater than $2r$.

Lemma 4. ***(Radius Pruning)*** *Given a candidate bipartite subgraph g (centered at user vertex u_q) and a radius r, a user vertex $u_i \in U(g)$ can be safely pruned from subgraph g, if it holds $dist(u_i, u_q) > 2r$.*

Proof. For the candidate subgraph g centered at the user vertex u_q, if $dist(u_i, u_q) > 2r$ holds for a user vertex $u_i \in U(g)$, then u_i violates the radius constraint. Therefore, we should prune the user vertex u_i from g, which completes the proof. □

4.5 Score Upper Bound Pruning

Based on Definition 7, any pair of user vertices belonging to a wedge in a (k, r, σ)-bitruss must have the user relationship score greater than or equal to σ, where σ is a score threshold specified by users online. In the sequel, we propose a *score upper bound pruning* strategy to eliminate the user vertex(es) with low scores.

Lemma 5. ***(Score Upper Bound Pruning)*** *Given a community g and a candidate user vertex $u_i \in U(g)$, the user vertex u_i can be safely pruned, if there exists a user vertex $u_j \in U(g)$ and a wedge $\angle(u_i, v, u_j) \subseteq g$, such that $ub\_score_{u_i,u_j}(g) < \sigma$ holds, where $ub\_score_{u_i,u_j}(g)$ is an upper bound of score $score_{u_i,u_j}(g)$.*

Proof. Given a candidate user vertex u_i and a user vertex $u_j \in U(g)$, since $ub\_score_{u_i,u_j}(g)$ is a score upper bound, it holds that $score_{u_i,u_j}(g) \leq ub\_score_{u_i,u_j}(g)$. From the assumption of the lemma, if $ub\_score_{u_i,u_j}(g) < \sigma$ holds, by the inequality transition, we have $score_{u_i,u_j}(g) \leq ub\_score_{u_i,u_j}(g) < \sigma$, violating the score constraint of the (k, r, σ)-bitruss (given in Definition 7). Thus, the candidate user vertex u_i can be safely pruned, which completes the proof. □

Discussions on How to Obtain the Score Upper Bound, $ub\_score_{u_i,u_j}(g)$: To enable the score upper bound pruning, we need to calculate the score upper bound, $ub\_score_{u_i,u_j}(g)$, for the user pair (u_i, u_j) in subgraph g that is connected to some common-neighbor item vertices with the desired (query) keywords. While Eq. (3) computes the user relationship score $score_{u_i,u_j}(G_t)$, by considering all possible common-neighbor item vertices $v \in L(G_t)$ of users u_i and u_j, and ignoring the constraint of query keywords (i.e., Q), the score $score_{u_i,u_j}(G_t)$ is essentially an upper bound of $score_{u_i,u_j}(g)$ (since $g \subseteq G$ holds). Therefore, we can use $score_{u_i,u_j}(G_t)$ as the score upper bound $ub\_score_{u_i,u_j}(g)$ (i.e., $ub\_score_{u_i,u_j}(g) = score_{u_i,u_j}(G_t)$).

Computation of the Score $score_{u_i,u_j}(G_t)$: Now the only remaining issue is how to efficiently compute the user relationship score, $score_{u_i,u_j}(G_t)$, which is given by the summed score for all butterflies containing u_i and u_j by combining Eqs. (3) and (2).

Based on the *multinomial theorem* [5] with power $n = 2$, we have:

$$\left(\sum_{i=1}^{m} o_i\right)^2 = \sum_{i=1}^{m} {o_i}^2 + 2 \sum_{1 \leq i < j \leq m} o_i o_j,$$

which can be rewritten as:

$$\sum_{1 \leq i < j \leq m} o_i o_j = \frac{1}{2}\left(\left(\sum_{i=1}^{m} o_i\right)^2 - \sum_{i=1}^{m} {o_i}^2\right). \tag{5}$$

Based on Eq. (5), the user relationship score, $score_{u_i,u_j}(G_t)$, between u_i and u_j (as given in Eq. (3)) can be rewritten as:

$$\begin{aligned} score_{u_i,u_j}(G_t) &= \sum_{\forall a<b, v_a, v_b \in L(G_t)} w_{\bowtie(u_i,u_j,v_a,v_b)} \qquad (6) \\ &= \sum_{\forall a<b, v_a, v_b \in L(G_t)} w_{\angle(u_i,v_a,u_j)} \cdot w_{\angle(u_i,v_b,u_j)} \\ &= \frac{1}{2}\left(\left(\sum_{\forall v \in L(G_t)} w_{\angle(u_i,v,u_j)}\right)^2 - \sum_{\forall v \in L(G_t)} \left(w_{\angle(u_i,v,u_j)}\right)^2\right) \\ &= \frac{1}{2}\left(\left(\sum_{\forall v \in L(G_t)} \min\{w_{u_i,v}, w_{u_j,v}\}\right)^2 - \sum_{\forall v \in L(G_t)} \left(\min\{w_{u_i,v}, w_{u_j,v}\}\right)^2\right) \\ &= \frac{1}{2}\left(X^2_{u_i,u_j} - Y_{u_i,u_j}\right). \end{aligned}$$

From Eq. (6), in order to calculate the user relationship score $score_{u_i,u_j}(G_t)$ for each user pair u_i and u_j, we only need to maintain two terms, $X_{u_i,u_j} = \sum_{\forall v \in L(G_t)} \min\{w_{u_i,v}, w_{u_j,v}\}$ and $Y_{u_i,u_j} = \sum_{\forall v \in L(G_t)} \left(\min\{w_{u_i,v}, w_{u_j,v}\}\right)^2$. Upon graph updates in stream S, we can incrementally and efficiently update X_{u_i,u_j} and Y_{u_i,u_j} online (as will be discussed later in Section 5.2).

5 AUXILIARY DATA INITIALIZATION AND INCREMENTAL MAINTENANCE

This section details the initialization stage (lines 1-4 of Algorithm 1) that computes auxiliary data over an initial bipartite graph G_0 in Section 5.1, and constructs a synopsis, *Syn*, for such auxiliary data in Section 5.3, as well as incremental maintenance of auxiliary data and synopsis (lines 5-8 of Algorithm 1) in Sections 5.2 and 5.4, respectively.

5.1 Auxiliary Data Initialization

To facilitate efficient online community detection, we will compute some auxiliary data over the initial graph G_0 in the initialization phase, which can enable our proposed pruning strategies (as mentioned in Section 4).

- **(Item Keyword Bit Vector)** a keyword bit vector, $v.BV$, of size B for each item vertex $v \in L(G_0)$, whose elements $v.BV[i]$ are 1 if a keyword in $v.K$ is hashed to the i-th position (otherwise, $v.BV[i] = 0$);
- **(User Keyword Bit Vector)** a keyword bit vector, $u.BV$, of each user vertex u, which is a bit-OR of bit vectors $v.BV$ for all u's neighbor item vertices $v \in N(u)$ (i.e., $u.BV = \bigvee_{\forall v \in N(u))} v.BV$);
- **(Support Upper Bound)** an upper bound, $ub\_sup(e_{u,v})$ (as given by Eq. (4)), of the support $sup(e_{u,v})$, for each edge $e_{u,v} \in E(G_0)$, and;
- **(Aggregated Wedge Scores)** the summed wedge score, $X_{u_i,u_j} = \sum_{\forall v \in L(G_t)} w_{\angle(u_i,v,u_j)}$, and the summation over the squares of wedge scores, $Y_{u_i,u_j} = \sum_{\forall v \in L(G_t)} (w_{\angle(u_i,v,u_j)})^2$, for each user pair $(u_i, u_j) \in U(G_0) \times U(G_0)$ and item vertices $v \in N(u_i) \cap N(u_j)$.

5.2 Incremental Maintenance of Auxiliary Data

In this subsection, we illustrate how to incrementally maintain auxiliary data of graph G_t for each user vertex, upon changes of the update items in the sliding window W_t. As mentioned in Section 2.1, the sliding window W_t at a new timestamp t (from W_{t-1}) needs to insert a new item p_t and remove an expired old item p_{t-s}. Upon such updates, we need to incrementally compute the edge weights in the graph G_t, as well as the auxiliary data in the synopsis *Syn*.

Maintenance of Keyword Bit Vectors: When an edge e_{u_i,v_a} is inserted or deleted in the bipartite graph G_t, we need to update the keyword bit vector $u_i.BV$. Specifically, for the insertion of a new edge e_{u_i,v_a}, the new keyword bit vector is given by $u_i.BV \bigvee v_a.BV$; for the deletion of an expired edge e_{u_i,v_a}, we recompute the keyword bit vector $u_i.BV = \bigvee_{\forall v \in N(u_i)} v.BV$.

Maintenance of Support Upper Bounds: At timestamp t, due to the insertion of p_t or the deletion of p_{t-s}, an edge e_{u_i,v_a} can be added or deleted, respectively. Consequently, we need to update the support upper bounds of the edges in all butterflies $\bowtie (u_i, u_j, v_a, v_b)$ containing the edge e_{u_i,v_a}. When inserting a new edge e_{u_i,v_a}, for each edge e in $\bowtie (u_i, u_j, v_a, v_b)$, we increase the support upper bound $ub\_sup(e)$ by 1. Similarly, when deleting an edge e_{u_i,v_a}, for each edge e in $\bowtie (u_i, u_j, v_a, v_b)$, we decrease the support upper bound $ub\_sup(e)$ by 1.

Maintenance of User Relationship Scores: Since the user relationship score $score_{u_i,u_j}(G_t)$ is calculated using X_{u_i,u_j} and Y_{u_i,u_j} (as given by Eq. (6)), we maintain these two terms incrementally. Specifically, we compute the differences, $\Delta X_{u_i,u_j}$ and $\Delta Y_{u_i,u_j}$, of X_{u_i,u_j} and Y_{u_i,u_j}, respectively, between timestamps $(t-1)$ and t. Upon updating the edge e_{u_i,v_a} at timestamp t, the edge weight w_{u_i,v_a} has an increase/decrease of 1. Accordingly, the wedge weight $w_{\angle(u_i,v_a,u_j)}$ $(= min\{w_{u_i,v_a}, w_{u_j,v_a}\})$ is updated with $w_{\angle(u_i,v_a,u_j)} + \lambda_{\angle(u_i,v_a,u_j)}$, where $\lambda_{\angle(u_i,v_a,u_j)} \in \{-1, 0, 1\}$ (depending on the change of $min\{w_{u_i,v_a}, w_{u_j,v_a}\}$).

Here, there are three possible cases for the $\lambda_{\angle(u_i,v_a,u_j)}$ value:

- **(Case 1)** if the weight w_{u_i,v_a} of edge e_{u_i,v_a} is increased by 1 and $w_{u_i,v_a} + 1 \leq w_{u_j,v_a}$, then we have $\lambda_{\angle(u_i,v_a,u_j)} = 1$;
- **(Case 2)** if the weight w_{u_i,v_a} of edge e_{u_i,v_a} is decreased by 1 and $w_{u_i,v_a} - 1 < w_{u_j,v_a}$, then we have $\lambda_{\angle(u_i,v_a,u_j)} = -1$;
- **(Case 3)** for the remaining case, we have $\lambda_{\angle(u_i,v_a,u_j)} = 0$.

The Computation of Terms $\Delta X_{u_i,u_j}$ and $\Delta Y_{u_i,u_j}$: We provide the formulae of the two terms $\Delta X_{u_i,u_j}$ and $\Delta Y_{u_i,u_j}$ below.

$$\begin{aligned}
&\Delta X_{u_i,u_j} &(7)\\
&= X^{(t)}_{u_i,u_j} - X^{(t-1)}_{u_i,u_j}\\
&= \left(\sum_{\forall v \in L(G), v \neq v_a} w_{\angle(u_i,v,u_j)} + (w_{\angle(u_i,v_a,u_j)} + \lambda_{\angle(u_i,v_a,u_j)}) \right) - \sum_{\forall v \in L(G)} w_{\angle(u_i,v,u_j)}\\
&= \lambda_{\angle(u_i,v_a,u_j)},
\end{aligned}$$

where $X^{(t)}_{u_i,u_j}$ and $X^{(t-1)}_{u_i,u_j}$ are the wedge-score-related data X_{u_i,u_j} at timestamps t and $(t-1)$, respectively.

From Eq. (7), for any user pair (u_i, u_j), the increase/decrease of X_{u_i,u_j} is equal to that of $w_{\angle(u_i,v_a,u_j)}$ (i.e., $\lambda_{\angle(u_i,v_a,u_j)}$).

$$\begin{aligned}
&\Delta Y_{u_i,u_j} &(8)\\
&= Y^{(t)}_{u_i,u_j} - Y^{(t-1)}_{u_i,u_j}\\
&= \left(\sum_{\forall v \in L(G), v \neq v_a} \left(w_{\angle(u_i,v,u_j)}\right)^2 + (w_{\angle(u_i,v_a,u_j)} + \lambda_{\angle(u_i,v_a,u_j)})^2 \right)\\
&\quad - \sum_{\forall v \in L(G)} \left(w_{\angle(u_i,v,u_j)}\right)^2\\
&= 2\lambda_{\angle(u_i,v_a,u_j)} w_{\angle(u_i,v_a,u_j)} + \lambda^2_{\angle(u_i,v_a,u_j)},
\end{aligned}$$

where $Y^{(t)}_{u_i,u_j}$ and $Y^{(t-1)}_{u_i,u_j}$ are the wedge-score-related data Y_{u_i,u_j} at timestamps t and $(t-1)$, respectively.

As given in Eq. (8), the increment/decrement of Y_{u_i,u_j} can be computed via $w_{\angle(u_i,v_a,u_j)}$ and $\lambda_{\angle(u_i,v_a,u_j)}$.

Time Complexity Analysis: When a new edge e_{u_i,v_a} is inserted, the updated keyword bit vector is given by $u_i.BV \bigvee v_a.BV$, which takes $O(1)$ cost; when an edge e_{u_i,v_a} is deleted, we can re-compute the keyword bit vector $u_i.BV = \bigvee_{\forall v \in N(u_i)} v.BV$, which has the time complexity $O(|N(u_i)|)$.

For an updated edge e_{u_i,v_a}, we update the support upper bound of edges connected to v_a and v_b, where $v_b \in N(u_i) \cap N(u_j) - \{v_a\}$, $u_j \in N(v_a)$. Thus, the worst-case time cost is $O(|N(v_a)| \cdot |N(u_i)|)$.

Upon the update of edge e_{u_i,v_a}, for any user pair (u_i, u_j), the time complexity of updating X_{u_i,u_j} and Y_{u_i,u_j} is given by $O(1)$. The worst-case time complexity is given by $O(|N(v_a)|)$, where $N(v_a)$ is a set of user neighbors u_j of the item vertex v_a.

Therefore, the total time complexity of the graph increment maintenance is given by $O(|N(v_a)| \cdot |N(u_i)|)$.

5.3 The Synopsis Construction

In this subsection, we present a hierarchical tree synopsis, *Syn*, over auxiliary data for the dynamic graph G_t (i.e., initial graph G_0 or its subsequent version at timestamp t) to accelerate online CD-SBN query processing.

The Data Structure of Synopsis, ***Syn*****:** We design a hierarchical synopsis, *Syn*, over the initial bipartite graph G_0, where each synopsis node, M, has multiple entries M_i, and each of them corresponds to a partition of G_0. In detail, the hierarchical synopsis consists of two types of nodes: leaf and non-leaf nodes.

Leaf Nodes: Each leaf node M contains entries, each in the form of a user vertex $u \in U(G_t)$, associated with auxiliary data, $u.agg$. For each possible radius $r \in [1, r_{max}]$, we calculate and store aggregates in the form $(u.BV_r, u.ub\_sup_r, u.ub\_score_r)$ as follows:

- a keyword bit vector, $u.BV_r$, of user vertex u, which is a bit-OR of bit vectors $u_i.BV$ for all user vertices u_i in $(2r)$-hop subgraph of u (i.e., $u.BV_r = \bigvee_{\forall u_i \in hop(u,2r))} u_i.BV$);
- a support upper bound, $u.ub\_sup_r$ of support upper bounds for edges in the $(2r)$-hop subgraph of u (i.e.,$u.ub\_sup_r = max_{\forall e \in hop(u,2r)} ub\_sup(e)$), and;
- a user relationship score upper bound, $u.ub\_score_r$, of scores, $score_{u_i,u_j}(G_t)$, for all user pairs (u_i, u_j) within the $(2r)$-hop subgraph of user u (i.e., $u.ub\_score_r = max_{\forall u_i,u_j \in hop(u,2r)}$ $score_{u_i,u_j}(G_t)$).

Non-Leaf Nodes: Each non-leaf node M in the synopsis Syn contains entries M_i, each associated with the following aggregated data $M_i.agg$ (w.r.t. each possible radius $r \in [1, r_{max}]$).

- an aggregated keyword bit vector $M_i.BV_r = \bigvee_{\forall u \in M_i} u.BV_r$;
- the maximum support upper bound $M_i.ub\_sup_r = max_{\forall u \in M_i} u.ub\_sup_r$, and;
- the maximum score upper bound $M_i.ub\_score_r = max_{\forall u \in M_i} u.ub\_score_r$.

Bottom-up Construction of the Synopsis Syn: We construct the hierarchical synopsis, Syn, as follows. For the initial bipartite graph G_0 (i.e., timestamp $t = 0$), we start from each vertex $u_i \in U(G_0)$ and perform a breadth-first search (BFS) to extract a $(2r)$-hop subgraph, $hop(u_i, 2r)$, centered at u_i with radius $2r$ (for $r \in [1, r_{max}]$). Next, for each subgraph $hop(u_i, 2r)$, we compute its auxiliary data, $u_i.agg$. We sort the vertices in G_0 according to their summed score upper bounds for all possible radii (i.e., $\sum_{\forall r \in [1, r_{max}]} u.ub\_score_r$), and divide them into multiple partitions of the same size γ (i.e., vertices with similar score upper bounds are grouped together), which are leaf nodes of the synopsis Syn (as discussed in Section 5.3). Then, in a bottom-up manner, we recursively obtain non-leaf nodes M of the synopsis Syn, by grouping leaf or non-leaf nodes M_i (of the same size) on the lower level. The score upper bound associated with node M is computed by aggregating the score upper bounds of its child nodes (i.e., $\sum_{\forall r \in [1, r_{max}]} M_i.ub\_score_r$). The recursive synopsis construction terminates until one root node, $root(Syn)$, is obtained. After building the tree structure of the synopsis, we will recursively calculate aggregates, $M_i.agg$, of entries M_i in leaf/non-leaf nodes of synopsis Syn. In addition, we keep an inverted list, inv_list, for user vertices $u_i \in U(G_t)$, where each user vertex u_i is associated with an ordered list of node IDs from root to a leaf node in synopsis Syn in which u_i resides. This inverted list, inv_list, can be used for accessing a particular user vertex for the synopsis maintenance (as discussed in Section 5.4).

5.4 Incremental Maintenance of the Synopsis

In this subsection, we discuss how to identify the affected vertices and nodes in the synopsis Syn and update their aggregates in Syn.
Identification of the Affected Vertices and Synopsis Nodes: When an item from S arrives or expires in W_t (i.e., an edge e_{u_i,v_a} changes its weight w_{u_i,v_a} or is inserted/removed into/from graph G_{t-1}), the auxiliary data, $u.agg$, of some user vertices need to be updated. Therefore, our first goal is to identify those potentially affected vertices (with aggregate updates), which are user vertices in the $(2r)$-hop subgraph of u_i for all possible radii $r \in [1, r_{max}]$.

Specifically, if edge e_{u_i,v_a} is inserted or deleted, then we consider those users $u \in hop(u_i, 2r)$ within $(2r)$ hops away from u_i as the affected vertices, which need to update keyword bit vectors BV_r. Moreover, since the support of each edge e in $\bowtie (u_i, u_j, v_a, v_b)$ may increase/decrease, user vertices, u, within $(2r)$-hop from u_i may be affected (as $(2r)$-hop subgraph of u may contain those edges e). Furthermore, the user relationship score between u_i and each vertex $u_j \in N(v_a)$ may change. Thus, we need to update the scores for those user vertices u_j whose $(2r)$-hop subgraphs contain both u_i and u_j.

Finally, in synopsis Syn, we identify those (non-)leaf nodes that contain the affected user vertices, whose aggregated data will be updated.

Maintenance of Leaf and Non-leaf Nodes: After identifying the affected vertices/nodes, we update their aggregates for all possible radii $r \in [1, r_{max}]$. Assume that an edge e_{u_i,v_e} is inserted or expired, and u is an affected vertex. For the edge insertion, we merge $u_i.BV$ into $u.BV_r$, update $u.ub\_sup_r$ and $u.ub\_score_r$ for each affected vertex u, and update aggregates of the affected synopsis nodes M containing u (via inv_list). For the edge expiration, we re-aggregate the data, $u.BV_r$, $u.ub\_sup_r$, and $u.ub\_score_r$, from $(2r)$-hop of each affected vertex u, and we re-aggregate the data in the affected nodes containing u (if the re-aggregated data on the lower-level affected nodes have changed).
Maintenance of the Newly Added User Vertices: When an edge is inserted with a new user vertex u_{new}, we first compute the aggregates, $u_{new}.agg$ for this new user u_{new} (within $(2r)$ hops from u_{new}) and add vertex u_{new} to a leaf node M_i in synopsis Syn (containing a user vertex with the closest score upper bound). Next, we update the aggregate data of non-leaf nodes containing leaf node M_i.
Time Complexity Analysis: Assume that an edge e_{u_i,v_a} is inserted or expired. For the identification of affected vertices, it takes $O(|U(hop(u_i, 2r)|)$ to obtain vertices that may need to update BV_r, $u.ub\_sup_r$, and $u.ub\_score_r$. After that, we check inv_list to retrieve the affected nodes containing the affected vertices with the time complexity of $O(1)$. When edge e_{u_i,v_a} is inserted, it takes $O(1)$ to merge $u_i.BV$ into $u.BV_r$, maintain support upper bound $u.ub\_sup_r$ and score upper bound $u.ub\_score_r$ for each affected vertex u. Then, we update aggregates in leaf and non-leaf nodes containing u with $O(1)$ cost. For the edge expiration, it takes $O(|U(hop(u, 2r))|)$ time complexity to aggregate the bit vector of user keywords, calculate support upper bound $u.ub\_sup_r$, and update score upper bound $u.ub\_score_r$ according to the $(2r)$-hop subgraph of each affected vertex u. To maintain non-leaf nodes, we recompute the aggregates for each non-leaf node, which takes $O(\gamma)$, where γ is the average size of synopsis nodes. In the worst case, we can find $|U(hop(u_i, 2r))|$ affected vertices and h synopsis nodes for each vertex, where h is the height of the tree synopsis Syn. In summary, it takes $O(|U(hop(u_i, 2r))| + h)$ for each edge insertion and $O(|U(hop(u_i, 2r))| \cdot |U(hop(u, 2r))| + h \cdot \gamma)$ for each edge expiration.

6 CD-SBN QUERY PROCESSING

This section presents pruning strategies over synopsis Syn to filter out unqualified synopsis nodes and provide algorithms for snapshot and continuous CD-SBN processing by accessing the synopsis.

6.1 Pruning via Synopsis

Keyword Pruning for Synopsis Entries: We first discuss how to use the aggregated keyword bit vector, $M_i.BV$, to rule out a synopsis entry M_i, if no query keyword in Q exists in the vertices under M_i.

Lemma 6. **(Synopsis Keyword Pruning)** *Given a synopsis entry M_i and a set, Q, of query keywords, entry M_i can be safely pruned, if it holds $M_i.BV \bigwedge Q.BV = \mathbf{0}$, where $Q.BV$ is a bit vector hashed from query keywords in Q.*

Proof. Since $M_i.BV$ is an aggregated keyword bit vector computed by $\bigvee_{\forall u \in M_i} u.BV$, the $f(w)$-th bit position of $M_i.BV$ with the value 0 means that the keyword w is not contained in the keyword set, $v.K$, of any item vertex v, where $v \in N(u)$ and the $f(\cdot)$ is a hash function. Therefore, if $M_i.BV \wedge Q.BV = \mathbf{0}$ holds, no query keywords in Q belong to M_i, which violates the keyword constraint in Definition 8 (i.e., $v.K \cap Q \neq \emptyset$). In this case, the synopsis entry M_i cannot contain any vertices in the CD-SBN query answer and thus can be safely pruned, which completes the proof. □

Support Pruning for Synopsis Entries: Next, we utilize the maximum support upper bound, $M_i.ub\_sup_r$, and the query support threshold k to filter out a synopsis entry M_i.

Lemma 7. **(Synopsis Support Pruning)** *Given a synopsis entry M_i and a support threshold k, entry M_i can be safely pruned, if it holds that $M_i.ub\_sup_r < k$, where $M_i.ub\_sup_r$ is the maximum edge support upper bound in all $(2r)$-hop subgraphs of vertices under M_i.*

Proof. Since $M_i.ub\_sup_r$ is the maximum edge support upper bound for all vertices $u \in M_i$, it holds that $M_i.ub\_sup_r \geq u.ub\_sup_r$. Moreover, $u.ub\_sup_r$ is greater than the support of any edge in the $2r$-hop subgraph of u. Therefore, if $M_i.ub\_sup_r < k$ holds, the support of any edge connected to $u \in M_i$ is less than k (for any user vertex $u \in M_i$), which indicates that M_i can be safely pruned. □

Score Upper Bound Pruning for Synopsis Entries: Finally, we exploit the maximum upper bound, $M_i.ub\_score_r$, of user relationship scores and the score threshold σ to discard synopsis entries.

Lemma 8. **(Synopsis Score Upper Bound Pruning)** *Given a synopsis entry M_i and a user relationship score threshold σ, entry M_i can be safely pruned, if it holds that $M_i.ub\_score_r < \sigma$.*

Proof. Since $M_i.ub\_score_r$ is the maximum score upper bound, it holds that $M_i.ub\_score_r \geq u.ub\_score_r$, where $u.ub\_score_r$ is the user relationship score upper bound of any user vertex $u \in M_i$. Moreover, it holds that $u.ub\_score_r \geq score_{u,u_l}(G_t)$, where $score_{u,u_l}(G_t)$ is the user relationship score between u and its 2-hop reachable user vertex u_l. By the inequality transition, we have $M_i.ub\_score_r \geq u.ub\_score_r \geq score_{u,u_l}(G_t)$. If $M_i.ub\_score_r < \sigma$ holds, then we have $score_{u,u_l}(G_t) < \sigma$ for all u under M_i, which violates the score constraint. Therefore, in this case, we can safely prune entry M_i, which completes the proof. □

6.2 Snapshot CD-SBN Processing Algorithm

Algorithm 2 details our snapshot CD-SBN processing algorithm to obtain initial CD-SBN community answers, which consists of three major parts: i) the initialization of data structures and variables for preparing the synopsis traversal (lines 1-4); ii) the traversal of the synopsis, *Syn*, to obtain a candidate set P (lines 5-18); and iii) the refinement of a snapshot CD-SBN result set R (line 19).

Initialization: Given a query keyword set Q, the algorithm first hashes all query keywords in Q into a query keyword bit vector $Q.BV$ (line 1). Then, we initialize a *maximum heap* $\mathcal{H}$ (for synopsis traversal), accepting heap entries in the form of (M, key), where M is a synopsis entry and *key* is the maximum score upper bound, $M.ub\_score_r$, of all user vertices in M (as mentioned in Section 5.3) (line 2). Intuitively, a heap entry with a large key (i.e., large maximum score upper bound) is more likely to contain communities with high user relationship scores. Next, we insert the root of the tree synopsis *Syn*, in the form $(root(Syn), +\infty)$, into the heap $\mathcal{H}$ (line 3). Moreover, we use an initial empty set P to keep the candidate subgraphs and an empty set R to store snapshot CD-SBN query answers (line 4).

Synopsis Traversal: Next, we utilize the maximum heap $\mathcal{H}$ to traverse the synopsis (lines 5-18). Each time we pop out a heap

Algorithm 2: Snapshot CD-SBN Processing

Input: i) a streaming bipartite network G_t; ii) a tree synopsis Syn over G_t, and; iii) a snapshot CD-SBN query with a set, Q, of query keywords, a query support threshold k, a query maximum radius r, and a threshold, σ, of user relationship score

Output: a set, R, of CD-SBN communities

// **Initialization**

1 hash all query keywords in Q into a bit vector $Q.BV$;
2 initialize a maximum heap $\mathcal{H}$ in the form of (M, key);
3 insert $(root(Syn), +\infty)$ into heap $\mathcal{H}$;
4 $P = \emptyset$; $R = \emptyset$;

// **Synopsis Traversal**

5 **while** *$\mathcal{H}$ is not empty* **do**
6   (M, key) = de-heap($\mathcal{H}$);
7   **if** $key < \sigma$ **then** // Lemma 8
8   terminate the loop;
9   **if** *M is a leaf node* **then**
10   **for** *each vertex $u_i \in M$* **do**
11   **if** *$2r$-hop subgraph $hop(u_i, 2r)$ cannot be pruned by Lemma 1, 2, 3, or 5* **then**
12   obtain a maximal (k, r, σ)-bitruss $g \subseteq hop(u_i, 2r)$ satisfying the keyword constraint;
13   **if** *g exists* **then**
14   add g to P;
15   **else** // M is a non-leaf node
16   **for** *each entry $M_i \in M$* **do**
17   **if** *M_i cannot be pruned by Lemma 6, 7, or 8* **then**
18   insert entry $(M_i, M_i.ub\_score_r)$ into heap $\mathcal{H}$;

// **Refinement**

19 refine candidate communities in P (removing the redundancy) and obtain the CD-SBN result set R;
20 **return** R;

entry (M, key) from heap $\mathcal{H}$ with the highest key, *key* (i.e., the maximum score upper bound) (lines 5-6). If *key* is less than the score threshold, σ, it indicates that all the remaining heap entries in $\mathcal{H}$ have a score less than σ, and they cannot be contained in our CD-SBN query results. Thus, we can safely terminate the synopsis traversal early (lines 7-8).

When M is a leaf node, we consider each user vertex $u_i \in M$ and check whether or not we can find any candidate community centered on u_i (lines 9-14). For each user $u_i \in M$, if the $2r$-hop subgraph $hop(u_i, 2r)$ cannot be pruned by our pruning strategies (i.e., *keyword*, *support*, and *score upper bound pruning*), then we will obtain a maximal (k, r, σ)-bitruss g (satisfying the keyword constraint) within $hop(u_i, 2r)$ (lines 10-12). If such a subgraph g exists, we will add g to the candidate community set P (lines 13-14).

On the other hand, when M is a non-leaf node, we will check each entry M_i of node M (lines 15-16). If synopsis entries M_i cannot be pruned using synopsis-level pruning strategies, then we will insert $(M_i, M_i.ub\_score_r)$ into the heap $\mathcal{H}$ for further checking (lines 17-18).

Refinement: Finally, we remove duplicate subgraphs from P (due to overlap of $2r$-hop subgraphs), refine candidate communities in

P to obtain actual CD-SBN community answers in R, and return R (lines 19-20).

Complexity Analysis: Let $PP^{(j)}$ be the pruning power (i.e., the percentage of node entries that can be pruned) at the j-th level of synopsis Syn, where $0 \leq j \leq h$ (here h is the height of the tree synopsis Syn). Denote f as the average fanout of the synopsis nodes in $\mathcal{I}$. For the index traversal, the number of nodes that need to be accessed is given by $\sum_{j=1}^{h} f^{h-j+1} \cdot (1 - PP^{(j)})$. Next, to obtain g satisfying the constraints, we need to prune the item vertices without any query keyword. Therefore, it is necessary to recompute the support of the relevant edges and the user relationship score of any pair of adjacent user vertices. Let $\overline{n}$ be the number of pruned item vertices and deg_{avg} be the average degree of the item vertices. The time cost of re-computation is $\overline{n} \cdot deg_{avg}^2$ (given in Section 5.2 and Section 5.4). Then, we can obtain the maximal (k, r, σ)-bitruss in $O(deg_{avg}^2)$ with computed edge support, which is demonstrated in [26]. Thus, it takes $O(f^{h+1} \cdot (1 - PP^{(0)}) \cdot \overline{n} \cdot deg_{avg}^2)$ to refine candidate seed communities, where $PP^{(0)}$ is the pruning power over r-hop subgraphs in leaf nodes. Therefore, the total time complexity of Algorithm 2 is given by $O\left(\sum_{j=1}^{h} f^{h-j+1} \cdot (1 - PP^{(j)}) + f^{h+1} \cdot (1 - PP^{(0)}) \cdot \overline{n} \cdot deg_{avg}^2\right)$.

6.3 Continuous CD-SBN Processing Algorithm

Algorithm 3 details our continuous CD-SBN query processing approach to monitoring answers. First, we initialize the CD-SBN result set R_t at timestamp t with that R_{t-1} at timestamp $(t-1)$ (line 1). Next, we check the constraints of the existing communities to refine the previous result set (lines 2-6). Then, we obtain the potential communities from the $2r$-hop of user vertices near the insertion edge and remove the duplication (lines 7-10) Finally, we merge the candidate communities into the result set and return the result set at timestamp t (lines 11-12).

Edge Expiration Processing: To deal with the expiration of an item p_{t-s} (= $(e_{u_i,v_a}, t-s)$) in the sliding window W_{t-1}, the weight w_{u_i,v_a} of the edge e_{u_i,v_a} decreases (which might turn to the edge deletion). In this case, we will identify those community answers g in the answer set R_t that contain edge e_{u_i,v_a} (lines 2-3), and update the community g with g' (upon weight/edge changes; line 4). If such a community g' does not exist (violating the CD-SBN predicates), we simply remove community answer g from R_t (lines 5-6).

Edge Insertion Processing: Next, for the insertion of a new item p_t (= $(e_{u_i',v_a'}, t)$), we aim to update R_t with new CD-SBN community answers (containing the new/updated edge $e_{u_i',v_a'}$) in a set ΔR (lines 7-11). In particular, for each user vertex u_j within the $2r$-hop away from u_i', we obtain a CD-SBN candidate community g and add it to ΔR (if g exists) (lines 8-9). Then, we will remove the redundancy of community candidates in ΔR (i.e., already in R_t) and refine the remaining candidate communities in ΔR (line 10). Finally, we add CD-SBN answers in ΔR to R_t, and return final CD-SBN answers in R_t (lines 11-12).

Complexity Analysis: Let deg_{avg} be the average degree of the item vertices. For edge insertion and expiration, the time cost of user relationship score re-computation is $O(deg_{avg})$ and of edge support update is $O(deg_{avg}^2)$. Since obtaining the maximal (k, r, σ)-bitruss with computed edge supports and user relationship scores [26] requires $O(deg_{avg}^2)$, it takes $O(|R_{t-1}|deg_{avg}^2)$ to maintain the result set at $t-1$ and $O(deg_{avg}^3)$ to detect potential communities. Therefore, the total time complexity of Algorithm 3 at each timestamp t is given by $O\left(|R_{t-1}|deg_{avg}^2 + deg_{avg}^3\right)$.

Algorithm 3: Continuous CD-SBN Processing

Input: i) a streaming bipartite network G_{t-1} with an update stream S; ii) a set, R_{t-1}, of community answers at timestamp $(t-1)$, and; iii) a continuous CD-SBN query with a set, Q, of query keywords, a support threshold k, a maximum radius r, and a threshold, σ, of user relationship score

Output: a set, R_t, of CD-SBN community answers at timestamp t

```
1  R_t = R_{t-1};
   // Processing the Expired Item p_{t-s} = (e_{u_i,v_a}, t - s) ∈ S
2  for each community answer g ∈ R_t do
3      if e_{u_i,v_a} ∈ E(g) then
4          update the CD-SBN community g with g', upon the weight
             update (or deletion) of edge e_{u_i,v_a};
5          if g' does not exist then
6              remove g from R_t;
   // Processing the Insertion of a New Item p_t = (e_{u'_i,v'_a}, t) ∈ S
7  ΔR = ∅;
8  for each user vertex u_j ∈ hop(u'_i, 2r) do
9      obtain a CD-SBN candidate community g from hop(u_j, 2r)
         (via pruning) and add it to ΔR;
10 remove duplicate communities g from ΔR (overlapping with R_t)
     and refine candidate communities g in ΔR;
11 R_t = R_t ∪ ΔR;
12 return R_t;
```

Table 2: Statistics of graph data sets.

Data Set	Type	$\|U\|$	$\|V\|$	$\|E\|$	$\|\Sigma\|$
AmazonW (AM)	Rating	26,112	800	111,265	N/A
BibSonomy (BS)	Publication	5,795	767,448	2,555,080	204,674
CiaoDVD (CM)	Rating	17,616	16,121	295,958	N/A
CiteULike (CU)	Publication	22,716	731,770	2,411,819	153,278
Movielens (ML)	Movie	4,010	7,602	95,580	16,529
Escorts (SX)	Rating	10,106	6,624	50,632	N/A
TripAdvisor (TA)	Rating	145,317	1,760	703,171	N/A
UCForum (UF)	Interaction	899	522	33720	N/A
ViSualizeUs (VU)	Picture	17,110	495,402	2,298,816	82,036
dBkU, dBkL, dBkP	Synthetic	25,000	25,000	152,175	N/A
dPkU, dPkL, dPkP	Synthetic	25,000	25,000	152,922	N/A

7 EXPERIMENTAL EVALUATION

7.1 Experimental Settings

We conduct a set of experiments to examine the performance of our snapshot and continuous CD-SBN query processing algorithms over different real-world bipartite graphs. Our source code is available on GitHub[1].

Bipartite Graph Data Sets: We evaluate our proposed CD-SBN algorithms on nine real-world and six synthetic graphs. In particular, as depicted in Table 2, we provide statistics of real-world bipartite graphs from the KONECT[2] project (i.e., Koblenz Network Collection), where $|\Sigma|$ refers to the domain size of keywords in item

[1]https://github.com/L1ANLab/CD-SBN
[2]http://konect.cc/

Table 3: Parameter settings.

Parameters	Values
sliding window size s	200, 300 **500**, 800, 1000
support, k, of bitruss structure	3, **4**, 5
radius r	1, **2**, 3
relationship score threshold σ	1, 2, **3**, 4, 5
size, $\|Q\|$, of query keyword set Q	2, 3, **5**, 8, 10
keyword domain size $\|\Sigma\|$	100, 200, **500**, 800, 1000
size, $\|v_i.K\|$, of keywords per item vertex	1, 2, **3**, 4, 5
the number, $\|U(G)\|$, of user vertices	10K, 15K, **25K**, 50K, 100K
the number, $\|L(G)\|$, of item vertices	10K, 15K, **25K**, 50K, 100K
the edge weight range $[min\_w, max\_w]$	**[1,2]**, [1,3], [1,4]

vertices of the original graphs ("N/A" for $|\Sigma|$ means that graph data have no keywords).

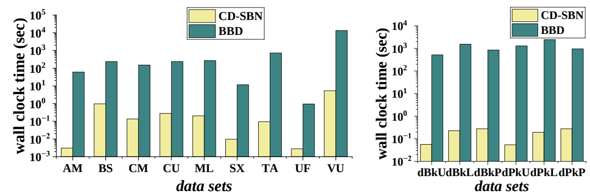
To generate synthetic bipartite graphs, we first randomly produce degrees, $deg(u)$, of user vertices u following *PowerLaw* or *Beta* distribution. Then, for each user vertex u, we connect it to $deg(u)$ random item vertices v. Next, to simulate the real-world frequency of user-item interaction (i.e., edge weights $w_{u,v}$), we assign each edge with weight ranging from $[min\_w, max\_w]$ following the *Gaussian* distribution, and associate them with edges, where the mean and standard deviation of the *Gaussian* distribution are $(1.5, 0.25)$, $(2, 0.5)$, and $(2.5, 0.75)$ for $[1, 2]$, $[1, 3]$, and $[1, 4]$, respectively. Finally, for each item vertex v, we generate its keyword set $v.K$, which contains integers following *Log-Normal*, *Pareto*, or *Uniform* distribution. For real-world graphs without keywords (e.g., AM, CM, SX, UF, and TA), we produce keyword sets of their item vertices following the *Log-Normal* distribution.

For CD-SBN predicates, we will randomly select $|Q|$ keywords from a keyword domain Σ, following the keyword distribution in the data graph G, and form a set, Q, of query keywords. Other parameter settings are provided in Table 3.

Competitors: Since no prior works studied the CD-SBN problem under the same semantics as (k, r, σ)-bitruss communities with specific keywords, we compare our CD-SBN algorithm with a baseline method, named *Bitrussness-Based Decomposition (BBD)*. Specifically, *BBD* first calculates the trussness of each edge, which is the maximum k of all the k-bitruss containing the edge and builds a synopsis based on the trussness. Then, upon edge updates at each timestamp, *BBD* filters out user vertices connected to edges with trussness less than k and computes the (k, r, σ)-bitruss community centered at each remaining vertex (containing query keywords).

Measures: We evaluate our CD-SBN performance, in terms of the *wall clock time*, which is the time cost to traverse the index and retrieve communities for snapshot CD-SBN, or that to maintain the CD-SBN results at each timestamp for continuous CD-SBN problem. The wall clock time is an average time cost over 10 runs with a set of 10 different CD-SBN predicates (e.g., query keywords).

Parameter Settings: Table 3 depicts the parameter settings of our subsequent experiments, where default values are in bold. Each time we vary one parameter while setting other parameters to their default values. Moreover, since our continuous CD-SBN processing algorithm has no data dependencies during the search of communities, we use multithreaded optimization to speed up the computation, where the number of threads is set to 20 by default. All the experiments are executed on a machine with an AMD Ryzen Threadripper 3990X 64-Core processor, Ubuntu 22.04 OS, 192GB



(a) real-world graphs (b) synthetic graphs

Figure 3: CD-SBN performance on data sets.

memory, and a 600GB disk. All algorithms were implemented in C++ (C++17 standard) and compiled with the g++ 11.4.0 compiler.

Research Questions: We design experiments for our CD-SBN algorithms to answer the following research questions (RQs):

RQ1 (Efficiency): Can our proposed algorithms efficiently process continuous CD-SBN problem?

RQ2 (Effectiveness): Can our proposed pruning strategies effectively filter out false alarms of candidate communities during the CD-SBN processing?

RQ3 (Utility): Do the resulting CD-SBN communities have practical significance for real-world applications?

7.2 CD-SBN Performance Evaluation

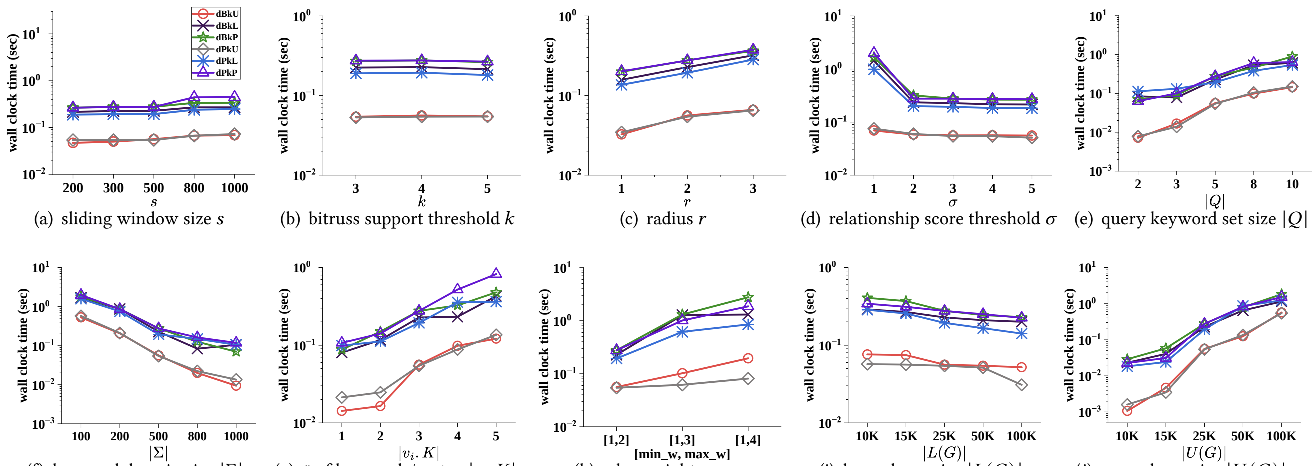
Continuous CD-SBN Efficiency (RQ1): Figure 3 compares the *wall clock time* of our continuous CD-SBN processing algorithm with that of *BBD* over real-world and synthetic bipartite graphs, where all parameters are set to default values in Table 3. The experiment results indicate that our continuous CD-SBN algorithm outperforms *BBD* by more than two orders of magnitude, confirming the efficiency of our CD-SBN approach on real/synthetic graphs.

We also test the robustness of our CD-SBN approach with different parameters (e.g., s, k, r, σ, and $|Q|$) on synthetic graphs with different structural parameters (e.g., $|\Sigma|$, $|v_i.K|$, $[min\_w, max\_w]$, $|L(G)|$, and $|U(G)|$).

Effect of Sliding Window Size s: Figure 4(a) presents the performance of our continuous CD-SBN approach, by varying the size s of sliding window W_t from 50 to 200, where other parameters are by default. From the experimental results, with the increasing s, more edges are involved or have higher edge weights, which result in more potential candidate communities. Therefore, larger window size s incurs higher time cost. Nonetheless, the time cost of our CD-SBN approach remains low (i.e.,0.047 ~ 0.726 *sec*) for different s values.

Effect of Bitruss Support Threshold k: Figure 4(b) illustrates the continuous CD-SBN performance, where the bitruss support parameter k varies from 3 to 5, and other parameters are set to default values. Since a larger support threshold k leads to higher pruning power, our CD-SBN approach can obtain fewer candidate communities, which in turn incurs lower time costs. The query cost remains low for different k values (i.e., 0.053 ~ 0.277 *sec*).

Effect of Radius r: Figure 4(c) shows the experimental results of our CD-SBN approach, by varying radius, r, the query community from 1 to 3, where default values are used for other parameters. When radius r becomes larger, the scale of $2r$-hop subgraphs will increase, which results in higher time costs to obtain maximal



(a) sliding window size s (b) bitruss support threshold k (c) radius r (d) relationship score threshold σ (e) query keyword set size $|Q|$

(f) keyword domain size $|\Sigma|$ (g) # of keywords/vertex $|v_i.K|$ (h) edge weight range (i) lower layer size $|L(G)|$ (j) upper layer size $|U(G)|$

Figure 4: Robustness evaluation of CD-SBN.

(k, r, σ)-bitruss. Nevertheless, for different r values, the *wall clock time* remains small (i.e., around 0.033 ∼ 0.375 *sec*).

Effect of Relationship Score Threshold σ: Figure 4(d) varies the relationship score threshold, σ, from 1 to 5, where other parameters are set by default values. Since higher σ leads to higher pruning power and cohesiveness of candidate communities, the number of candidate communities decreases for larger σ. Thus, the CD-SBN time cost decreases for larger σ values. Overall, for different σ values, the time cost remains low (i.e., 0.051∼1.989 *sec*).

Effect of Query Keyword Set Size $|Q|$: Figure 4(e) evaluates the CD-SBN performance, with different sizes, $|Q|$, of the query keyword set Q from 2 to 10, where we use default values for other parameters. Intuitively, a larger query keyword set Q allows more item vertices to be included (with lower pruning power by the keyword pruning), and in turn more candidate communities will be retrieved for filtering/refinement. Thus, from the figure, when $|Q|$ becomes larger, the query cost also increases. Nonetheless, the wall clock time remains low (i.e., 0.007 ∼ 0.893 *sec*) for different $|Q|$ values.

Effect of Keyword Domain Size $|\Sigma|$: Figure 4(f) shows the wall clock time of our CD-SBN approach, where the keyword domain size, $|\Sigma|$, varies from 100 to 1,000 and other parameters are set to their default values. In the figure, we can see that, when $|\Sigma|$ becomes larger, our CD-SBN approach requires less time to retrieve CD-SBN communities, which indicates the effectiveness of our keyword pruning for large keyword domain size. Overall, with different $|\Sigma|$ values, the time cost remains low (i.e., 1.974 ∼ 0.009 *sec*).

Effect of # of Keywords Per Item Vertex $|v_i.K|$: Figure 4(g) illustrates the CD-SBN performance for different numbers, $|v_i.K|$, of keywords per item vertex from 1 to 5, where we set other parameters to default values. Intuitively, more keywords in item vertices v_i will increase the chance to match with query keyword set Q, which results in more or larger candidate communities to filter and refine. Therefore, with the increase of $|v_i.K|$, the time cost of our CD-SBN approach also increases, but, nevertheless, remains low (i.e., 0.014∼0.821 *sec*).

Effect of Edge Weight Range $[min\_w, max\_w]$: Figure 4(h) examines the CD-SBN performance with different edge weight ranges, $[min\_w, max\_w]$ (i.e., $[1, 2]$, $[1, 3]$, and $[1, 4]$), where default values are used for other parameters. When the range of edge weights expands, more/larger candidate communities will be retrieved and refined. Thus, the query cost increases as the edge weight interval $[min\_w, max\_w]$ becomes wider. Nonetheless, the wall clock time remains low (i.e., 0.0543∼2.761 *sec*).

Effect of Lower Layer Size $|L(G)|$: Figure 4(i) tests the scalability of our proposed CD-SBN approach by varying the lower layer size, $|L(G)|$, from $10K$ to $100K$, where default values are used for other parameters. From the figure, we can see that the query cost decreases as $|L(G)|$ increases. This is because, with a constant average degree of each user vertex, larger $|L(G)|$ leads to a more sparse bipartite network, which incurs fewer candidate communities and lower query costs. The time cost remains low (i.e., 0.031∼0.407 *sec*).

Effect of Upper Layer Size $|U(G)|$: Figure 4(j) evaluates the scalability of our CD-SBN approach with different upper layer sizes, $|U(G)|$, from $10K$ to $100K$, where other parameters are set by default. In the figure, with the increase of $|U(G)|$, the CD-SBN time cost increases due to more retrieved candidate communities. Nonetheless, the time cost remains low (i.e., 0.001 ∼ 1.815 *sec*), which confirms the good scalability of our CD-SBN approach for large $|U(G)|$.

Ablation Study on the Snapshot CD-SBN Pruning Strategies (RQ2): To test the effectiveness of our proposed pruning methods, we conduct an ablation study of our snapshot CD-SBN pruning strategies over real-world/synthetic graphs, where all parameters are set to default values. We execute the snapshot CD-SBN processing algorithm by adding one pruning strategy at a time, which forms 3 experimental groups: (1) *support pruning* only, (2) *support + score pruning*, and (3) *support + score + keyword pruning*. Figure 5 reports the number of pruned candidate communities by using different pruning method groups. In this figure, we can see that for all data sets, the number of pruned communities increases by about an order of magnitude, as we apply more pruning strategies, which indicates the effectiveness of our proposed pruning strategies.

CD-SBN Community Case Study (RQ3): To evaluate the utility of our CD-SBN results, we carry out a case study to compare our proposed (k, r, σ)-bitruss pattern with the biclique [22] (which is a variant of bipartite community structure) over *BibSonomy* (BS)

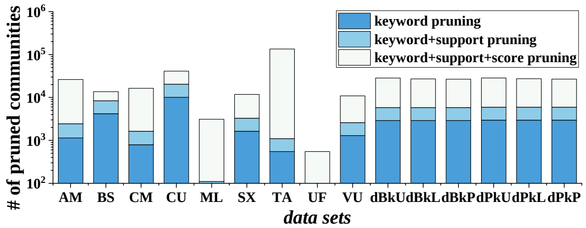


Figure 5: Ablation study of the pruning power for the snapshot CD-SBN problem.

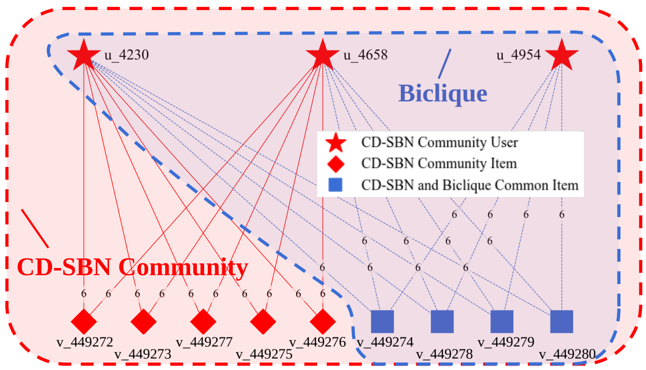


Figure 6: Case study of (k, r, σ)-bitruss vs. biclique [22].

graph data. The entire graph shown in Figure 6 is a $(4, 2, 3)$-bitruss containing 3 users (in blue) and 9 items (in blue or red), whereas the 3 users, 4 items (in blue) form a 3×4 biclique. We can see that the 3×4 biclique shares the same 3 users as that in $(4, 2, 3)$-bitruss, but the 3×4 biclique ignores some items that have edges connected to some users with large weights (e.g., 6). In contrast, our (k, r, σ)-bitruss captures these items, and can recommend them to users who are not connected to them (e.g., recommending items v_449272, v_449273, v_449277, v_449275 and 449276 to user u_4954). This confirms the utility of our proposed CD-SBN community semantics, which can recommend items to more potentially interested users.

8 RELATED WORK

Community Search/Detection Over Unipartite Graphs: As fundamental operations in social network analysis, the *community search* (CS) and *community detection* (CD) have been extensively studied for the past few decades [3, 8–10, 12, 17, 24, 28, 32, 40], which obtain communities that satisfy different query parameters (e.g., keyword relevance and structural cohesiveness). In unipartite graphs, such as social networks, different community semantics have been proposed, including minimum degree [10], k-core [28], k-clique [9], and (k, d)-truss [3, 17]. Previous works on community detection usually retrieved all communities considering link information only [12, 24]. Some recent works used clustering techniques to detect communities [8, 32, 40]. In contrast, our work is conducted on bipartite graphs with two distinct types of nodes (instead of unipartite graphs with the same type of nodes). Moreover, our proposed (k, r, σ)-bitruss in bipartite graphs considers not only the high connectivity of community structure but also edge weight and query keywords, which is more challenging. Thus, previous work on CS/CD over unipartite graphs cannot be used for streaming bipartite graphs (i.e., focus of our CD-SBN problem).

Dynamic Community Search/Detection Over Unipartite Graphs: In recent years, due to the importance of mining hidden trends or patterns in streaming data, community search/detection over large-scale dynamic graphs has become an increasingly important research problem. Some existing studies aim to find potential communities over dynamic social networks with different specified community structures to gain valuable insights, e.g., over unipartite dynamic graphs, [39] searches for and maintains a *Triangle-connected k-Truss Community* where edges should be contained in adjacent triangles and [30, 31] work on the k-core with edge weights higher than θ (i.e., (θ, k)-core). Unlike the existing tasks, our proposed CD-SBN problem focuses on detecting communities over streaming bipartite graphs, requiring more complicated calculations and greater computational overhead.

Community Search/Detection Over Bipartite Graphs: Several prior works on the CS/CD problem over bipartite graphs considered different community semantics, including (α, β)-core community [34], size-bounded (α, β)-community [43], personalized maximum biclique community [33], and k-wing community [1]. However, since these works are focused on searching potential communities over static bipartite graphs, we cannot directly apply previous techniques to solve our CD-SBN problem with dynamically changing bipartite graphs. This is because our proposed CD-SBN problem focuses on a different community semantic, i.e., (k, r, σ)-bitruss, which includes the constraints of structural cohesiveness, keyword matching, and user relationship scores. On the other hand, few prior works [2, 35] consider detecting or searching communities on dynamic streaming bipartite graphs. Although all of them construct an index that can be dynamically updated, for each query, these methods still need to re-traverse the entire index. In contrast, our CD-SBN approach only needs to incrementally maintain the result set upon edge updates, which significantly reduces the query cost.

9 CONCLUSIONS

In this paper, we propose a novel CD-SBN problem to detect those communities with user-specified query keywords and high structural cohesiveness over streaming bipartite graphs in snapshot and continuous scenarios. To efficiently tackle the CD-SBN problem, we provide effective pruning strategies to rule out false alarms of candidate communities and design a hierarchical synopsis to facilitate the CD-SBN processing. We also develop efficient algorithms to enable snapshot/continuous CD-SBN answer retrieval by quickly accessing the synopsis and applying our proposed pruning methods. Comprehensive experiments confirmed the effectiveness and efficiency of our proposed CD-SBN approaches over real/synthetic streaming bipartite networks upon graph updates.

REFERENCES

[1] Aman Abidi, Lu Chen, Rui Zhou, and Chengfei Liu. 2023. Searching Personalized k-wing in Bipartite Graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 35, 8 (2023), 8515–8528.

[2] Aman Abidi, Lu Chen, Rui Zhou, and Chengfei Liu. 2023. Searching Personalized Kk-Wing in Bipartite Graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 35, 8 (2023), 8515–8528.

[3] Ahmed Al-Baghdadi and Xiang Lian. 2020. Topic-based community search over spatial-social networks. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2104–2117.

[4] Michael J Barber. 2007. Modularity and community detection in bipartite networks. *Physical Review E* 76, 6 (2007), 066102.

[5] D. W. Bolton. 1968. The Multinomial Theorem. *The Mathematical Gazette* 52, 382 (1968), 336–342.

[6] Mohamed Bouguessa and Khaled Nouri. 2021. BiNeTClus: Bipartite network community detection based on transactional clustering. *ACM Transactions on Intelligent Systems and Technology (TIST)* 12, 1 (2021), 1–26.

[7] Hongxu Chen, Hongzhi Yin, Tong Chen, Weiqing Wang, Xue Li, and Xia Hu. 2022. Social Boosted Recommendation With Folded Bipartite Network Embedding. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 34, 2 (2022), 914–926.

[8] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. 2018. D2K: scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 1272–1281.

[9] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *Proceedings of International Conference on Management of Data (SIGMOD)*. 277–288.

[10] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of International Conference on Management of Data (SIGMOD)*. 991–1002.

[11] Lu Fan, Qimai Li, Bo Liu, Xiao-Ming Wu, Xiaotong Zhang, Fuyu Lv, Guli Lin, Sen Li, Taiwei Jin, and Keping Yang. 2022. Modeling user behavior with graph convolution for personalized product search. In *Proceedings of the ACM Web Conference (WWW)*. 203–212.

[12] Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3-5 (2010), 75–174.

[13] Xiangyang Gou and Lei Zou. 2023. Sliding Window-Based Approximate Triangle Counting with Bounded Memory Usage. *The VLDB Journal* 32, 5 (2023), 1087–1110.

[14] Inc. Groupon. 2024. GROUPON Offical Site. https://www.groupon.com/

[15] Jelena Grujić. 2008. Movies Recommendation Networks as Bipartite Graphs. In *Proceedigns of International Conference on Computational Science (ICCS)*. 576–583.

[16] Mengyue Hang, Ian Pytlarz, and Jennifer Neville. 2018. Exploring student check-in behavior for improved point-of-interest prediction. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 321–330.

[17] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[18] Yelp Inc. 2024. Yelp Open Dataset. https://www.yelp.com/dataset

[19] Junghoon Kim, Kaiyu Feng, Gao Cong, Diwen Zhu, Wenyuan Yu, and Chunyan Miao. 2022. ABC: attributed bipartite co-clustering. *Proceedings of the VLDB Endowment* 15, 10 (2022), 2134–2147.

[20] Ziyi Kou, Saurav Manchanda, Shih-Ting Lin, Min Xie, Haixun Wang, and Xiangliang Zhang. 2023. Modeling Sequential Collaborative User Behaviors For Seller-Aware Next Basket Recommendation. In *Proceedings of International Conference on Information and Knowledge Management (CIKM)*. 1097–1106.

[21] Chen Lang, Ze Wang, Kaiming He, and Shimin Sun. 2022. POI Recommendation Based on a Multiple Bipartite Graph Network Model. *The Journal of Supercomputing* 78, 7 (2022), 9782–9816.

[22] Bingkai Lin. 2018. The Parameterized Complexity of the k-Biclique Problem. *J. ACM* 65, 5, Article 34 (Aug. 2018), 23 pages. https://doi.org/10.1145/3212622

[23] Michal Ben Neria, Nancy-Sarah Yacovzada, and Irad Ben-Gal. 2017. A risk-scoring feedback model for webpages and web users based on browsing behavior. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 4 (2017), 1–21.

[24] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.

[25] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. 2018. Butterfly Counting in Bipartite Networks. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*. 2150–2159.

[26] Ahmet Erdem Sarıyüce and Ali Pinar. 2018. Peeling Bipartite Networks for Dense Subgraph Discovery. In *Proceedings of ACM International Conference on Web Search and Data Mining (WSDM)*. 504–512.

[27] Xihan Shan and Junlong Zhao. 2017. Adaptive Kendall's τ Correlation in Bipartite Network for Recommendation. In *Proceedings of International Conference on Intelligent Computing Methodologies (ICIC)*. 389–399.

[28] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 939–948.

[29] Yizhou Sun, Yintao Yu, and Jiawei Han. 2009. Ranking-based clustering of heterogeneous information networks with star network schema. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 797–806.

[30] Yifu Tang, Jianxin Li, Nur Al Hasan Haldar, Ziyu Guan, Jiajie Xu, and Chengfei Liu. 2024. Reliability-Driven Local Community Search in Dynamic Networks. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 36, 2 (2024), 809–822.

[31] Yifu Tang, Jianxin Li, Nur Al Hasan Haldar, Ziyu Guan, Jia-Jie Xu, and Chengfei Liu. 2022. Reliable Community Search in Dynamic Networks. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2826–2838.

[32] Nate Veldt, David F Gleich, and Anthony Wirth. 2018. A correlation clustering framework for community detection. In *Proceedings of the World Wide Web Conference (WWW)*. 439–448.

[33] Kai Wang, Wenjie Zhang, Xuemin Lin, Lu Qin, and Alexander Zhou. 2022. Efficient Personalized Maximum Biclique Search. In *Proceedings of International Conference on Data Engineering (ICDE)*. 498–511.

[34] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, Lu Qin, and Yuting Zhang. 2021. Efficient and Effective Community Search on Large-scale Bipartite Graphs. In *Proceedings of International Conference on Data Engineering (ICDE)*. 85–96.

[35] Kai Wang, Wenjie Zhang, Ying Zhang, Lu Qin, and Yuting Zhang. 2023. Discovering Significant Communities on Bipartite Graphs: An Index-Based Approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 35, 3 (2023), 2471–2485.

[36] Yan Wang, Zongxu Qin, Jun Pang, Yang Zhang, and Jin Xin. 2017. Semantic annotation for places in LBSN through graph embedding. In *Proceedings of ACM on Conference on Information and Knowledge Management (cIKM)*. 2343–2346.

[37] Tongfeng Weng, Xu Zhou, Kenli Li, Kian-Lee Tan, and Keqin Li. 2022. Distributed approaches to butterfly analysis on large dynamic bipartite graphs. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 34, 2 (2022), 431–445.

[38] Qingyu Xu, Feng Zhang, Zhiming Yao, Lv Lu, Xiaoyong Du, Dong Deng, and Bingsheng He. 2022. Efficient Load-Balanced Butterfly Counting on GPU. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2450–2462.

[39] Tianyang Xu, Zhao Lu, and Yuanyuan Zhu. 2022. Efficient triangle-connected truss community search in dynamic graphs. *Proceedings of the VLDB Endowment* 16, 3 (2022), 519–531.

[40] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A model-based approach to attributed graph clustering. In *Proceedings of International Conference on Management of Data (SIGMOD)*. 505–516.

[41] Tzu-Chi Yen and Daniel B Larremore. 2020. Community detection in bipartite networks with stochastic block models. *Physical Review E* 102, 3 (2020), 032309.

[42] Yiyeon Yoon, Juneseok Hong, and Wooju Kim. 2020. Item Recommendation by Predicting Bipartite Network Embedding of User Preference. *Expert Systems with Applications* 151 (2020), 113339.

[43] Yuting Zhang, Kai Wang, Wenjie Zhang, Wei Ni, and Xuemin Lin. 2024. Size-Bounded Community Search over Large Bipartite Graphs. In *Proceedings of International Conference on Extending Database Technology (EDBT)*. 320–331.

[44] Alexander Zhou, Yue Wang, and Lei Chen. 2021. Butterfly counting on uncertain bipartite graphs. *Proceedings of the VLDB Endowment* 15, 2 (2021), 211–223.

[45] Alexander Zhou, Yue Wang, and Lei Chen. 2021. Butterfly Counting on Uncertain Bipartite Graphs. *Proceedings of the VLDB Endowment* 15, 2 (2021), 211–223.

[46] Zhixin Zhou and Arash A. Amini. 2019. Analysis of spectral clustering algorithms for community detection: the general bipartite setting. *Journal of Machine Learning Research (JMLR)* 20, 47 (2019), 1–47.