Efficient Projection-Based Algorithms for Tip Decomposition on Dynamic Bipartite Graphs

Tongfeng Weng, Yumeng Liu, Mo Sha, Xinyuan Chen, Xu Zhou, Kenli Li, *Senior Member, IEEE*, and Kian-Lee Tan, *Senior Member, IEEE*

***Abstract*—This paper addresses the pressing need for effective k-tips decomposition in dynamic bipartite graphs, a crucial aspect of real-time applications that analyze and mine binary relationship patterns. Recognizing the dynamic nature of these graphs, our study is the first to provide a solution for k-tips decomposition in such evolving environments. We introduce a pioneering projection-based algorithm, coupled with advanced incremental maintenance strategies for edge modifications, tailored specifically for dynamic graphs. This novel approach not only fills a significant gap in the analysis of dynamic bipartite graphs but also substantially enhances the accuracy and timeliness of data-driven decisions in critical areas like public health. Our contributions set a new benchmark in the field, paving the way for more nuanced and responsive analyses in various domains reliant on dynamic data interpretation.**

***Index Terms*—Bipartite graph, community detection, dynamic graph, incremental algorithm, tip number.**

I. Introduction

BIPARTITE graphs represent a significant tool that is instrumental in modeling and analyzing 2-hop relationships [1], [2], [3]. They find extensive application in various domains, such as social network analysis [4], [5] (depicting individual-group relationships), recommendation systems [6], [7] (linking consumers to products), and bioinformatics [8] (mapping genes to traits), among others. The unique properties of these graphs make them an ideal solution for addressing numerous practical challenges. The study of bipartite graphs, though well-established and thoroughly explored, continues to be vibrant and critical. This enduring relevance is driven by the ongoing diversification and democratization of data collection sources, alongside the surge in demand for emerging applications. The recent focus is on joint analysis of multi-source data to mine simultaneous movement patterns or collaborative event participation, generating significant interest [9], [10], [11].

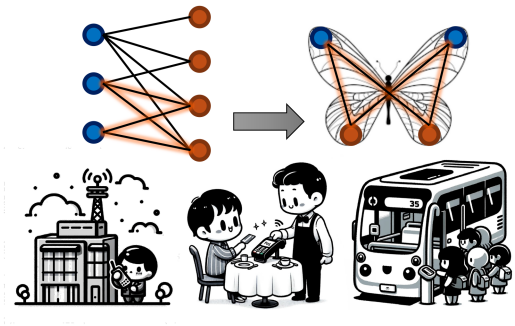

Fig. 1. An example of a butterfly in a bipartite graph. When modeled as an individual-location relationship, a butterfly represents a co-movement pattern.

As a fundamental building block, this type of analysis supports a plethora of crucial societal decisions. These include the dynamic allocation of public resources during emergencies [12], [13], tracking close contact behavior in suspected contagion cases for disease control [14], and identifying risks in terrorist attacks and public safety incidents [15], [16]. In the recent global pandemic, a profoundly memorable event, governments worldwide employ various technological strategies in the early stages of combating the disease. These approaches focus on assessing the transmission paths of detected cases, enabling the implementation of effective measures to curb the spread [17], [18]. This method marks a departure from historical responses to pandemics, as it was previously challenging to record large-scale population movements effectively.

The enhanced availability of binary relationship records enables more detailed and precise analysis. However, the scale of data generated from such analyses is considerable, necessitating the development of efficient and effective analytical methods to optimally utilize this data for public health benefits. In this context, studying butterfly patterns in bipartite graphs plays a particularly prominent role [19], [20], [21]. As illustrated in Fig. 1, when individuals and locations are modeled as a

Received 2 February 2024; revised 25 July 2024; accepted 17 October 2024. Date of publication 24 October 2024; date of current version 12 January 2025. This work was supported in part by the Creative Research Groups Program of the National Natural Science Foundation of China under Grant 62321003, Grant 62402168, and Grant U23A20322, in part by the Natural Science Foundation of China under Grant U23A20317, Grant 62172146, and Grant 62402481, in part by the Natural Science Foundation of Hunan Province under Grant 2023JJ10016 and Grant 2023JJ30083, in part by the Key R&D Program of Hunan Province under Grant 2023GK2002, and in part by the Programs of Hunan Province under Grant 2024JJ6156. Recommended for acceptance by A. Bonifati. *(Corresponding authors: Kian-Lee Tan; Yumeng Liu.)*

Tongfeng Weng and Kian-Lee Tan are with the School of Computing, National University of Singapore, Singapore 117417 (e-mail: tf.weng@nus.edu.sg; tankl@comp.nus.edu.sg).

Yumeng Liu is with the Institute of Software, Chinese Academy of Sciences, Beijing 100045, China (e-mail: yumeng@iscas.ac.cn).

Mo Sha is with Alibaba Cloud, Singapore 189554 (e-mail: shamo.sm@alibaba-inc.com).

Xinyuan Chen is with the College of Automation Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China (e-mail: 2671236149@nuaa.edu.cn).

Xu Zhou and Kenli Li are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China (e-mail: zhxu@hnu.edu.cn; lkl@hnu.edu.cn).

Digital Object Identifier 10.1109/TKDE.2024.3486310

bipartite graph, a butterfly pattern, or a (2,2)-biclique, signifies the concurrent presence of two individuals at two identical locations. Taking the butterfly count of each vertex as a metric effectively characterizes the vertex's significance in potential transmission pathways. Furthermore, this concept can support cluster mining. k-tips are defined as k-cohesive subgraphs in a bipartite graph [22], where each vertex is involved in at least k butterflies as an effective tool for identifying infection groups.

For instance, during critical moments in the fight against infectious diseases, countries adopt more overt methods to collect contact data, constructing a more comprehensive bipartite graph representation. Particularly in Singapore, the widespread deployment of the Exposure Notification System [23] (ENS) utilizes mobile devices' low-energy Bluetooth discovery mechanisms to record proximity interactions. The government also implements the TraceTogether system, distributing free hardware tokens and deploying numerous short-wave devices in public and congregational areas to collect token broadcast signals [24]. Furthermore, the SafeEntry system requires scanning QR codes at checkpoints to log interval information for entering and exiting [25]. The k-tip decomposition of the dynamic people-places bipartite graph significantly covers key communities in the social transmission of disease in time, providing vital support for public health decision-making. This technological evolution in disease control reflects a paradigm shift in managing global health crises, leveraging digital tools to enhance response efficiency and accuracy [26], [27], [28].

The core premise of these mining applications is the decomposition of bipartite graphs into butterfly patterns for efficient counting. However, subgraph pattern counting, including bipartite butterfly counting, remains computationally intensive even with modern hardware [29], [30], [31], [32], and bipartite graph butterfly counting is no exception. Although the current state-of-the-art algorithm has been optimized to $O(\sum_{v\in V} d(v)^2)$ [33], further refinement is necessary. Crucially, existing butterfly counting algorithms are predominantly static, requiring complete recalculations upon any graph update. This static nature is problematic given the dynamic nature of real-world graphs, characterized by continuous edge insertions and deletions [34]. The limitations of these static methods are particularly evident in scenarios demanding timely insights, such as disease outbreak control or emergency resource allocation.

To address these limitations, researchers have developed approximate algorithms for dynamic maintenance [35], [36], [37]. While these methods offer computational advantages for large-scale data streams, their reliance on approximations can involve inaccuracies. In applications where precision is critical, such as contact tracing during a pandemic, these inaccuracies can be detrimental. The need for an exact dynamic method is underscored by the demand for precise and immediate understanding in crisis management scenarios. Missed connections or overlooked patterns can result in missed opportunities for intervention, highlighting the need for algorithms that maintain butterfly patterns with both efficiency and accuracy. This paper aims to bridge this gap.

In our pursuit of optimizing butterfly counting methodologies, we conduct a comprehensive investigation into the primary performance bottlenecks of existing frameworks. Current computational frameworks, especially those involving subgraph decomposition, face significant overhead due to the topological characteristics of the butterfly pattern, which necessitates extensive access to 2-hop neighbors [38]. To address these challenges, we introduce a projection-based pattern decomposition algorithm (PTDA) that adeptly handles the evolution of the original graph's structure. PTDA begins by deploying an exact static algorithm to transform the original bipartite graph into a weighted general graph. This transformation is pivotal as it projects each edge's weight to represent the shared butterfly count between two vertices, encapsulating intricate co-movement patterns and inter-vertex relationships in a form amenable to efficient analysis. This foundational phase ensures an accurate initial state for subsequent dynamic updates, which is crucial in applications where minor inaccuracies can have significant implications. The experimental evaluations (Section V) demonstrate that our algorithm not only achieves a more effective decomposition of tips compared to existing solutions but also reveals that vertices with changes in tip number exhibit strong local connectivity.

According to the PTDA experimental observation, we further propose tip maintenance algorithms (TMA) for pruning the projection as the graph evolves, specifically focusing on edge weights that represent the interrelation between vertices. TMA leverages the insight that changes to the bipartite graph's structure have localized effects on its projection, affecting only the vertices involved in or adjacent to the altered edges. This realization enables a targeted approach to updating the projection upon graph modifications. Instead of recalculating from scratch, we identify the affected vertices within our weighted general graph. This allows us to selectively reconstruct projections for these specific candidates, significantly reducing the computational overhead. The synergy between PTDA and TMA is pivotal. The accuracy of the initial static algorithm ensures that our dynamic heuristics operate on a precise baseline, while the efficiency of the localized update strategy preserves computational resources as data evolves continuously. PTDA is designed to adapt and thrive within perpetually changing landscapes without compromising on either precision or performance. This dual approach optimizes computational resources and underscores TMA's adaptability to a wide array of scenarios requiring both accuracy and agility, positioning our method at the forefront of bipartite graph analysis methodologies.

Our major contributions are summarized as follows.

- We propose a novel projection-based tip decomposition algorithm to effectively manage tip counts in dynamic bipartite graphs, which does not incur additional computational costs during projection generation. Moreover, we note that this technique allows for the pruning of 2-hop neighbor access operations, thereby enhancing efficiency.
- We delve into the incremental tip maintenance algorithms for both the insertion and deletion of edges. This involves the innovative "sub-edge generation" strategy, which breaks down the updated edge into multiple sub-edges. Additionally, we have investigated various theoretical lemmas pertinent to the retrieval of candidate

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
$G = (U, V, E)$	A bipartite graph
$\Psi(G) = (U, E, W)$	The projection graph of a bipartite graph G
$N(u)$	A neighbor set of vertex u
$d(u)$	The degree of vertex u
$d_{\bowtie}(u)$	The butterfly degree of vertex u
$H = (U', V', E')$	A subgraph H of G
$d_H(u)$	The induced degree of u in H
$\bowtie_{u_1}^{u_2}$	A butterfly contains vertices u_1 and u_2
$\omega_{\bowtie}(u_1, u_2)$	The number of sharing butterflies between u_1, u_2
$\theta(u)$	u's tip number
$H_{\theta(u)}$	The $\theta(u)$-tip H contains vertex u

vertices, aiding in the construction of a candidate projection. Utilizing these strategies and lemmas, we present two algorithms, TMA-Ins and TMA-Del, designed for the incremental maintenance of tip numbers.

- We evaluate the performance and efficacy of our proposed algorithms on eight real-world bipartite graphs. The results from these experiments substantiate the algorithms' effectiveness and efficiency in practical scenarios.

II. PRELIMINARIES

A. Problem Formulation

In this work, we explore incremental maintenance of k -tip decomposition of dynamic networks modeled as undirected and unweighted bipartite graphs. In this section, we introduce several definitions that are used throughout the whole paper. Table I gives the notations.

In a bipartite graph $G = (U, V, E)$, where U and V denote two types of entities, each edge $e \in E$ connects a vertex from U to one from V. Thus, for $u \in U$, its neighbors $N(u)$ belong to V. The degree of a vertex u, denoted as $d(u)$, equals the number of vertices in its neighborhood $N(u)$. As there are no direct edges between vertices of the same type, we introduce a motif called a butterfly, denoted as $\bowtie$, to measure their connection. A butterfly is a (2,2)-clique consisting of four vertices, two from U and two from V, with four edges among them. The butterfly degree of a vertex u, denoted as $d_{\bowtie}(u)$, indicates the number of butterflies that u participates in. We use $\bowtie_{u_1}^{u_2}$ to indicate that vertices u_1 and u_2 share a butterfly, and $\omega_{\bowtie}(u_1, u_2)$ denotes the number of shared butterflies between u_1 and u_2. More specific definitions are provided below.

Definition 1. (Wedge): In a bipartite graph, a wedge is defined as a structure consisting of three vertices and two edges among them, which can be represented as either (u_1, v_1, u_2) or (v_1, u_1, v_2).

Definition 2. (Butterfly): In graph $G = (U, V, E)$, a butterfly is defined as a $(2, 2)$-clique $H = (U', V', E') \subseteq G$, where U' and V' each contain two vertices. E' consists of four edges among these four vertices. A butterfly, denoted as $\bowtie$, contains two wedges.

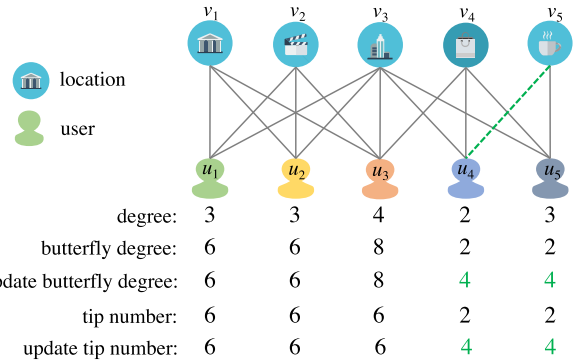

Fig. 2. A toy bipartite graph. The green dotted edge $e = (u_4, v_5)$ will be inserted into the source graph.

A k-tip is a cohesive subgraph consisting of vertices from one type of entity, either set U or set V. In this work, we specifically focus on set U. We denote the tip number of vertex u as $\theta(u)$. The definitions of k-tip community and tip maintenance are provided below.

Definition 3. (k-tip): In graph $G = (U, V, E)$, if a subgraph $H = (U', V', E')$ is a k-tip, it should satisfy

- *Connectivity:* vertices in U' are reachable via butterflies.
- *Tightness:* each vertex $u \in U'$ participates in at least k butterflies.
- *Uniqueness:* no other k-tip subsumes H.

Definition 4. (Tip number): In graph $G = (U, V, E)$, the tip number of $u \in U$ is defined as the maximum value k for which there exists a k-tip containing u, denoted as $\theta(u)$. The $k_{\max}$-tip H that contains u is denoted as $H_{\theta(u)}$.

Definition 5. (Support): The support of a vertex $u \in U$ is the number of 2-hop neighbors that have tip numbers not less than $\theta(u)$.

Problem 1. (Tip decomposition): In graph $G = (U, V, E)$, tip decomposition is designed to calculate the tip numbers of vertices in U.

Problem 2. (Tip maintenance): Given a bipartite graph $G = (U, V, E)$ and a set of new edges that will be inserted into or deleted from G, tip maintenance is proposed to maintain tip numbers of vertices in U.

Example: We use Fig. 2 to illustrate these above definitions. Given the toy graph $G = (U, V, E)$, the source graph has 10 users (vertices) and 15 locations (edges). Edges $e_1 = (v_1, u_1)$ and $e_2 = (u_1, v_2)$ form a wedge. Also, edges $e_3 = (v_1, u_2)$ and $e_4 = (u_2, v_2)$ form another wedge. These two wedges form a butterfly. According to Definition 2, the butterfly degree is shown in the second row. After peeling vertices that have the minimum tip number iteratively, we can obtain the tip number of vertices in U (the fourth row in Fig. 2). Finally, we get a 4-tip community consisting of vertices $\{u_4, u_5\}$ and a 6-tip community $\{u_1, u_2, u_3\}$.

B. Bottom-Up Peeling Tip Decomposition Algorithm

To mine cohesive communities consisting of vertices from the same type of entity, Sariyuce et al. [22] introduced k-tip

Algorithm 1: Bottom-Up Peeling Tip Decomposition (BUP).

Input: Bipartite graph $G{=}(U, V, E)$
Output: Tip numbers $\theta(u)\ \forall u \in U$
1 Count butterfly degree of vertex $u \in U$
2 **while** $U \neq \varnothing$ **do**
3 Peel $u \in U$ which has the minimum butterfly degree
4 Update butterfly degrees of 2-hop neighbors of u

community to identify vertex-induced subgraphs with a large number of butterflies. Given a bipartite graph $G = (U, V, E)$, there are two main phases to get the tip number of each vertex. First, we need to count the butterfly degree of all vertices in U. Chiba and Nishizeki [39] proposed an efficient vertex-priority-based algorithm to count the number of butterflies for each vertex that it can participate in. The algorithm only traverses wedges where the degree of the last vertex is greater than the degree of the start and middle vertices. It traverses $O(\sum_{(u,v)\in E} min(d_u, d_v))$ wedges. The parallel version of the algorithm [40], [41] processes multiple start vertices concurrently. This parallel variant for per-vertex counting adds the contributions from traversed wedges to start, mid, and end vertices.

It is inefficient to store all k-tips of a bipartite graph because a k-tip overlaps with k'-tips for all $k' \leq k$ [33]. As shown in Definition 4, $\theta(u)$ is defined as the maximum k for which u can participate in a k-tip. Tip numbers provide a space-efficient representation of k-tip hierarchy with quick retrieval. The bottom-up peeling tip decomposition algorithm is presented in Algorithm 1. In graph $G = (U, V, E)$, the algorithm counts butterfly degrees for vertices from U. Then, it peels the vertex u with minimum butterfly degree and the tip number of u is set as the current butterfly degree (Line 3). Subsequently, the 2-hop neighbors of vertex u adjust their butterfly degrees by deducting the count of shared butterflies with u, with a maximum reduction capped at $\theta(u)$ (i.e., $\theta(u') = Max(\theta(u') - \omega(u, u'), \theta(u))$). (Line 4). When all vertices in U are peeled, the tip numbers are obtained (Lines 2-4).

III. Tip Number Maintenance on Projection

Different from the decomposition algorithm on unipartite graphs, there are more 2-hop neighbor access operations on bipartite graphs. First, the degree of a vertex in a unipartite graph is equal to the number of neighbors while we need to find the number of sharing butterflies of each pair of vertices in a unipartite graph. According to Definition 2, it needs two 2-hop neighbor access operations to find a butterfly. Second, there are also many 2-hop neighbor access operations during tip decomposition when updating butterfly degrees of remaining vertices (Algorithm 1 line 4). Therefore, applying the tip decomposition algorithm to maintain tip numbers on dynamic bipartite graphs is time-consuming.

To avoid 2-hop access operations when maintaining tip numbers in dynamic bipartite graphs, we propose a projection-based tip maintenance algorithm (PTMA). As introduced in Section II,

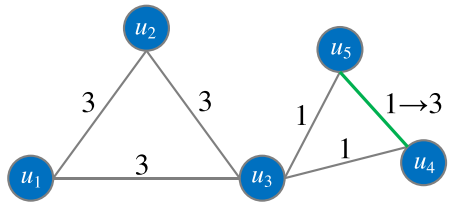

Fig. 3. The projection of Fig. 2. The weight of edge $e = (u_4, u_5)$ increases from 1 to 3 because the new edge $e = (u_4, v_5)$ insertion.

the tip decomposition algorithm should first count the butterfly degree of each vertex in U. In this process, we can construct a projection $\Psi(G) = (U, E, W)$ of the source bipartite graph $G = (U, V, E)$ (i.e., convert the bipartite graph to a weighted unipartite graph). Specifically, vertices in the projection represent those vertices from U of the source bipartite graph G and the weight of each edge indicates the number of sharing butterflies between the two endpoints. As shown in Fig. 3, it is a projection of the toy bipartite graph in Fig. 2. Vertices u_1 and u_2 have three common neighbors v_1, v_2, and v_3, and there are three butterflies containing u_1 and u_2. The weight of edge $e = (u_1, u_2)$ should be 3. Before we present PTDA, some lemmas will be introduced in advance.

Observation 1: In graph $G = (U, V, E)$ and a new edge $e = (u, v)$ that will be inserted into or deleted from G, edges in the projection have weight changes should be $E' = \{(u, u')|u' \in N(v)\}$. The new weights of edges in E' can be updated by

$$ori\_sharing\_nei = \frac{1 + \sqrt{1 + 8 \times ori\_weight}}{2}, \tag{1}$$

$$\omega = \frac{(ori\_sharing\_nei \pm 1) \times (ori\_sharing\_nei \pm 1 - 1)}{2}, \tag{2}$$

where $+$ and $-$ indicate edge insertion and deletion, respectively.

Proof: Vertex v is a common neighbor of vertex u and $N(v)$ and sharing butterflies are generated from these common neighbors, therefore only edges in E' may have weights changed.

Given two vertices u_1 and u_2 with n common neighbors, $\omega_{\triangleright\triangleleft}(u_1, u_2)$ can be calculated by

$$\omega_{\triangleright\triangleleft}(u_1, u_2) = \frac{n \times (n-1)}{2}. \tag{3}$$

Then, we can get the $ori\_sharing\_nei$ by (3). For edge insertion, n should be increased by 1 and minus 1 for edge deletion.

The projection of a bipartite graph results in a weighted unipartite graph, where the edge weights represent the number of shared butterflies between the two endpoints in the original bipartite graph. Updating an edge in the bipartite graph will alter the weights of the corresponding edges in the projection. The new weights can be computed using (2). Consequently, the updated projection can be maintained by adjusting the weights of the relevant edges after inserting or deleting an edge in the original bipartite graph. If an edge's weight increases from 0 to 1, this will introduce a new edge with weight 1 into the projection.

Example: We use the toy graph (Fig. 2) and its projection (Fig. 3) to illustrate the weight update process. After inserting

Algorithm 2: Projection-Based Tip Decomposition.

Input: Projection $\Psi(G) = (U, E, W)$, butterfly degree array tip
Output: Tip numbers $\theta(u)\ \forall u \in U$

1 Initialize $vert$ array to store vertices by the ascending order of butterfly degree
2 Initialize bin array to store the start offset of different degrees in $vert$
3 Initialize pos array to store the position of each vertex in $vert$
4 **for** $u \in vert$ **do**
5 **for** $u' \in N(u)$ **do**
6 **while** $d_{\bowtie}(u') > d_{\bowtie}(u)$ && $\omega_{\bowtie}(u, u') > 0$ **do**
7 $du \leftarrow d_{\bowtie}(u'), pu \leftarrow pos[u']$
8 $pw \leftarrow bin[du], w \leftarrow vert[pw]$
9 **if** $u' \neq w$ **then**
10 $pos[u'] \leftarrow pw, vert[pu] \leftarrow w$
11 $pos[w] \leftarrow pu, vert[pw] \leftarrow u$
12 $bin[du] \leftarrow bin[du] + 1$
13 $d_{\bowtie}(u') \leftarrow d_{\bowtie}(u') - 1$
14 $\omega_{\bowtie}(u, u') \leftarrow \omega_{\bowtie}(u, u')$ - 1
15 **return** $d_{\bowtie}(u) | u \in U$

edge $e = (u_4, v_5)$ into G, the number of common neighbors of u_4 and u_5 increases from 2 to 3. According to Observation 1, the weight of edge $e = (u_4, u_5)$ in projection $\Psi(G)$ should be 3.

Based on the new projection, we propose a peeling-based decomposition algorithm to compute the tip numbers of vertices. This algorithm is inspired by WG_BZ [42], the state-of-the-art method for core decomposition in unipartite graphs [43]. The pseudocode is provided in Algorithm 2.

PTDA: Projection-based Tip Decomposition Algorithm. Given the projection $\Psi = (U, E, W)$ of a bipartite graph $G = (U, V, E)$, PTDA peels vertices in U according to their butterfly degree. First, the $vert$ array is initialized to store vertices in U by the ascending order of butterfly degrees (Line 1). Vertices with the same butterfly degree are stored consecutively on $vert$ and the start offset of this scope of vertex is recorded in bin (Line 2). The pos array is used to store the index of each vertex in $vert$ (Line 3). Then, PTDA processes vertices in $vert$ sequentially (Lines 4-15). For a specific vertex u, its neighbors in $N(u)$, where u' is a 2-hop neighbor of u in G, update butterfly degrees according to the edge weight $\omega_{\bowtie}(u, u')$ (Lines 6-14). In each iteration, u' is relocated to the starting offset of the vertex group with butterfly degrees equal to $d_{\bowtie}(u')$ (Lines 7-11). Subsequently, Lines 12-13 adjust the offset array bin and decrement u''s butterfly degree by 1.

Complexity: Given a bipartite graph $G = (U, V, E)$, $\Psi = (U, E', W)$ is the projection, where each edge $e' = (u, u') \in E'$ indicates that the two vertices from U can be connected by at least one butterfly. As a result, $|E'| \leq C^2_{|U|}$. According to Algorithm 2, after peeling a vertex u, the butterfly degree of its neighbor u' is reduced by at most $\omega_{\bowtie}(u, u')$. All peeled vertices and related edges will not be accessed later. As a result, the complexity of PTDA is $O(\sum_{e' \in E'} \omega(e'))$, where E' is the weighted edge set in the projection $\Psi = (U, E', W)$. If we apply the conventional peeling algorithm [1] on the source bipartite graph, each weighted edge $e' = (u, u') \in E'$ needs another $\frac{\sqrt{1+8\times\omega(e')}+1}{2} \times 2 = \sqrt{1 + 8 \times \omega(e')} + 1$ neighbor accesses ((1)) to find these sharing butterflies.

Example: Fig. 4 illustrates the process of peeling operation on projection. The weight of edge $e = (u_4, u_5)$ increases from 1 to 3 because of the new edge $e = (u_4, v_5)$ insertion on the source bipartite graph. PTDA first initializes the arrays bin and $vert$, as depicted in step-1 of Fig. 4. In $vert$, vertices are sorted by their butterfly degrees, while bin holds the starting index in $vert$ for vertices with a specific butterfly degree. For instance, given that $d_{\bowtie}(u_4) = 4$ and the index of u_4 in $vert$ is 0, we have $bin[4] = 0$. Since vertex u_4 has the minimum butterfly degree of 4, it is peeled first. Step-2 of Fig. 4 illustrates the process of peeling u_4. During this step, the butterfly degree of u_3 decreases from 8 to 7. Consequently, $bin[7]$ is set to 4, the index of u_3 in $vert$. In step-3, after removing u_5, the butterfly degrees of u_1, u_2, and u_3 all decrease to 6. Since u_1 is the first vertex with a butterfly degree of 6 and its index in $vert$ is 2, we set $bin[6] = 2$. Subsequently, u_1, u_2, and u_3 are peeled from the projection. After peeling, the tip numbers of u_4 and u_5 increase from 2 to 4.

IV. Incremental Tip Maintenance Algorithms

Although PTDA can avoid 2-hop access while maintaining tip numbers in a bipartite graph, it generates many more edges in the projection than the source bipartite graph. We take the dataset YouTube as an example to describe this scenario. The dataset consists of 131,072 vertices in U, 32,768 vertices in V, and 293,360 edges in E. The projection of YouTube has 59,826,175 weighted edges. It may lead to the OOM (out of memory) problem while processing large-scale dynamic bipartite graphs. Therefore, the proposed PTDA cannot maintain tip numbers in large graphs effectively.

According to the experimental results of PTDA in Section V, we found that the number of vertices that have tip numbers changed due to edge insertion or deletion is a small subset vertex of the source graph. For example, when we insert edge u_4, v_5 into the toy graph in Fig. 2, only vertices u_4 and u_5 need to change their tip numbers. This inspires the research of locally-based incremental tip maintenance algorithms.

A. Theoretical Findings

In graph $G = (U, V, E)$, there is an edge $e = (u, v)$ inserted into or deleted from G. The number of common vertices between u and $N(v)$ will be changed by 1, then the number of sharing butterflies containing these pairs of vertices will be changed. As a result, the butterfly degree of some vertices in U may be increased or reduced. Unlike unipartite graphs, the magnitude of changes in butterfly degree is usually more than 1. The conventional incremental theorems for k-core maintenance [44], [45], [46] cannot be applied to bipartite graphs.

According to Observation 1, only $\omega_{\bowtie}(u, u')$, where $u' \in N(v)$, will be changed. The changed values of different u' are

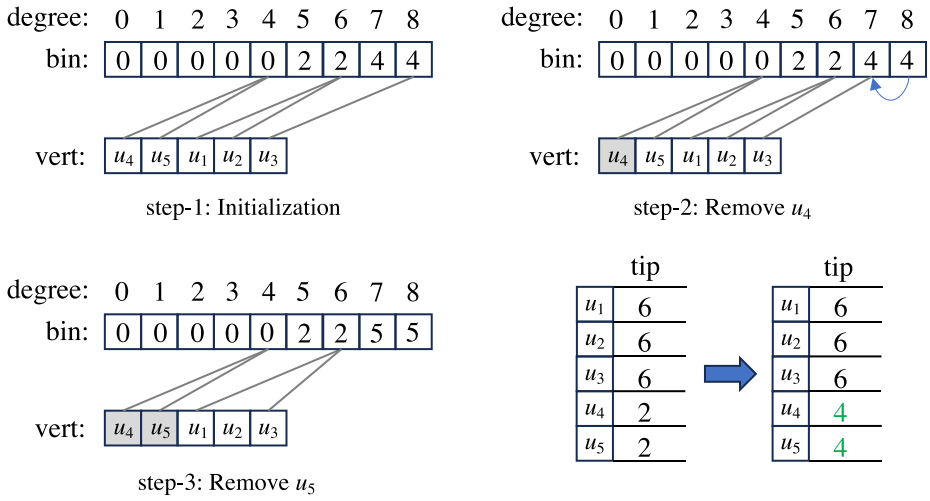

Fig. 4. PTDA on the projection in Fig. 3.

not the same. It is difficult to locate vertices that may have tip numbers changed. To find candidate vertices that may experience changes in their tip numbers accurately, we process these pairs of vertices separately.

Strategy 1. sub-edge generate: Given a bipartite graph $G = (U, V, E)$, there is an edge $e = (u, v)$ inserted into or deleted from G. The sub-edge set is $\xi = \{e = (u, u')|u' \in N(v) \setminus u\}$.

For each sub-edge, we process it separately. Specifically, we first find a candidate vertex set, in which vertex's tip number may be changed. It needs to be noted that before all sub-edges are processed, the new edge $e = (u, v)$ should not be updated into G. Otherwise, the support (Definition 5) of candidate vertices will be confused while processing a single sub-edge. The lemma for candidate vertices finding is shown as follows.

Lemma 1: Given a bipartite graph $G = (U, V, E)$ and a new edge $e = (u, v)$, the weight of a sub-edge $e = (u, u')$ is $max(n, m) - 1$ that represents the change in the number of sharing butterflies between u and u', where n and m are the original and new number of common neighbors of u and u', denoted as $\Delta_{\bowtie}(u, u')$.

Proof: For a sub-edge $e = (u, u')$, there are n common neighbors in the bipartite graph. Then the number of sharing butterflies is $n \times (n-1)/2$ ((3)). For edge insertion, $n < m = n + 1$, the number of increment butterflies is

$$\Delta_{\bowtie}(u, u') = \frac{(n+1) \times n}{2} - \frac{n \times (n-1)}{2} = n = m - 1. \tag{4}$$

For edge deletion, $n > m = n - 1$, we can get the same result by

$$\Delta_{\bowtie}(u, u') = \frac{n \times (n-1)}{2} - \frac{(n-1) \times (n-2)}{2} = n - 1. \tag{5}$$

In summary, $\Delta_{\bowtie}(u, u') = max(n, m) - 1$.

Lemma 2: Given a k-tip H_k and $\Delta_{\bowtie}$ butterflies that will be inserted into or deleted from H_k, there must be a k'-tip $H_{k'} \subseteq H_k$, where $k' \in [k - \Delta_{\bowtie}, k + \Delta_{\bowtie}]$.

Proof: We first prove the butterfly deletion scenario. According to the definition of k-tip (Definition 3), vertices in a k-tip can participate in k butterflies (i.e., the induced butterfly degree is not less than k). When the $\Delta_{\bowtie}$ butterflies, the butterfly degree of each vertex in H_k is reduced at most $\Delta_{\bowtie}$. Then, the minimum induced butterfly degree in $H_{k'}$ equals to $k - \Delta_{\bowtie}$. As a result, $k' \geq k - \Delta_{\bowtie}$.

For butterfly insertion, let's delete these inserted butterflies and the tip numbers will be rolled back (i.e., $k' - \Delta_{\bowtie} \leq k$). Therefore, $k' \leq k + \Delta_{\bowtie}$.

By reason of the foregoing, $k' \in [k - \Delta_{\bowtie}, k + \Delta_{\bowtie}]$.

Lemma 3: In graph $G = (U, V, E)$ and a sub-edge $e = (u, u')$, the change in the number of sharing butterfly between u and u' is $\Delta_{\bowtie}(u, u')$ (Lemma 1). For each vertex $\mu \in U$, $|\theta(\mu)' - \theta(\mu)| \leq \Delta_{\bowtie}(u, u')$, where $\theta(\mu)'$ is the new tip number of μ.

Proof. (Add $\Delta_{\bowtie}(u, u')$ butterflies): Assume there is a vertex $\mu \in U$, $\theta(\mu)' - \theta(\mu) > \Delta_{\bowtie}(u, u')$. Then, there is a $\theta(\mu)'$-tip $H_{\theta(\mu)'} \subseteq G + \Delta_{\bowtie}(u, u')$. The $\Delta_{\bowtie}(u, u')$ butterflies must in $H_{\theta(\mu)'}$. Otherwise, $\theta(\mu)$ can be equal to $\theta(\mu)'$ without these $\Delta_{\bowtie}(u, u')$ butterflies. Let $H' = H_{\theta(\mu)'} - \Delta_{\bowtie}(u, u')$. According to Lemma 2, there is a $\theta(\mu)' - \Delta_{\bowtie}(u, u')$-tip in H'. Because $\theta(\mu)' - \Delta_{\bowtie}(u, u') > \theta(\mu)$, the tip number of μ can be larger than $\theta(\mu)$ in G. It is a contradiction.

(Remove $\Delta_{\bowtie}(u, u')$ butterflies): After adding $\Delta_{\bowtie}(u, u')$ butterflies into G, we remove these butterflies again. Then the tip numbers of vertices in U will be recovered. As a result, vertices in U will reduce their tip number by at most $\Delta_{\bowtie}(u, u')$.

Lemma 4. Candidate: Given a bipartite graph $G = (U, V, E)$ and a sub-edge $e = (u, u')$, where $\Delta_{\bowtie}(u, u')$ represents the change in the number of shared butterflies between vertices u and u', the root vertex is defined as $root = \theta(u) \leq \theta(u')\ ?\ u : u'$. Candidate vertices that may experience changes in their tip numbers are determined as follows:

- For edge insertion, vertices that can reach the root vertex $root$ through a path consisting of vertices, which are connected by butterflies, with tip numbers in the range

$[\theta(root), \theta(root) + \Delta_{\bowtie}(u, u'))$ may experience changes in their tip numbers.

- For edge deletion, vertices that can reach the root vertex $root$ through a path consisting of vertices, which are connected by butterflies, with tip numbers in the range $(\theta(root) - \Delta_{\bowtie}(u, u'), \theta(root)]$ may experience changes in their tip numbers.

Proof: According to Definitions 3 and 4, the tip number of a vertex u indicates that it has at least $\theta(u)$ 2-hop neighbors with tip numbers no less than $\theta(u)$. Otherwise, it cannot form a $\theta(u)$-tip community. Therefore, any change in the tip numbers of a vertex u's 2-hop neighbors that have tip numbers no less than $\theta(u)$ can potentially alter the structure of the $\theta(u)$-tip community $H_{\theta(u)}$, thereby affecting $\theta(u)$. If a vertex does not have any 2-hop neighbors whose tip numbers change, then the tip number of that vertex remains unchanged. Thus, the candidate vertex set should form a connected component, where vertices are linked by butterflies. Next, we will prove the lemma for two scenarios: sub-edge insertion and deletion.

Scenario 1. Edge Insertion: For a vertex $\mu \in U$, if its tip number $\theta(\mu)$ increases after processing sub-edge $e = (u, u')$, the new edge $e = (u, v)$ has not yet been inserted into G. However, the butterfly degree increment $\Delta_{\bowtie}(u, u')$ should be considered, representing the additional sharing butterflies between u and u'. In $G + \Delta_{\bowtie}(u, u')$, the new k_max-tip $H_{\theta(\mu)'}$ of μ must contain sub-edge $e = (u, u')$. The inclusion of sub-edge $e = (u, u')$ implies the addition of $\Delta_{\bowtie}(u, u')$ butterflies to both u and u'. If no new butterflies are added to $H_{\theta(\mu)'}$, then $\theta(\mu)' = \theta(\mu)$, leading to a contradiction.

According to Lemma 3, all vertices may change their tip numbers in the range $[0, \Delta_{\bowtie}(u, u')]$. For each vertex with a tip number larger than $\theta(root) + \Delta_{\bowtie}(u, u')$, there will be no additional 2-hop neighbors that can increase its support, defined as the number of 2-hop neighbors with tip numbers larger than the vertex itself (see Definition 5). Therefore, if $\theta(\mu) \geq \theta(root) + \Delta_{\bowtie}(u, u')$, the tip number $\theta(\mu)$ will remain unchanged.

All vertices except the root will not gain new butterflies to join, and therefore, the tip numbers of other vertices will not decrease when inserting new butterflies into the bipartite graph. For a vertex with a tip number less than $\theta(root)$, although some of its 2-hop neighbors may increase their tip numbers, the support of the vertex will remain unchanged, and thus its tip number will not change.

In summary, only vertices with tip numbers in the range $[\theta(root), \theta(root) + \Delta_{\bowtie}(u, u')]$ may experience changes in their tip numbers.

Scenario 2. Edge Deletion: Let's remove the sub-edge from G, and then all vertices will recover their original tip numbers. As discussed earlier, after inserting the sub-edge, the maximum new tip number will be $\theta(root)_{ori} + \Delta_{\bowtie}(u, u')$, and the minimum tip number of those candidate vertices is $\theta(root)_{ori}$, where $\theta(root)_{ori}$ is the tip number of $root$ before inserting the sub-edge. Vertices that have tip numbers larger than $\theta(root)_{ori} + \Delta_{\bowtie}(u, u')$ have not changed their tip numbers. Therefore, these vertices will not experience any reduction in their tip numbers after the sub-edge deletion. Because the minimum tip number of candidate vertices while inserting the sub-edge is $\theta(root)_{ori}$, all vertices can only reduce their tip numbers to $\theta(root)_{ori}$ at the minimum. Vertices where $\theta(\mu) \leq \theta(root)_{ori}$ will not have 2-hop neighbors that reduce their tip numbers from $\geq \theta(root)_{ori}$ to $< \theta(root)_{ori}$. According to Definition 5, these vertices will not experience any changes in their tip numbers.

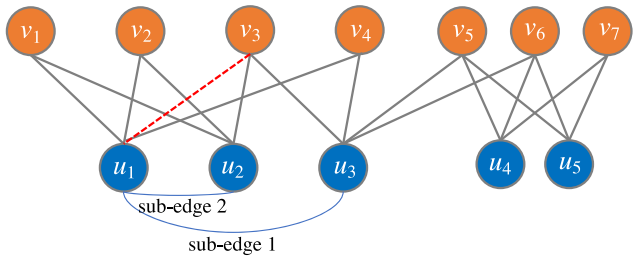

Fig. 5. sub-edge based tip maintenance.

After inserting the sub-edge, $\theta(root)$ should be at most $\theta(root)_{ori} + \Delta_{\bowtie}(u, u')$. Then, when we delete the sub-edge, we conclude that only vertices with tip numbers in $(\theta(root) - \Delta_{\bowtie}(u, u'), \theta(root)]$ may have their tip numbers changed.

Lemma 5. Correctness: Given a bipartite graph $G = (U, V, E)$ and a new edge $e = (u, v)$, the sub-edge set is denoted as $\xi = e = (u, u')|u' \in N(v) \setminus u$. After processing each sub-edge sequentially, the tip numbers will be maintained. Note: when processing a sub-edge, it should be aware of the sub-edges that have been processed previously.

Proof: As outlined in the PTDA algorithm proposed in Section III, when a new edge is inserted into or deleted from the source bipartite graph, it can be decomposed into several new weighted edges in the corresponding projection. It is evident that these new weighted edges, akin to sub-edges, can be sequentially inserted into the projection and PTDA can be applied to update the tip numbers for each insertion. Consequently, after processing all sub-edges, tip numbers will be accurately maintained.

B. Incremental Tip Maintenance Algorithm

In this subsection, we introduce two algorithms for edge insertion and edge deletion, respectively. Our algorithms comprise two main phases: the candidate-finding phase and the candidate-peeling phase. For a new update edge, we initially generate a sub-edge set according to Strategy 1. Subsequently, we employ our incremental tip maintenance algorithms for each sub-edge to maintain tip numbers.

It is important to note that the new edge will not be inserted into G before all sub-edges are processed. Otherwise, the candidate-finding phase may collect irregular vertices. However, in a bipartite graph, we cannot insert a sub-edge explicitly. To address this issue, we add butterfly degrees before applying the peeling operation. The specific algorithm for edge insertion is depicted in Algorithm 3.

Algorithm 3: Tip Maintenance Algorithm for Edge Insertion (TMA-Ins).

Input: Bipartite graph $G = (U, V, E)$, new edge $e = (u, v)$
Output: Tip numbers $\theta(u)\ \forall u \in U$

```
1  /****** Sub-edge Generation ******/
2  Insert edge e = (u, v) into G
3  Initialize a hasp_map
     sub_edge_map = {(key, −1)|key ∈ N(v)}
4  for v' ∈ N(u) do
5      for u' ∈ N(v') do
6          if u' ∈ N(v) then
7              sub_edge_map[u'] + +
8  Delete edge e = (u, v) from G
9  /****** Tip Maintenance ******/
10 for sub-edge (u', Δ⋈(u, u')) ∈ sub_edge_map do
11     root ← the endpoint that has a smaller tip
         number
12     min_tip ← θ(root)
13     max_tip ← θ(root) + Δ⋈(u, u') − 1
14     initialize a empty projection Ψ
15     FindCandidate(root, sub_edge_map, Ψ)
16     PeelCandidate(Ψ)
17 Insert edge e = (u, v) into G
18 return θ(u) ∀u ∈ U
```

Function: FindCandidate($root$, sub_edge_map, Ψ).

Input: Bipartite graph $G = (U, V, E)$, min_tip, max_tip
Output: The local projection Ψ consisting of candidate vertices

```
1  Initialize active set Frontier = {root}
2  while Frontier ≠ ∅ do
3      Pop u from Frontier
4      for v ∈ N(u) do
5          for u' ∈ N(v) do
6              // for deletion should be
                 θ(u) ∈ (min_tip, max_tip]
7              if θ(u') ∈ [min_tip, max_tip) then
8                  Ψ[u][u'] + +
9                  if u' ∉ Frontier and u' ∉ Ψ then
10                     Put u' into Frontier
11 Convert the number of common neighbors in Ψ to
     butterfly degree by Equation 3
12 // for deletion should remove butterflies from
     processed sub-edges
13 Add butterflies from processed sub-edges into Ψ
14 return Ψ
```

Function: PeelCandidate(Ψ).

Input: The candidate projection Ψ
Output: Tip numbers $\theta(u)\ \forall u \in \Psi$

```
1  while Ψ ≠ ∅ do
2      Peel u from Ψ with the minimum induced
         butterfly degree
3      for u' ∈ N(u) do
4          ω⋈(u, u') ← Ψ[u][u']
5          d⋈(u') ← max(d⋈(u') − ω⋈(u, u'), θ(u))
6  return θ(u) ∀u ∈ Ψ
```

Algorithm TMA-Ins: Tip Maintenance Algorithm for Edge Insertion. Line 2 first inserts the new edge into G to generate sub-edges and calculate the corresponding weights (Lines 3-7). Line 8 deletes the new edge to ensure the accuracy of the candidate-finding operation. For each sub-edge, the endpoint with a smaller tip number is set as $root$ (Line 11). Lines 12-13 set the boundaries of vertices that may experience changes in their tip numbers according to Lemma 4. Then, we apply two functions to find and peel the candidate projection Ψ. Here, the candidate projection Ψ is a weighted graph that only consists of vertices that may experience changes in their tip numbers. In Ψ, the weight of each edge denotes the number of sharing butterflies between the pair of candidates. Candidate-peeling can be performed on this intermediate projection directly without 2-hop neighbor access.

In Function FindCandidate, we start from $root$ and expand to 2-hop neighbors to find candidate vertices with tip numbers in the satisfied range (Lines 4-8). The new candidates are put into $Frontier$ for the next candidate-finding iteration (Lines 9-10). As emphasized before, we should update these processed sub-edges into the projection Ψ. Line 11 converts the number of common neighbors in Ψ to butterfly degree by (3). For edge deletion, line 12 removes butterflies from processed sub-edges. Line 13 adds these additional butterflies into Ψ.

Now the tip numbers of vertices in Ψ can be updated by the candidate-peeling operation. The PTDA (Algorithm 2) cannot be applied to peel this candidate projection Ψ directly. This is because vertices in Ψ are usually a small part of the source vertex set U. The arrays initialized in PTDA are designed for all vertices, where an index of an array indicates the ID of the corresponding vertex. In Function PeelCandidate,, we peel the vertex in Ψ with the minimum tip number iteratively (Lines 1-5). When a vertex u is peeled from Ψ, the induced neighbors of u in Ψ need to update their induced butterfly degree(Lines 3-5).

After processing all sub-edges, vertices in the candidate projection Ψ get their new tip numbers, and the new edge $e = (u, v)$ is updated in G (Line 17).

Example: We use the toy graph in Fig. 5 to illustrate these lemmas and Algorithm 3. In the scenario where the dotted new edge $e = (u_1, v_3)$ is to be inserted into G, according to Strategy 1, we generate the sub-edge set $\xi = e_1 = (u_1, u_3), e_2 = (u_1, u_2)$, where $\Delta_{\bowtie}(e_1) = 1$ and $\Delta_{\bowtie}(e_2) = 2$. When we insert sub-edge e_1 into G, the butterfly degrees of u_1 and u_3 will increase by 1. The initial tip numbers of u_1 and u_3 are 1 and 2, respectively. We set u_1 as the $root$. According to Lemma 4, the candidate vertices should have tip numbers in the range $[1, 2)$. Thus, the candidate set is u_1, u_2. After the peeling operation on the candidate set, no vertex has its tip number changed. For sub-edge

Algorithm 4: Tip Maintenance Algorithm for Edge Deletion (TMA-Del).

Input: Bipartite graph $G = (U, V, E)$, deletion edge $e = (u, v)$
Output: Tip numbers $\theta(u)\ \forall u \in U$

```
1  /****** Sub-edge Generation ******/
2  Initialize a hasp_map
     sub_edge_map = {(key, −1)|key ∈ N(v)}
3  for v′ ∈ N(u) do
4      for u′ ∈ N(v′) do
5          if u′ ∈ N(v) then
6              sub_edge_map[u′] + +
7  /****** Tip Maintenance ******/
8  for sub-edge (u′, Δ⋈(u, u′)) ∈ sub_edge_map do
9      root ← the endpoint that has a smaller tip
         number
10     min_tip ← θ(root) − Δ⋈(u, u′) + 1
11     max_tip ← θ(root)
12     initialize a empty projection Ψ
13     FindCandidate(root, sub_edge_map, Ψ)
14     PeelCandidate(Ψ)
15 Delete edge e = (u, v) from G
16 return θ(u) ∀u ∈ U
```

e_2, the butterfly degree increment of u_1 and u_2 is 2. The initial tip numbers of u_1 and u_3 are both 1. One of them will be selected as the $root$. It's important to note that sub-edge $e_1 = (u_1, u_3)$ was processed, and the new butterfly has been added between u_1 and u_3. Therefore, u_3 can reach $root$ via a satisfactory butterfly path. Consequently, the candidate set is u_1, u_2, u_3. After the peeling operation, $\theta(u_1) = 3$, $\theta(u_2) = 3$, and $\theta(u_3) = 3$, yielding the same results as PTDA (Algorithm 2). If the sharing butterfly generated by sub-edge e_1 was ignored while processing sub-edge e_2, the candidate set would be u_1, u_2. After the peeling operation, $\theta(u_1) = 3$ and $\theta(u_2) = 3$, but $\theta(u_3)$ would not change, resulting in an incorrect outcome.

TMA-Del: Tip Maintenance for Edge Deletion. As shown in Algorithm 4, the tip maintenance for edge deletion is roughly similar to TMA-Ins. It first generates sub-edges from the source bipartite graph (Lines 2-6). Because the update edge was in G, the sub-edges can be found directly without modifying G. Then, for each sub-edge, the tip number range of candidate vertices is set according to Lemma 4 (Lines 10-11). After that, TMA-Del applies candidate-finding Function FindCandidate and candidate-peeling Function PeelCandidate, operations to update the tip numbers of these candidates. In the end, the edge $e = (e, v)$ is deleted from G (Line 15).

Complexity: For a sub-edge, we need to generate a candidate projection $\Psi(U', E', W)$ from the given bipartite graph $G = (U, V, E)$, where U' and E' are the candidate vertex set and weighted edges among them, respectively. The most complicated scenario is that all vertices in U are candidate vertices (i.e., $U' = U$). Therefore, the maximum complexity of Function FindCandidate is $\sum_{u \in U} \sum_{v \in N(u)} \sum_{u' \in N(v)} \delta(u')$, where $\delta(u')$ indicates if u' is a new candidate vertex. Function PeelCandidate peels the candidate projection $\Psi(U', E', W)$ generated by Function FindCandidate. All edges in Ψ will be accessed once. Hence, the complexity of Function PeelCandidate is $O|E'|$, where $|E'|$ is the number of weighted edges in Ψ. The maximum complexity is $O|E|$ when all vertices in U are candidates. In this worst scenario, the conventional tip decomposition algorithm would be more efficient because it can peel the source bipartite graph directly without constructing the candidate projection.

V. Experiment

A. Settings

The first set of experiments compares the performance of PTDA (Algorithm 2) with the bottom-up peeling-based tip decomposition (BUP) [22] and the parallel algorithm (RECEIPT) [33], by studying the runtime for per-edge update using 64 threads. In addition, we analyze the additional space overhead introduced by PTDA by counting the number of edges and the scope of vertices that have tip numbers changed after each edge update on average. The second set of experiments shows the performance of TMA-Ins and TMA-Del for edge insertion and edge deletion, respectively. Specifically, we investigate the runtime performance of BUP, RECEIPT, PTDA, and TMA-Ins & TMA-Del for small bipartite graphs. Due to the space limitation, we only evaluate the performance of RECEIPT and TMA-Ins & TMA-Del for large-scale bipartite graphs.

In our experiment, we randomly select 500 edges from distinct vertices in set U of the bipartite graph $G = (U, V, E)$. These edges are then deleted from G and subsequently reinserted. Then we invoke BUP, RECEIPT, and our proposed algorithms PTDA, TMA-Ins, and TMA-Del to calculate the update tip numbers of each vertex $u \in U$. Following the experimental setup outlined in [45], we compute the average time taken for each new edge update and compare it to the time required by the BUP and RECEIPT algorithms. In addition, we conduct statistical tests on five synthetic bipartite graphs to evaluate the performance and verify the scalability across bipartite graphs of various sizes. The projection construction phase is considered a preprocessing step, so it is generally not regarded as part of the main computational processes.

Our algorithms are implemented in C++ and compiled with *gcc* 9.4.0. All sets of experiments are conducted on Linux OS running on a server with Intel(R) Xeon(R) Gold 6226R CPU @ 2.90 GHz, with 192 GB of RAM.

B. Datasets

Our experiments are all evaluated on real-world bipartite graphs [47], [48], [49]. The descriptions of each dataset are shown in Table II. For the first six datasets (YT, BC, DR, DT, GH, and IM), we apply BUP, RECEIPT, PTDA and TMA-Ins/TMA-Del to maintain tip numbers. For the last two datasets (TR and OR), we only use RECEIPT and TMA-Ins/TMA-Del. This is because BUP is too time-consuming and Function FindCandidate will lead to an OOM problem. All datasets can be downloaded from http://konect.cc/networks/ [50]. In addition,

TABLE II
REAL-WORLD AND SYNTHETIC BIPARTITE GRAPHS

Graph	Description	$\|U\|$	$\|V\|$	$\|E\|$	Projection $\|E'\|$	$\|E'\|/\|E\|$
YouTube (YT)	Users and memberships in YT	94,239	30,088	293,360	59,826,175	204
BookCrossing (BC)	Books read by members in BC	105,279	340,524	1,149,739	18,551,334	16
Dbpedia-Recordlabel (DR)	Musical artists and their record labels	168,338	18,422	233,286	143,714,073	616
Dbpedia-Team (DT)	Athletes and their teams	901,167	34,462	1,366,466	325,294,100	238
GitHub (GH)	Users and projects in GH	56,520	120,868	440,237	22,280,573	51
IMDB (IM)	Movie-actor network	685,569	186,415	2,715,604	38,491,926	14
Trackers (TR)	Internet domains and trackers in them	27,665,731	12,756,245	140,613,762	-	-
Orkut (OR)	Users' group memberships in OR	2,783,197	8,730,858	327,037,487	-	-
Syn-1	Total 2^{18} vertices	199898	199847	598809	22443504	37
Syn-2	Total 2^{19} vertices	399878	399945	1198037	59703228	50
Syn-3	Total 2^{20} vertices	599910	599857	1798186	77266016	43
Syn-4	Total 2^{21} vertices	799878	799939	2398543	128957174	54
Syn-5	Total 2^{22} vertices	999822	999809	2997665	186608617	62

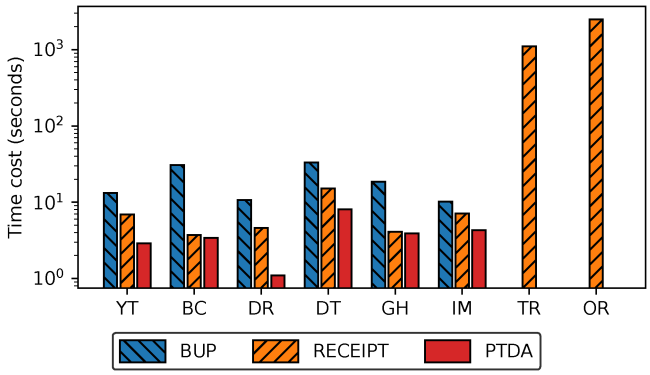

Fig. 6. Time cost of BUP, RECEIPT, and PTDA for decomposing real-world bipartite graphs.

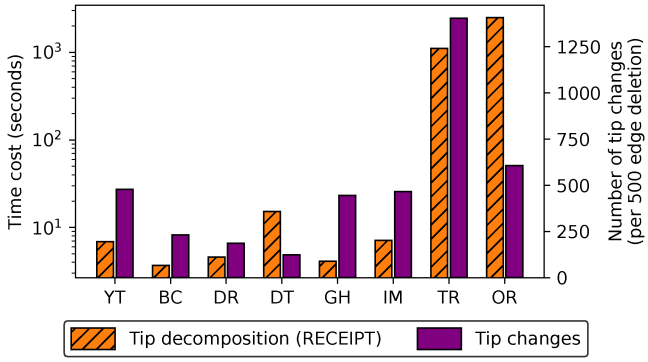

Fig. 7. The time cost of RECEIPT (left y-axis) and the number of vertices that have tip numbers changed after deleting 500 edges (right y-axis).

we also generate five synthetic bipartite graphs by the method used in [41] to verify the scalability under various graph sizes.

C. Evaluation of PTDA

Time cost overview: In this set of experiments, we compare the time cost of PTDA (Algorithm 2) with BUP [22] and RECEIPT [33] on datasets YT, BC, DR, DT, GH, and IM. This is because the generation of projection of large graphs (TR and OR) will cause the OOM (out of memory) problem, which will be analyzed later. For BUP, it is too time-consuming to be applied to maintain tip numbers in the two large-scale datasets. For each new edge, we need to apply BUP, RECEIPT, and PTDA to calculate tip numbers. The per-edge runtime cost results are shown in Fig. 6. Compared to BUP and RECEIPT, PTDA gets significant improvement in runtime cost. In specific, PTDA can speed up BUP and RECEIPT by 2.4x-9.7x and 1.1x-4.2x, respectively. This is because vertices can access their 2-hop neighbors in the projection while applying TPDA.

Defect analysis: Although PTDA is highly efficient in processing the tip maintenance problem, it introduces a considerable amount of additional memory overhead. As can be seen from Table II column *Projection* $|E'|$, the number of edges in the projection used in PTDA is orders of magnitude greater than the number of edges in the source bipartite graph. For example, there are 325,294,100 edges in the projection of dataset DT while the source bipartite graph has only 1,366,466 edges. Compared to the source bipartite graphs, the increase in the number of edges in corresponding projections is shown in the last column of Table II. Therefore, for large-scale bipartite graphs like TR and OR, storing the entire projection in DRAM is infeasible.

In addition, according to the analysis in Section III, the complexity of PTDA is $O(\sum_{e\in E}\omega(e))$, where E is the weighted edge set in the projection. The more edges in the projection, the more time will be cost by PTDA. As shown in Fig. 6, the projection of DT has the maximum weighted edges, and PTDA tasks more time to re-decompose it. This will lead to the inability to meet the real-time update requirement of tip numbers when dealing with large-scale bipartite graphs.

New expectation: According to the results of datasets YT, BC, DR, DT, GH, and IM, we find only a small part of vertices will have tip numbers changed. Then, we invoked RECEIPT to re-decompose TR and OR after deleting the 500 edges. Fig. 7 (right y-axis) presents the number of vertices that have tip numbers changed. In proportion to the number of vertices in the source bipartite graph, it is extremely small. For example, in the dataset Orkut, there are 2,783,197 vertices in U, and only 607 tip number changes. Fig. 7 (left y-axis) also provides the time cost of RECEIPT on different datasets. As the size of the graph

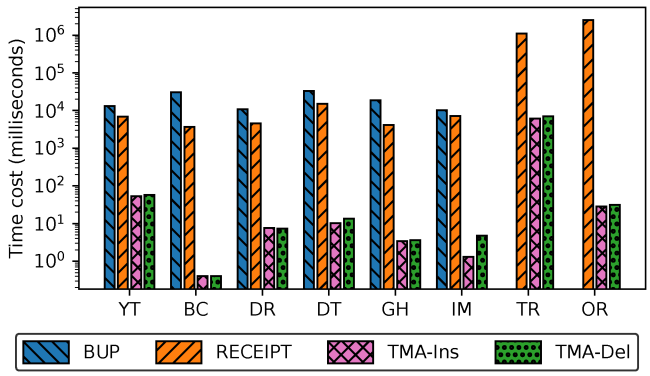

Fig. 8. Average time cost of maintaining tip numbers per edge update.

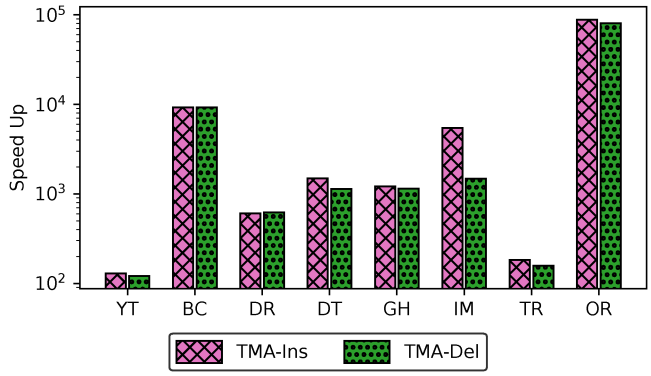

Fig. 9. Speedup of incremental insertion and removal algorithms for real-world bipartite graphs.

grows, the time consumption exhibits exponential growth. Applying RECEIPT to maintain tip numbers in dynamic bipartite graphs, particularly large-scale ones, is inefficient. Researching incremental algorithms in this context offers significant potential for improvement.

D. Evaluations of TMA-Ins and TMA-Del

Inspired by the experimental results of PTDA, we explored the incremental tip maintenance algorithm in Section IV. This set of experiments is designed to evaluate the performance of TMA-Ins (Algorithm 3) and TMA-Del (Algorithm 4) for edge insertion and edge deletion, respectively. First, we select 500 edges and delete them from the source bipartite graph. For each edge deletion, we generate a sub-edge set and apply Algorithm 4 to maintain tip numbers. Then, we insert these 500 edges back into the bipartite graph. Algorithm 3 is invoked to calculate tip numbers incrementally. The results of average per-edge processing time are presented in Fig. 8, which demonstrate millisecond-level response time achieved for the tip maintenance problem. Even for large-scale bipartite graphs, TR and OR, which cannot be handled by PTDA and RECEIPT also need thousands of seconds to decompose, TMA-Ins/TMA-Del can maintain tip numbers in 6.1/7.0 seconds for TR and 28.4/31.2 milliseconds for per-edge update, respectively.

In addition, we compare the runtime cost to that of BUP and RECEIPT. The experiment results are also shown in Fig. 8. Compared to BUP, RECEIPT tasks less time to decompose a bipartite graph. In the following, we analyze the performance of our proposed tip maintenance algorithms (TMA-Ins and TMA-Del) with RECEIPT as the benchmark. The results show that our algorithms outperform the state-of-the-art parallel tip decomposition algorithm RECEIPT. As can be seen from Fig. 9, for edge insertion, the maximum speedup reaches 87,950x. For edge deletion, the highest speedup is 80,057x. But on dataset YT, the speedup is only 128x and 121x for per-edge insertion and per-edge deletion, respectively. This indicates that the performance of our algorithms is strongly influenced by the graph structure, rather than being significantly related to the size of the graph. We discuss this in more detail in the following section.

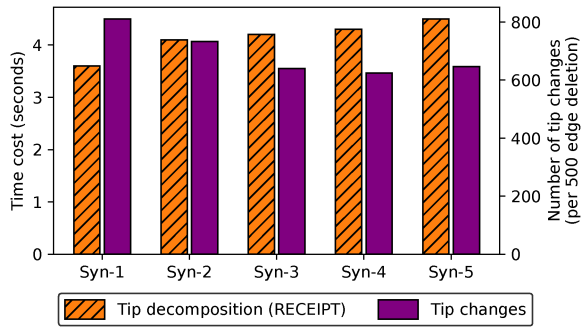

Fig. 10. The time cost of RECEIPT (left y-axis) and the number of vertices that have tip numbers changed after deleting 500 edges (right y-axis) on synthetic graphs.

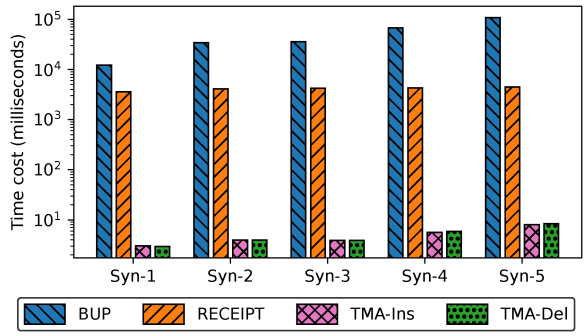

Fig. 11. Average time cost of maintaining tip numbers per edge update on synthetic graphs.

E. Evaluation on Synthetic Graphs

Based on experimental data from real datasets, the greatest change in tip values observed when inserting or deleting 500 edges accounts for 0.79% of the total number of vertices. We conduct statistical tests on five synthetic graphs to verify this proportion. In addition, we also assess the scalability of TMA-Ins and TMA-Del on these synthetic graphs. According to the experimental results shown in Figs. 10 and 11, after inserting or deleting 500 edges in synthetic datasets, the ratio of the change in tip numbers to the total vertices ranges from 0.06% to 0.4%, which are all less than 0.79%. Compared to existing tip

decomposition algorithms BUP and RECEIPT, TMA-Ins and TMA-Del achieved significant performance in maintaining the tip number of vertices in dynamic bipartite graphs of different sizes. This demonstrates that the algorithm proposed in this paper exhibits good scalability across bipartite graphs of various sizes.

F. Discussion

As mentioned earlier, the proposed algorithms exhibit significant performance variations across different datasets, and there is no positive correlation with the size of the graph. This is related to the bipartite graph structure and the execution flow of the algorithm. Whether Algorithm 3 for edge insertion or Algorithm 4 for edge deletion, they do need to construct the candidate projection Ψ by Function FindCandidate. If there are too many vertices that can be expanded as candidates, the scale of Ψ would be too large. In this context, on the one hand, it takes more time to construct the candidate projection Ψ, on the other hand, Function PeelCandidate will also cost more time to get the tip numbers. Therefore, in certain extreme cases, when the vast majority or even all vertices are candidates, our algorithm would be inefficient. This is also a problem that we plan to address in future work.

VI. Related Work

A. Butterfly Counting

A butterfly, the smallest cohesive bipartite subgraph, has been a key focus in various studies. Chiba et al. [39] introduced an efficient vertex-priority butterfly counting algorithm that operates with O(1) complexity per wedge. Wang et al. [41] then delved into optimizing the vertex-priority algorithm, prioritizing vertices based on decreasing degree. Additionally, Sanei et al. [51] proposed randomized algorithms for quickly estimating the number of butterflies in a graph with guaranteed accuracy. Further research by Sheshbolouki et al. [36], [37] explored the butterfly approximation framework within streaming bipartite graphs.

B. Cohesive Community Detection on Bipartite Graphs

Several algorithms have been designed for cohesive motif or subgraph detection on bipartite graphs [52], [53]. Ding et al. [54] proposed a linear time complexity algorithm for mining (α, β)-core. Liu et al. [55] introduced a BiCore-Index to retrieve the (α, β)-core more efficiently. Biclique is a complete subgraph, in which each pair of vertices from the different vertex sets are connected [56]. The biclique enumeration problem was explored in many recent works [57], [58], [59], [60]. In addition, existing work [22], [61], [62], [63] solved the k-bitruss decomposition problem on bipartite graphs, in which each edge is contained in at least k butterflies [64]. In particular, Sariyuce et al. [22] presented the k-tip/k-wing model and proposed the peeling-based algorithms to calculate the tip number/wing number for each vertex/edge. k-tip is a fundamental structure in mining cohesive subgraphs in bipartite graphs. In this work, we focus on tip decomposition in dynamic bipartite graphs.

k-tip is the maximal butterfly-connected subgraph where each vertex can participate in at least k butterflies. Wang et al. [65] proposed the algorithm for butterfly counting over vertices in $O(\sum_{v \in V} deg(v)^2)$ time. The approximate counting algorithms based on sampling and graph sparsification are presented in [35], [51]. Zou [66] developed the first algorithm for butterfly peeling per edge to calculate tip numbers of vertices. Shi et al. [40] designed a framework called PARBUTTERFLY that contains new parallel algorithms for global butterfly counting, tip decomposition, and wing decomposition. Lakhotia et al. [33] exploited the massive parallelism across different levels of k-tip hierarchy to further improve the tip decomposition efficiency.

However, these existing works didn't focus on tip maintenance problems on dynamic bipartite graphs. Applying the global tip decomposition algorithm to re-peel the whole graph for each edge update would be time-consuming.

C. Core Decomposition on Unipartite Graphs

Many techniques have been explored to mine hierarchical dense communities in unipartite graphs, such as k-core decomposition and k-truss decomposition [67], [68], [69]. k -core is the most fundamental type of subgraph structure, which can be detected in polynomial time. To maintain core numbers efficiently, Sariyuce et al. [45] first proposed streaming algorithms to maintain core numbers in dynamic unipartite graphs. In [70], [71], [72], parallel algorithms were presented for batch edge updates. These algorithms are designed based on a preliminary condition that the new edge will change the degree of each vertex by at most 1, and the core number of all vertices will also change at most 1. However, as discussed in Section I, a new edge will introduce an uncertain quantity of butterflies into the source bipartite graph and more than one pair of vertices will experience changes in sharing butterflies. As a result, existing core maintenance techniques cannot be applied to solve the tip maintenance problems in bipartite graphs.

In summary, the existing tip decomposition algorithms cannot calculate tip numbers efficiently and the proposed incremental core maintenance techniques cannot be applied to solve the tip maintenance problem directly. In this context, it is desirable to explore incremental algorithms to maintain tip numbers in bipartite graphs.

VII. Conclusion

In this paper, we have introduced incremental algorithms for tip decomposition on dynamic bipartite graphs. We first designed a projection-based tip decomposition algorithm to avoid 2-hop neighbor access operations. It can improve the tip decomposition while graphs evolve with time, but the extra space may lead to an OOM problem. Then, we investigated incremental algorithms that can be constructed on the source bipartite graph directly. Several theoretical studies have been proposed and proven. Based on this, we designed tip maintenance algorithms TMA-Ins and TMA-Del for edge insertion and edge deletion, respectively. Our experimental evaluation shows that these incremental algorithms can perform significantly better than the state-of-the-art tip decomposition algorithm. We believe the proposed theorems

and algorithms will serve as fundamental tools for other dynamic bipartite graph analysis problems.

Acknowledgment

The authors would like to thank anonymous reviewers for their kind suggestions.

References

[1] H. Zha, X. He, C. H. Q. Ding, M. Gu, and H. D. Simon, "Bipartite graph partitioning and data clustering," in *Proc. Int. Conf. Inf. Knowl. Manage.*, 2001, pp. 25–32.
[2] X. He, M. Gao, M. Kan, and D. Wang, "BiRank: Towards ranking on bipartite graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 1, pp. 57–71, Jan. 2017.
[3] H. Deng, M. R. Lyu, and I. King, "A generalized co-hits algorithm and its application to bipartite graphs," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2009, pp. 239–248.
[4] Ö. Kart, E. Hayirci, A. Kut, and Z. Işik, "Supervised link prediction developed for bipartite social networks," in *Proc. 3rd Int. Conf. Adv. Artif. Intell.*, 2019, pp. 14–17.
[5] K. Wang, W. Zhang, X. Lin, L. Qin, and A. Zhou, "Efficient personalized maximum biclique search," in *Proc. IEEE 38th Int. Conf. Data Eng.*, 2022, pp. 498–511.
[6] Z. Huang, D. D. Zeng, and H. Chen, "Analyzing consumer-product graphs: Empirical findings and applications in recommender systems," *Manag. Sci.*, vol. 53, no. 7, pp. 1146–1164, 2007.
[7] V. W. Zheng et al., "Heterogeneous embedding propagation for large-scale e-commerce user alignment," in *Proc. IEEE Int. Conf. Data Mining*, Singapore, 2018, pp. 1434–1439, doi: 10.1109/ICDM.2018.00198.
[8] G. A. Pavlopoulos, P. I. Kontou, A. Pavlopoulou, C. Bouyioukos, E. Markou, and P. G. Bagos, "Bipartite graphs in systems biology and medicine: A survey of methods and applications," *GigaScience*, vol. 7, no. 4, 2018, Art. no. giy014.
[9] Q. Fan, D. Zhang, H. Wu, and K. Tan, "A general and parallel platform for mining co-movement patterns over large-scale trajectories," in *Proc. VLDB Endowment*, vol. 10, no. 4, pp. 313–324, 2016.
[10] Z. Fang, Y. Gao, L. Pan, L. Chen, X. Miao, and C. S. Jensen, "CoMing: A real-time co-movement mining system for streaming trajectories," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2777–2780.
[11] S. Helmi and F. B. Kashani, "Multiscale frequent co-movement pattern mining," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 829–840.
[12] D. Sarma, U. K. Bera, and A. Das, "A mathematical model for resource allocation in emergency situations with the co-operation of NGOs under uncertainty," *Comput. Ind. Eng.*, vol. 137, 2019, Art. no. 106000.
[13] A. Lamiable and J. Tomasik, "Spatial frequency reuse in a novel generation of PMR networks," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2013, pp. 1410–1415.
[14] K. Li et al., "Towards controlling the transmission of diseases: Continuous exposure discovery over massive-scale moving objects," in *Proc. Int. Joint Conf. Artif. Intell.*, 2022, pp. 3891–3897.
[15] N. E. H. B. Chaabene, A. Bouzeghoub, R. Guetari, and H. H. B. Ghézala, "Applying machine learning models for detecting and predicting militant terrorists behaviour in Twitter," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, 2021, pp. 309–314.
[16] N. G. Miloslavskaya, A. Nikiforov, K. Plaksiy, and A. Tolstoy, "Applying graph theory to detect cases of money laundering and terrorism financing," in *Handbook of Research on Advanced Applications of Graph Theory in Modern Society*, Hershey, PA, USA: IGI Global, 2020, pp. 297–319.
[17] S. Whitelaw, M. A. Mamas, E. Topol, and H. G. Van Spall, "Applications of digital technology in COVID-19 pandemic planning and response," *Lancet Digit. Health*, vol. 2, no. 8, pp. e435–e440, 2020.
[18] S. Ribeiro-Navarrete, J. R. Saura, and D. Palacios-Marqués, "Towards a new era of mass data collection: Assessing pandemic surveillance technologies to preserve user privacy," *Technological Forecasting Social Change*, vol. 167, 2021, Art. no. 120681.
[19] A. Sheshbolouki and M. T. Özsu, "sGrow: Explaining the scale-invariant strength assortativity of streaming butterflies," *ACM Trans. Web*, vol. 17, 2022, Art. no. 24.
[20] Q. Zhu, J. Zheng, H. Yang, C. Chen, X. Wang, and Y. Zhang, "Hurricane in bipartite graphs: The lethal nodes of butterflies," in *Proc. 32nd Int. Conf. Sci. Statist. Database Manage.*, 2020, Art. no. 18.
[21] J. A. Acosta, T. M. Low, and D. Parikh, "Families of butterfly counting algorithms for bipartite graphs," in *Proc. 2022 IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2022, pp. 304–313.
[22] A. E. Sariyüce and A. Pinar, "Peeling bipartite networks for dense subgraph discovery," in *Proc. 11th ACM Int. Conf. Web Search Data Mining*, 2018, pp. 504–512.
[23] A. S. Krishnan, Y. Yang, and P. Schaumont, "Risk and architecture factors in digital exposure notification," in *Proc. 20th Int. Conf. Embedded Comput. Syst.: Architectures Model. Simul.*, Samos, Greece, Springer, 2020, pp. 308–319.
[24] H. H. Lee and T. Lee, "The tracetogether matrix has you–surveillance, rationalisation and tactics of governance in Singapore's COVID-19 app," *Platform: J. Media Commun.*, vol. 9, no. 2, pp. 77–91, 2022.
[25] S. H. S. Lai, C. Q. Y. Tang, A. Kurup, and G. Thevendran, "The experience of contact tracing in Singapore in the control of COVID-19: Highlighting the use of digital technology," *Int. Orthopaedics*, vol. 45, pp. 65–69, 2020.
[26] U. Gasser, M. Ienca, J. Scheibner, J. Sleigh, and E. Vayena, "Digital tools against COVID-19: Taxonomy, ethical challenges, and navigation aid," *Lancet. Digit. Health*, vol. 2, pp. e425–e434, 2020.
[27] J. L. Herrera-Diestra and L. Meyers, "Local risk perception enhances epidemic control," *PLoS One*, vol. 14, 2018, Art. no. e0225576.
[28] A. Aiello, A. Renson, and P. Zivich, "Social media- and internet-based disease surveillance for public health," *Annu. Rev. Public Health*, vol. 41, pp. 101–118, 2020.
[29] W. Guo, Y. Li, M. Sha, B. He, X. Xiao, and K. Tan, "GPU-accelerated subgraph enumeration on partitioned graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1067–1082.
[30] M. Sha, Y. Li, B. He, and K. Tan, "Accelerating dynamic graph analytics on GPUs," in *Proc. VLDB Endowment*, vol. 11, no. 1, pp. 107–120, 2017. [Online]. Available: http://www.vldb.org/pvldb/vol11/p107-sha.pdf
[31] M. Sha, Y. Li, and K. Tan, "GPU-based graph traversal on compressed graphs," in *Proc. 2019 Int. Conf. Manage. Data*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, Eds., Amsterdam, The Netherlands, 2019, pp. 775–792, doi: 10.1145/3299869.3319871.
[32] M. Sha, Y. Li, and K. Tan, "Self-adaptive graph traversal on GPUs," in *Proc. Int. Conf. Manage. Data*, China, 2021, pp. 1558–1570, doi: 10.1145/3448016.3457279.
[33] K. Lakhotia, R. Kannan, V. K. Prasanna, and C. A. F. D. Rose, "RECEIPT: Refine coarse-grained independent tasks for parallel tip decomposition of bipartite graphs," in *Proc. VLDB Endowment*, vol. 14, no. 3, pp. 404–417, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol14/p404-lakhotia.pdf
[34] W. Guo, Y. Li, M. Sha, and K. Tan, "Parallel personalized PageRank on dynamic graphs," in *Proc. VLDB Endowment*, vol. 11, no. 1, pp. 93–106, 2017. [Online]. Available: http://www.vldb.org/pvldb/vol11/p93-guo.pdf
[35] R. Li et al., "Approximately counting butterflies in large bipartite graph streams," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 12, pp. 5621–5635, Dec. 2022, doi: 10.1109/TKDE.2021.3062987.
[36] S. Sanei-Mehri, Y. Zhang, A. E. Sariyüce, and S. Tirthapura, "FLEET: Butterfly estimation from a bipartite graph stream," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, W. Zhu et al., Eds., Beijing, China, 2019, pp. 1201–1210, doi: 10.1145/3357384.3357983.
[37] A. Sheshbolouki and M. T. Özsu, "sGrapp: Butterfly approximation in streaming graphs," *ACM Trans. Knowl. Discov. Data*, vol. 16, no. 4, pp. 76:1–76:43, 2022, doi: 10.1145/3495011.
[38] Q. Lyu, M. Sha, B. Gong, and K. Lyu, "Accelerating depth-first traversal by graph ordering," in *Proc. 33rd Int. Conf. Sci. Statist. Database Manage.*, Tampa, FL, USA, Q. Zhu, X. Zhu, Y. Tu, Z. Xu, and A. Kumar, Eds., 2021, pp. 13–24, doi: 10.1145/3468791.3468796.
[39] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM J. Comput.*, vol. 14, no. 1, pp. 210–223, 1985.
[40] J. Shi and J. Shun, "Parallel algorithms for butterfly computations," in *Massive Graph Analytics*, Boca Raton, FL, USA: Chapman and Hall/CRC, 2022, pp. 287–330.
[41] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Vertex priority based butterfly counting for large-scale bipartite networks," in *Proc. VLDB Endowment*, vol. 12, pp. 1139–1152, 2019.
[42] V. Batagelj and M. Zaversnik, "An O(m) algorithm for cores decomposition of networks," 2003, *arXiv:0310049*.
[43] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, "K-core decomposition of large networks on a single PC," in *Proc. VLDB Endowment*, vol. 9, no. 1, pp. 13–23, 2015.

[44] R.-H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2453–2465, Oct. 2014.
[45] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek, "Streaming algorithms for k-core decomposition," in *Proc. VLDB Endowment*, vol. 6, no. 6, pp. 433–444, 2013.
[46] T. Weng, X. Zhou, K. Li, P. Peng, and K. Li, "Efficient distributed approaches to core maintenance on large dynamic graphs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 1, pp. 129–143, Jan. 2022.
[47] M. Ley, "The DBLP computer science bibliography: Evolution, research issues, perspectives," in *Proc. Int. Symp. String Process. Inf. Retrieval*, Springer, 2002, pp. 1–10.
[48] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. 7th ACM SIGCOMM Conf. Internet Meas.*, 2007, pp. 29–42.
[49] S. Schelter et al., "On the ubiquity of web tracking: Insights from a billion-page web crawl," *J. Web Sci.*, vol. 4, pp. 53–66, 2018.
[50] J. Kunegis, "KONECT: The Koblenz network collection," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 1343–1350.
[51] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura, "Butterfly counting in bipartite networks," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 2150–2159.
[52] K. Wang, W. Zhang, Y. Zhang, L. Qin, and Y. Zhang, "Discovering significant communities on bipartite graphs: An index-based approach," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 3, pp. 2471–2485, Mar. 2023.
[53] K. Wang et al., "Cohesive subgraph discovery over uncertain bipartite graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 11, pp. 11165–11179, Nov. 2023.
[54] D. Ding, H. Li, Z. Huang, and N. Mamoulis, "Efficient fault-tolerant group recommendation using alpha-beta-core," in *Proc. 2017 ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 2047–2050.
[55] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation: An index-based approach," in *Proc. World Wide Web Conf.*, 2019, pp. 1130–1141.
[56] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum biclique search at billion scale," in *Proc. VLDB Endowment*, vol. 13, pp. 1359–1372, 2020.
[57] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient maximal biclique enumeration for large sparse bipartite graphs," in *Proc. VLDB Endowment*, vol. 15, no. 8, pp. 1559–1571, 2022.
[58] J. Wang, J. Yang, Z. Ma, C. Zhang, S. Yang, and W. Zhang, "Efficient maximal biclique enumeration on large uncertain bipartite graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12634–12648, Dec. 2023.
[59] Y. Zhang, C. A. Phillips, G. L. Rogers, E. J. Baker, E. J. Chesler, and M. A. Langston, "On finding bicliques in bipartite graphs: A novel algorithm and its application to the integration of diverse biological data types," *BMC Bioinf.*, vol. 15, pp. 1–18, 2014.
[60] Y. Zhao, Z. Chen, C. Chen, X. Wang, X. Lin, and W. Zhang, "Finding the maximum k-balanced biclique on weighted bipartite graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 8, pp. 7994–8007, Aug., 2023.
[61] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Efficient bitruss decomposition for large-scale bipartite graphs," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 661–672.
[62] Y. Wang, R. Xu, X. Jian, A. Zhou, and L. Chen, "Towards distributed bitruss decomposition on bipartite graphs," in *Proc. VLDB Endowment*, vol. 15, no. 9, pp. 1889–1901, 2022.
[63] Z. Zou, "Bitruss decomposition of bipartite graphs," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, Springer, 2016, pp. 218–233.
[64] Y. Yang, Y. Fang, M. E. Orlowska, W. Zhang, and X. Lin, "Efficient bi-triangle counting for large bipartite networks," in *Proc. VLDB Endowment*, vol. 14, no. 6, pp. 984–996, 2021.
[65] J. Wang, A. W.-C. Fu, and J. Cheng, "Rectangle counting in large bipartite graphs," in *Proc. 2014 IEEE Int. Congr. Big Data*, 2014, pp. 17–24.
[66] R. Zhu, Z. Zou, and J. Li, "Fast rectangle counting on massive networks," in *Proc. 2018 IEEE Int. Conf. Data Mining*, 2018, pp. 847–856.
[67] F. Bonchi, A. Khan, and L. Severini, "Distance-generalized core decomposition," in *Proc. 2019 Int. Conf. Manage. Data*, 2019, pp. 1006–1023.
[68] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proc. 31st Int. Conf. Very Large Data Bases*, 2005, pp. 721–732.
[69] J. Wang and J. Cheng, "Truss decomposition in massive networks," 2012, *arXiv:1205.6693*.
[70] Q.-S. Hua et al., "Faster parallel core maintenance algorithms in dynamic graphs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1287–1300, Jun. 2020.
[71] H. Jin, N. Wang, D. Yu, Q.-S. Hua, X. Shi, and X. Xie, "Core maintenance in dynamic graphs: A parallel approach based on matching," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2416–2428, Nov. 2018.
[72] N. Wang, D. Yu, H. Jin, C. Qian, X. Xie, and Q.-S. Hua, "Parallel algorithm for core maintenance in dynamic graphs," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2366–2371.

Tongfeng Weng received the master's degree from the School of Information Engineering, Wuhan University of Technology, in 2018, and the doctoral's degree from the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. He is a research fellow with the School of Computing, National University of Singapore. His research interests include graph computing and parallel computing.

Yumeng Liu received the doctor's degree from the University of Chinese Academy of Sciences, in 2023. He is a senior engineer with the Institute of Software, Chinese Academy of Sciences. His research interest covers data mining and database technology.

Mo Sha received the PhD degree from the School of Computing, National University of Singapore, in 2020. He is currently a research scientist with Apsara Lab, Alibaba Cloud. His research focuses on leveraging emerging and modern hardware to enhance performance, security, and scalability in data management systems.

Xinyuan Chen received the bachelor's degree from the College of Automation Engineering, Nanjing University of Aeronautics and Astronautics, China, in 2024. He is currently working toward the PhD degree with Zhejiang University. His research focuses on algorithm analysis.

Xu Zhou received the master's degree from the College of Computer Science and Electronic Engineering, Hunan University, in 2009. She is currently an associate professor with the Department of Information Science and Engineering, Hunan University, Changsha, China. Her research interests include parallel computing and data management.

Kenli Li (Senior Member, IEEE) received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He is currently a Cheung Kong professor of computer science and technology with Hunan University, and the vice-principal of Hunan University. His major research interests include parallel and distributed processing. He serves on the editorial board of the *IEEE Transactions on Computers*.

Kian-Lee Tan (Senior Member, IEEE) received the PhD degree in computer science from the National University of Singapore, Singapore, in 1994. He is the dean of the School of Computing, National University of Singapore (NUS), Singapore. His current research interests include query processing and optimization, database performance, data science, and distributed graph computing. He is a member of ACM.