Efficient Biclique Counting in Large Bipartite Graphs

XIAOWEI YE, RONG-HUA LI, QIANGQIANG DAI, HONGCHAO QIN, and
GUOREN WANG, Beijing Institute of Technology, China

A (p,q)-biclique is a complete subgraph (X,Y) that $|X| = p, |Y| = q$. Counting (p,q)-bicliques in bipartite graphs is an important operator for many bipartite graph analysis applications. However, getting the count of (p,q)-bicliques for large p and q (e.g., $p,q \geq 10$) is extremely difficult, because the number of (p,q)-bicliques increases exponentially with respect to p and q. The state-of-the-art algorithm for this problem is based on the (p,q)-biclique enumeration technique which is often costly due to the exponential blowup in the enumeration space of (p,q)-bicliques. To overcome this problem, we first propose a novel exact algorithm, called EPivoter, based on a newly-developed edge-pivoting technique. The striking feature of EPivoter is that it can count (p,q)-bicliques for all pairs of (p,q) using a combinatorial technique, instead of exhaustively enumerating all (p,q)-bicliques. Second, we propose a novel dynamic programming (DP) based h-zigzag sampling technique to provably approximate the count of the (p,q)-bicliques for all pairs of (p,q), where an h-zigzag is an ordered simple path in G with length $2h-1$ ($h = \min\{p,q\}$). We show that our DP-based sampling technique is very efficient. Third, to further improve the efficiency, we also propose a hybrid framework that integrates both the exact EPivoter algorithm and sampling-based algorithms. Extensive experiments on 7 real-world graphs show that our algorithms are several orders of magnitude faster than the state-of-the-art algorithm.

CCS Concepts: • **Networks → Network algorithms**; • **Mathematics of computing → Probabilistic algorithms**.

Additional Key Words and Phrases: biclique counting, large bipartite graphs, graph sampling

ACM Reference Format:
Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongchao Qin, and
Guoren Wang. 2023. Efficient Biclique Counting in Large Bipartite Graphs. *Proc. ACM Manag. Data* 1, 1, Article 78 (May 2023), 26 pages. https://doi.org/10.1145/3588932

1 INTRODUCTION

Given a bipartite graph $G(U,V,E)$ which comprises two disjoint vertex sets U and V and an edge set $E \subseteq U \times V$. A (p,q)-biclique in G is a complete subgraph $C(L,R)$ of G with $|L| = p$, $|R| = q$, and $\forall(u,v) \in L \times R, (u,v) \in E$. In this paper, we focus on the problem of counting (p,q)-bicliques for every pairs of p and q. Counting the bicliques in a bipartite graph is a fundamental operator for many higher-order bipartite graph analysis applications. We give two concrete examples as follows.

Higher-order clustering coefficient. The higher-order clustering coefficient based on traditional k-clique [37, 38] is an important metric to analyze the statistical properties of complex networks. It was shown in [37] that networks from the same domain often have similar higher-order clustering coefficient characteristics. As indicated in [37, 38], such a higher-order clustering coefficient can be easily extended to bipartite graphs. Specifically, in bipartite graphs, the higher-order clustering

Authors' address: Xiaowei Ye, yexiaowei@bit.edu.cn; Rong-Hua Li, lironghuabit@126.com; Qiangqiang Dai, qiangd66@gmail.com; Hongchao Qin, qhc.neu@gmail.com;
Guoren Wang, wanggrbit@gmail.com, Beijing Institute of Technology, Beijing, China.

2836-6573/2023/5-ART78 $15.00
https://doi.org/10.1145/3588932

coefficient can be defined as the ratio between the counts of (p, q)-bicliques and (p, q)-*wedges*, where a (p, q)-wedge $G(U_W, V_W, E_W)$ is a connected non-induced subgraph that contains a $(p-1, q)$-biclique (or $(p, q-1)$-biclique) and one additional vertex $u \in U_W$ (or $v \in V_W$) which connects at least one vertex in V_W (or U_W). Such a higher-order clustering coefficient measures the probability of a (p, q)-*wedges* becoming a (p, q)-biclique, and it can characterize the internal nature of the bipartite graph data (as confirmed in our experiments). Since the count of (p, q)-wedges can be efficiently computed from the count of (p, q)-bicliques (as shown in Section 8), the key to compute the higher-order clustering coefficient in bipartite graph is to count the (p, q)-bicliques.

Higher-order densest subgraph mining. Finding the densest subgraph from a graph is a fundamental graph ming operator. Recent studies focus mainly on mining higher-order densest subgraph based on k-cliques on traditional graphs [11, 26, 28] and based on (p, q)-bicliques on bipartite graphs [21], because such a higher-order densest subgraph is often a quasi-clique [21, 28] which is very useful for network analysis applications. In bipartite graphs, the (p, q)-biclique densest subgraph is a subgraph with the maximum (p, q)-biclique density, which is defined as the ratio between the count of (p, q)-bicliques in a subgraph S and the number of vertices in S [21]. The exact algorithm to compute the (p, q)-biclique densest subgraph is based on a parametric max-flow procedure, which is often intractable for large bipartite graphs [21]. To obtain a practical solution, we can develop a *peeling* algorithm (as used in [11, 28] for traditional graphs) by iteratively removing the vertex that has the minimum (p, q)-biclique count. Clearly, such a peeling algorithm needs to frequently count the number of (p, q)-bicliques of each vertex. Thus, an efficient approach to count the (p, q)-bicliques is crucial for mining the (p, q)-biclique densest subgraph in bipartite graphs.

Despite of the practical importance of (p, q)-biclique counting, we still lack efficient algorithms to count all bicliques in large bipartite graphs, due to the intrinsic hardness of the biclique counting problem. Indeed, real-world bipartite graphs often contain a huge number of (p, q)-bicliques even for very small p and q. For example, Fig. 1 shows the number of bicliques contained in 7 real-world bipartite graphs given that $p = 4$. As can be seen, the number of bicliques with $p = 4$ can be more than 10^{39} in a medium-sized graph Twitter ($|U| = 175,214$, $|V| = 530,418$, and $|E| = 1,890,661$). As a consequence, it is often intractable to count all (p, q)-bicliques for relatively large p and q in large graphs.

The state-of-the-art algorithm to count the (p, q)-biclique is based on a backtracking enumeration technique [34]. The basic idea of this algorithm is to maintain a sub-biclique and recursively add the vertices from the candidate set (the common neighbors of the vertices in the sub-biclique) into the sub-biclique to generate a larger biclique. The limitations of this algorithm are twofold: (1) it needs to enumerate all (p, q)-bicliques to get the (p, q)-biclique count which is very costly for relatively large p and q; and (2) it is mainly tailored for counting (p, q)-bicliques for only a pair of (p, q), and is often intractable to count the bicliques for all possible pairs of (p, q).

To overcome these limitations, we first propose a novel exact algorithm, called EPivoter, to count all bicliques in a bipartite graph G for all pairs of (p, q). A striking feature of EPivoter is that it can count all bicliques for all pairs of (p, q) without exhaustively enumerating every biclique. To achieve this, we first develop a novel edge-pivoting technique to enumerate maximal bicliques. With such a powerful edge-pivoting technique, we can uniquely represent every biclique by using a set of *large bicliques* (not necessarily maximal) which can be enumerated based on our edge-pivoting technique. As a consequence, we can count all bicliques for all pairs of (p, q) in those *large bicliques* using a combinatorial counting method, instead of exhaustively enumerating each biclique. Since enumerating those *large bicliques* is much cheaper than enumerating all bicliques, our algorithm is often tractable to handle large bipartite graphs.

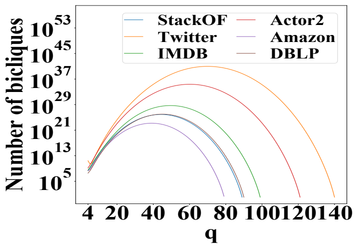
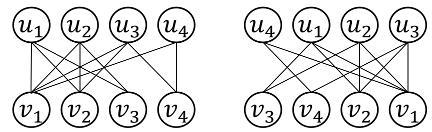


Fig. 1. The counts of (p, q)-bicliques for $p = 4$



(a) A bipartite graph (b) Degree-ordered graph

Fig. 2. A running example

To improve the efficiency, we develop two novel sampling-based algorithms, called ZigZag and ZigZag++, to estimate the counts of the bicliques for all pairs of (p, q). Both ZigZag and ZigZag++ are based on a newly-developed h-zigzag sampling technique, where an h-zigzag is a vertex-ordered simple path in G with length $2h - 1$ and $h = \min\{p, q\}$. It is easy to see that any (p, q)-biclique of G contains at least one h-zigzag, thus we can estimate the number of (p, q)-bicliques by sampling h-zigzags. To obtain uniform h-zigzag samples, we propose a dynamic programming (DP) based h-zigzag counting and sampling algorithms. We show that the time complexity of our DP-based sampling algorithm is bounded by $O(h|E|)$, thus it is very efficient when h is not very large. In addition, to further improve the sampling performance, we also propose a hybrid framework by integrating both our exact algorithm and sampling-based algorithms. The hybrid framework is based on the following observation: the sampling-based algorithms often work well in the dense region of the bipartite graph (because in the dense region, an h-zigzag is likely to be contained in a certain biclique, thus improving the sampling performance), while the exact algorithm performs very well in the sparse region of the graph. Therefore, to achieve better performance, the hybrid framework uses the exact algorithm to count bicliques in the sparse region of the graph, while leverages the sampling-based algorithms to process the dense region of the graph. We also devise an efficient and effective algorithm to partition the bipartite graph into a sparse region and a dense region. The results of extensive experiments demonstrate the high efficiency of our solutions. Below, we briefly summarize our contributions.

Novel exact algorithms. We proposed a novel EPivoter algorithm to count the (p, q)-bicliques for all pairs of (p, q). The novelties of EPivoter are twofold: (1) it is the first algorithm that can exactly count all bicliques without exhaustively enumerating each biclique; and (2) it relies on a new edge-pivoting technique that can also be used to enumerate all maximal bicliques. We believe that our edge-pivoting technique may be of independent interest.

Novel approximation algorithms. We develop two novel approximation algorithms ZigZag and ZigZag++ to estimate the counts of all bicliques for all pairs of (p, q). The novelties of our two approximation algorithms lie in that (1) both of them are based on a new and efficient DP-based h-zigzag sampling technique, and (2) both of them are the first algorithm that can provably approximate the number of (p, q)-bicliques for all pairs of (p, q). In addition, to further improve the accuracy of the approximation algorithms, we also present a hybrid framework by integrating our exact algorithm and approximation algorithms based on a carefully-designed graph partition technique.

Extensive experiments. We conduct extensive experiments on 7 real-life bipartite graphs to evaluate our algorithms. The results show that (1) our exact algorithm is very efficient to count all bicliques and it is more than two orders magnitude faster than the state-of-the-art (SOTA) algorithm. For example, on Actor2 ($|U| = 303,617$, $|V| = 896,302$, and $|E| = 3,782,463$), EPivoter takes only 49 seconds to count all bicliques for all pairs of (p, q), while the SOTA algorithm consumes 12,476 seconds.(2) All our approximation algorithms are not only very efficient but also pretty accurate to estimate the counts of (p, q)-bicliques with $h = \min\{p, q\} \leq 10$. For example, on Actor2, our best approximation algorithm takes only 15 seconds, while the SOTA algorithm uses 9,015 seconds. Moreover, the average estimator error of our best approximation algorithm can be lower than 0.7% on Actor2.(3) Even when estimating the count of the (p, q)-bicliques for only one pair of (p, q), our exact and approximation algorithms are still much faster than the SOTA algorithm for relatively large p and q (e.g., $p = 8$ and $q = 8$). In addition, we also conduct two application experiments including the fast computation of higher-order clustering coefficient on bipartite graphs and computing the (p, q)-biclique densest subgraph by a newly-developed peeling-based $\frac{1}{p+q}$-approximate algorithm. The experiment results demonstrate the effectiveness of our biclique counting techniques. We release our source code at https://github.com/LightWant/biclique.

2 PRELIMINARIES

Let $G = (U, V, E)$ be a bipartite graph, where U and V are two set of vertices and $E = \{e(u, v) | u \in U, v \in V\}$ denotes the set of edges. For each vertex u in U (or V), its neighbors are $N(u, G) = \{v | e(u, v) \in E\}$. For a vertex set S, the set of the common neighbors of S is defined as $N(S, G)$, i.e., $N(S, G) = \cap_{u \in S} N(u, G)$. Let $d(u, G)$ be the degree of u in G, i.e. $d(u, G) = |N(u, G)|$. If the context is clear, $N(u, G)$, $N(S, G)$ and $d(u, G)$ are abbreviated as $N(u)$, $N(S)$ and $d(u)$, respectively. In addition, we also define the set of neighbors of an edge $e(u, v) \in E$ denoted by $N(e(u, v), G)$ as $N(e(u, v), G) \triangleq \{e(u', v') \in E | u' \in N(v) \setminus \{u\}, v' \in N(u) \setminus \{v\}\}$.

Denote by $\{u_1, u_2, ..., u_{n_1}\}$ ($\{v_1, v_2, ..., v_{n_2}\}$) a non-decreasing ordering of vertices in U (V) with respect to (w.r.t.) the vertices' degrees (break ties by vertex IDs), where $n_1 = |U|$ ($n_2 = |V|$). For convenience, we refer to such an ordering as a degree ordering $\prec_d$. With such a degree ordering $\prec_d$, we have $d(u_i) \leq d(u_j)$ ($d(v_i) \leq d(v_j)$) for each u_i and u_j (v_i and v_j) if $u_i \prec_d u_j$ ($v_i \prec_d v_j$). Then, based on $\prec_d$, we define the set of *ordering neighbors* of v w.r.t. a reference vertex u_i as $N^{>u_i}(v, G) \triangleq \{u_j | e(u_j, v) \in E, u_i \prec_d u_j\}$, which includes the neighbors of v in G with ranks higher than u_i's rank according to the degree ordering. Based on these definitions, we further define the set of ordering neighbors of an edge $e(u_i, v_j)$ as $\vec{N}(e(u_i, v_j)) \triangleq \{e(u_{i'}, v_{j'}) \in E | u_{i'} \in N^{>u_i}(v_j), v_{j'} \in N^{>v_j}(u_i)\}$.

Definition 2.1. Given a bipartite graph $G(U, V, E)$, a biclique in G is a complete subgraph with a pair of vertex sets (X, Y) where $X \subseteq U, Y \subseteq V$ and $\forall u \in X, \forall v \in Y, e(u, v) \in E$.

A biclique is maximal if no vertex can be added into it to generate a larger biclique. A (p, q)-biclique (X, Y) is a biclique with $|X| = p$ and $|Y| = q$. In this paper, we investigate two problems on (p, q)-biclique counting: (1) the first problem is to compute the number of (p, q)-bicliques in a bipartite graph G with a given p and q; and (2) the second problem is to simultaneously count the (p, q)-bicliques in G for every pair of p and q.

As discussed in Section 1, the first problem is very hard for large p and q (e.g., $p \geq 10, q \geq 10$) due to the exponential blowup of the biclique counts. For example, in the Twitter dataset ($|U| = 175,214$, $|V| = 530,418$, $|E| = 1,890,661$), even for small p and q, the number of (p, q)-bicliques can be very large (for $p = 2$ and $q = 2$, the $(2, 2)$-biclique count is more than 2×10^8; and for $p = 5$ and $q = 5$, the $(5, 5)$-biclique count is around 1×10^{13}). Clearly, the second problem is much more difficult than the first problem, as it requires to count all (p, q)-bicliques for every pair of p and q.

3 THE PROPOSED EPIVOTER ALGORITHM

In this section, we propose a novel exact algorithm, called EPivoter, to count the (p, q)-bicliques for all p and q. The EPivoter is inspired by the PIVOTER algorithm which was originally designed to count the k-cliques in traditional graphs [13]. Specifically, PIVOTER uses the classic pivoting

Algorithm 1: EPMBCE

Input: A bipartite graph $G = (U, V, E)$
Output: All maximal bicliques in G
$C \leftarrow \emptyset$;
MBCE $(U, V, \emptyset, \emptyset)$;
return C **Procedure** MBCE(C_l, C_r, P_l, P_r)
  Denote by $G'(C_l, C_r, E')$ the subgraph of G, where $E' = \{e(u, v) \in E | u \in C_l, v \in C_r\}$;
  if $E' = \emptyset$ **then**
    if $C_l \neq \emptyset$ and $C_r \neq \emptyset$ **then** Check$(P_l \cup C_l, P_r)$; Check$(P_l, P_r \cup C_r)$;
    else Check$(P_l \cup C_l, P_r \cup C_r)$;
    return;
  $e(u, v) \leftarrow \max_{e(u,v) \in E'} |N(e(u, v), G')|$;
  if $C_l \setminus N(v) \neq \emptyset$ **then** Check$(P_l \cup C_l, P_r)$;
  if $C_r \setminus N(u) \neq \emptyset$ **then** Check$(P_l, P_r \cup C_r)$;
  Reset the IDs of vertices in C_l and C_r such that for each $u' \in C_l \setminus N(v)$, $v' \in C_r \setminus N(u)$, we have $u' < u''$ and $v' < v''$ if $u'' \in C_l \cap N(v)$ and $v'' \in C_r \cap N(u)$;
  foreach $e(u', v') \in E'$ *s.t.* $(u' \notin N(v)$ *or* $v' \notin N(u))$ **do**
    $C'_l \leftarrow C_l \cap N^{>u'}(v')$; $C'_r \leftarrow C_r \cap N^{>v'}(u')$;
    MBCE$(C'_l, C'_r, P_l \cup \{u'\}, P_l \cup \{v'\})$;
  $C'_l \leftarrow C_l \cap N(v) \setminus \{u\}$; $C'_r \leftarrow C_r \cap N(u) \setminus \{v\}$;
  MBCE$(C'_l, C'_r, P_l \cup \{u\}, P_l \cup \{v\})$;
Procedure Check(X, Y)
  if (X, Y) is maximal **then** $C \leftarrow C \cup \{(X, Y)\}$;

technique in maximal clique enumeration [7, 27] to generate a *succinct clique tree* (SCT) structure which can uniquely encode every k-clique. Based on this *SCT* structure, PIVOTER then counts all k-cliques using a combinatorial counting method, instead of exhaustively enumerating all k-cliques. However, extending the idea of PIVOTER for counting bicliques is quite non-trivial, because there is no similar pivoting technique in existing maximal clique enumeration algorithms [1, 22, 39] that can be used to construct a *SCT-style* structure. Moreover, even if we have a pivoting technique, the construction of a unique representation for every (p, q)-biclique is still very challenging, because a biclique has two sides of vertices and both of them needs to be uniquely encoded. Indeed, as we shown in Section 3.2, we need to consider 6 different and more complicated cases to encode every (p, q)-biclique, while it is sufficient to consider two simple cases to encode each k-clique in PIVOTER [13] (i.e, a k-clique either contains the pivot vertex or not).

To overcome these problems, we first develop a new maximal biclique enumeration algorithm with a carefully-designed edge-pivoting technique, which selects an edge for pivoting instead of a vertex in each recursion. Based on this edge-pivoting technique, we then propose a representation method that can uniquely encode every (p, q)-clique. Armed with such a representation approach, we are able to count the (p, q)-bicliques for all p and q using a combinatorial counting method. Below, we detail our solutions.

3.1 New maximal biclique enumeration algorithm

In this subsection, we propose a new backtracking algorithm called EPMBCE to enumerate all maximal bicliques. The novelty of EPMBCE lies in the facts that (1) in each recursion, EPMBCE selects an edge to expand the biclique, while all existing algorithms choose a vertex to expand the biclique [1, 22, 39]; and (2) EPMBCE is integrated with a newly-developed edge-pivoting technique to further reduce the search space.

Specifically, the basic idea of our algorithm is that for every maximal biclique (L, R) of G, it (1) either contains an edge $e(u, v) \in E$, or (2) does not contain $e(u, v)$. This indicates that we can recursively divide the original problem into two sub-problems: the first one is to enumerate all maximal bicliques containing $e(u,v)$, and the other one is to enumerate all maximal bicliques excluding $e(u,v)$.

However, such a basic algorithm may generate many non-maximal bicliques, which leads to redundant calculations. This is because for any non-maximal biclique, it also satisfies the property that it either contains an edge $e(u,v)$ or does not contain $e(u,v)$. To reduce non-maximal bicliques explored by the enumeration procedure, we further present a novel edge-pivoting technique, which is shown in the following theorem. Due to the space limit, we mainly give the proof sketches for our results, and the complete proofs are given in the full version of this paper [4].

Theorem 3.1. *Given a bipartite graph $G(U,V,E)$ with $|E| > 0$. If we choose an edge $e(u,v) \in E$ as the pivot edge, every maximal biclique must contain at least one edge in $\{e(u,v)\} \cup \{e(u',v') \in E | u' \notin N(v)$ or $v' \notin N(u)\}$.*

Proof sketch. The maximal bicliques that do not contain $e(u,v)$ must contain at least one edge in $\{e(u',v') \in E | u' \notin N(v)$ or $v' \notin N(u)\}$, otherwise adding $e(u,v)$ to them will generate larger bicliques. □

Equipped with Theorem 3.1, we can easily derive that for any maximal biclique in G, it either contains an edge $e(u,v)$, or contains an edge $e(u',v')$ with $u' \notin N(v)$ or $v' \notin N(u)$. Based on this, we can develop an edge-pivoting technique to further improve our basic *edge-based* enumeration algorithm. Specifically, we first select an edge $e(u,v)$ in E as a pivot, and then find all maximal bicliques in G by only enumerating the maximal bicliques containing each edge in $\{e(u,v)\} \cup \{e(u',v') \in E | u' \notin N(v)$ or $v' \notin N(u)\}$. The detailed pseudo-code is shown in Algorithm 1.

In Algorithm 1, it invokes the MBCE procedure to recursively enumerate all maximal bicliques (line 2). Specifically, MBCE admits four parameters C_l, C_r, P_l, and P_l, where ($P_l \subset U$, $P_r \subset V$) is the partial biclique, and (C_l, C_r) is the candidate set such that every vertex $u \in C_l$ ($v \in C_r$) can be added to P_l (P_r) to form a larger biclique. Let E' be the set of edges in the subgraph G' of G induced by the candidate sets (C_l, C_r) (line 4). If E' is empty (line 5), the algorithm first determines whether the candidate sets are empty, and then adds the non-empty candidate sets into the current biclique to generate a larger biclique and also checks the maximality of the generated bicliques (lines 6-7). After that, the algorithm can terminate the current recursion, as in this case no pivot can be used to further branching (line 8). If E' is non-empty, the algorithm chooses the edge $e(u,v)$ which has the maximum number of neighbors in the candidate set as the pivot edge (line 9), because such a pivot edge can prune the most number of candidates. Then, according to Theorem 3.1, each edge in $\{e(u,v)\} \cup \{e(u',v') \in E' | u' \notin N(v)$ or $v' \notin N(u)\}$ can be used to expand the current partial biclique (P_l, P_r), and the corresponding sub-recursive calls are further invoked to continue the enumeration (lines 13-15 correspond to the case of including a non-neighbor edge and lines 16-17 correspond to the case of including the pivot edge). The notation $N^{>u'}(v')$ and $N^{>v'}(u')$ in line 14 is the set of ordering neighbors $\{u_j | e(u_j, v') \in E, u' \prec u_j\}$ and $\{v_j | e(u', v_j) \in E, v' \prec v_j\}$ defined in Section 2. Note that in line 19, the algorithm can check the maximality using existing techniques in maximal biclique enumeration algorithms [1, 22, 39]. The correctness of Algorithm 1 is shown in the following theorem.

Theorem 3.2. *Algorithm 1 correctly outputs all maximal bicliques of a given bipartite graph G.*

Proof sketch. According to Theorem 3.1, (X,Y) must (1) contain e_p, or (2) at least a non-neighbor of e_p. For the case (1), line 17 in Algorithm 1 enumerates the maximal bicliques in the neighbors of e_p. For the case (2), lines 10, 11 and 15 enumerates the maximal bicliques containing non-neighbors of e_p. □

Based on the results established in [27], we can derive the time and space of Algorithm 1 as shown in the following theorem.

Algorithm 2: Unique representation of each biclique

Input: A bipartite graph $G(U, V, E)$
Output: The unique representation for each biclique

BCEUnique$(U, V, \emptyset, \emptyset, \emptyset, \emptyset)$;
Procedure BCEUnique$(C_l, C_r, \tilde{P}_l, H_l, \tilde{P}_r, H_r)$
  Denote by $G'(C_l, C_r, E')$ the subgraph of G, where $E' = \{e(u,v) \in E | u \in C_l, v \in C_r\}$;
  if $E' = \emptyset$ **then**
    if $C_l \neq \emptyset$ *and* $C_r \neq \emptyset$ **then**
      Represent$(\tilde{P}_l \cup C_l, H_l, \tilde{P}_r, H_r)$;
      for each non-empty subset set S of C_r **do** Represent$(\tilde{P}_l, H_l, \tilde{P}_r, H_r \cup S)$;
    else Represent$(\tilde{P}_l \cup C_l, H_l, \tilde{P}_r \cup C_r, H_r)$;
    return;
  $e(u,v) \leftarrow \max_{e(u,v)\in E'} |N(e(u,v), G')|$;
  Reset the IDs of vertices in C_l and C_r such that for each $u' \in C_l \cap N(v)$, $v' \in C_r \cap N(u)$, we have $u' < u''$ and $v' < v''$ if $u'' \in C_l \cap N(v)$ and $v'' \in C_r \cap N(u)$;
  foreach $e(u', v') \in E'$ *s.t.* $(u' \notin N(v)$ *or* $v' \notin N(u))$ **do**
    $C_l' \leftarrow C_l \cap N^{>u'}(v')$; $C_r' \leftarrow C_r \cap N^{>v'}(u')$;
    BCEUnique$(C_l', C_r', \tilde{P}_l, H_l \cup \{u'\}, \tilde{P}_r, H_r \cup \{v'\})$;
  $C_l' \leftarrow C_l \cap N(v) \setminus \{u\}$; $C_r' \leftarrow C_r \cap N(u) \setminus \{v\}$;
  BCEUnique$(C_l', C_r', \tilde{P}_l \cup \{u\}, H_l, \tilde{P}_r \cup \{v\}, H_r)$;
  foreach $w \in C_l \setminus N(v)$ **do**
    $C_l \leftarrow C_l \setminus \{w\}$; Represent$(\tilde{P}_l \cup C_l, H_l \cup \{w\}, \tilde{P}_r, H_r)$;
  foreach $w \in C_r \setminus N(u)$ **do**
    $C_r \leftarrow C_r \setminus \{w\}$; Represent$(\tilde{P}_l, H_l, \tilde{P}_r \cup C_r, H_r \cup \{w\})$;
Procedure Represent$(\tilde{P}_l, H_l, \tilde{P}_r, H_r)$
  foreach *subset* X *of* $\tilde{P}_l$ **do**
    foreach *subset* Y *of* $\tilde{P}_r$ **do**
      $(X \cup H_l, Y \cup H_r)$ is a unique biclique;

Theorem 3.3. *Given a bipartite graph $G(U, V, E)$, the worst-case time and space complexity of Algorithm 1 is $O(3^{|E|/3})$ and $O(|E| + |U| + |V| + d_{\max}^2)$ respectively, where $d_{\max}$ is the maximal degree among all vertices.*

Proof sketch. By applying the results established in [27], we can derive that the time complexity of our algorithm is $O(3^{|E|/3})$. The search depth is bounded by $O(d_{\max})$, and the space usage for storing (C_l, C_r) is also $O(d_{\max})$, thus the space complexity of Algorithm 1 is $O(|E| + |U| + |V| + d_{\max}^2)$. □

3.2 Unique representation for each biclique

In this subsection, we establish a unique representation for each biclique based on our edge-pivoting technique. Below, we first classify all bicliques into six categories based on the pivot edge, and then we will show how to uniquely represent all these types of bicliques based on the edge-pivoting technique.

Theorem 3.4. *Let $e_p = e(u, v)$ be the pivot edge, and $N(e_p, G)$ be the set of neighbor edges of e_p in G. Then, each biclique (including $(p, 0)$-biclique and $(0, q)$-biclique) must belong to one of the following six categories.*

(1) *It contains u and v, and the other vertices are included in the subgraph induced by $N(e_p, G)$.*
(2) *It contains u but v, and the other vertices are included in the subgraph induced by $N(e_p, G)$.*
(3) *It contains v but u, and the other vertices are included in the subgraph induced by $N(e_p, G)$.*
(4) *It does not contain u or v, while the other vertices are included in the subgraph induced by $N(e_p, G)$.*

(5) *It only contains vertices in U (or V), and contains at least one vertex in $U \setminus N(v)$ (or $V \setminus N(u)$).*
(6) *It contains at least one vertex in $U \setminus N(v)$ or $V \setminus N(u)$, and includes at least one edge in $\{e(u', v') \in E | u' \notin N(v)$ or $v' \notin N(u)\}$.*

Proof sketch. The bicliques can be initially classified into two categories. The first category contains at least one non-neighbor of e_p and the second one does not. Cases (1)-(4) belong to the first category. Cases (5)-(6) belong to the second category. Note that Case (5) only include the $(p, 0)$-bicliques and $(0, q)$-bicliques. □

Let G' be the subgraph of $G(C_l, C_r)$ induced by $N(e_p, G(C_l, C_r))$ ($G(C_l, C_r)$ is a bipartite subgraph of G induced by the candidate sets (C_l, C_r)). By Theorem 3.4, any biclique of G is either contained in $G' \cup \{e_p\}$ (cases (1)-(4)) or includes a vertex outside of $G' \cup \{e_p\}$ (cases (5)-(6)). It is easy to see that all the bicliques belonging to cases (1)-(4) must be contained in the maximal bicliques enumerated in the enumeration branch of Algorithm 1 that includes e_p (line 17 of Algorithm 1). Similarly, we can easily derive that each biclique belonging to case (6) must be contained in a maximal biclique enumerated in the enumeration branch of Algorithm 1 that includes a non-neighbor edge of e_p (lines 13-15 of Algorithm 1). Note that the bicliques belonging to case (5) are the $(p, 0)$-bicliques or $(0, q)$-bicliques. Such bicliques can be computed by using a combinatorial counting method, because in Algorithm 1, these bicliques must be included in C_l (or C_r) and contain at least one vertex in $C_l \setminus N(v)$ (or $C_l \setminus N(u)$) , where $e(u, v)$ is the pivot edge.

Based on the above analysis, a unique representation for each biclique of G can be obtained based on the enumeration tree generated by our edge-pivoting technique. Specifically, for the current partial biclique (P_l, P_r) in each recursion of Algorithm 1, we divide the vertex set P_l into two subset $\tilde{P}_l$ and H_l, where $\tilde{P}_l$ only contains the vertices of all the selected pivot edges (in the current recursion, there may exist several pivot edges that have already added into (P_l, P_r) in the previous recursions) and H_l contains the remaining vertices in P_l (i.e., $P_l = \tilde{P}_l \cup H_l$). Similarly, we partition P_r as $\tilde{P}_r$ and H_r. With these notations, we can easily distinguish the bicliques. This is because whenever a vertex in $\tilde{P}_l$ or $\tilde{P}_r$ is selected or not, it can yield a different biclique (cases (1)-(4) of Theorem 3.4). In other words, any combination of the vertices in $\tilde{P}_l$ and $\tilde{P}_r$ can generate a different biclique. Algorithm 2 gives a detailed implementation to uniquely represent the bicliques in G based on our edge-pivoting technique.

The general procedure of Algorithm 2 is similar to Algorithm 1, as both of them are based on our edge-pivoting technique. The main differences are summarized as follows. First, to maintain the current partial biclique in each recursion, Algorithm 2 requires four sets $\tilde{P}_l$, $\tilde{P}_r$, H_l and H_r as we defined before. The algorithm will add the vertices of pivot edges (or non-pivot edges) into $\tilde{P}_l$ and $\tilde{P}_r$ (or H_l and H_r) for the next recursion (line 14 and 16). Second, when there is no edges in the candidate set (C_l, C_r) (line 4), Algorithm 2 terminates the recursion and invokes the Represent procedure to uniquely represent each biclique contained in the current *enumerated biclique* by using a combinatorial technique (lines 21-24). For convenience, in Algorithm 2, we refer to the input biclique of the Represent procedure, denoted by $(\tilde{P}_l \cup H_l \cup C_l, \tilde{P}_r \cup H_r)$, as the *enumerated biclique*. The set of all these *enumerated bicliques* uniquely encode every biclique in G. Note that in line 5 of Algorithm 2, when $C_l \neq \emptyset$ and $C_r \neq \emptyset$, there are two kinds of bicliques, one included in $(\tilde{P}_l \cup H_l \cup C_l, \tilde{P}_r \cup H_r)$ and the other contained in $(\tilde{P}_l \cup H_l, \tilde{P}_r \cup H_r \cup C_r)$. Both of them can represent the bicliques that are contained in $(\tilde{P}_l \cup H_l, \tilde{P}_r \cup H_r)$. Therefore, after obtaining all bicliques included in $(\tilde{P}_l \cup H_l \cup C_l, \tilde{P}_r \cup H_r)$, the other bicliques must contain at least one vertex in C_r to avoid duplicate representation (line 7). In addition, for the bicliques containing at least one vertex in $C_l \setminus N(v)$ (or $C_r \setminus N(u)$), which corresponds to the bicliques belonging to the case (5) in Theorem 3.4, we also need to invoke Represent to uniquely represent them (lines 17-20 of Algorithm 2). The following theorem shows that every biclique in G can be uniquely represented by Algorithm 2.

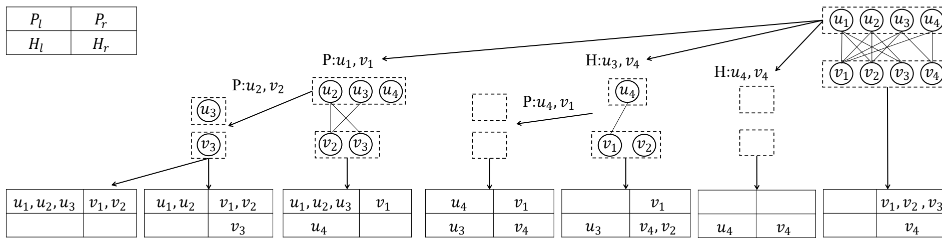


Fig. 3. Illustration of the unique representation of each biclique

Theorem 3.5. *Algorithm 2 uniquely represents every biclique in the bipartite graph G.*

Proof sketch. Since each biclique belongs to only one case among the 6 cases by Theorem 3.4, each biclique exactly belongs to one branch in the search tree of Algorithm 2. □

The following example illustrates how Algorithm 2 works.

Example 3.6. Fig. 3 shows an example of the unique representation for every biclique. The seven quadrille tables at the bottom of Fig. 3 are the 7 *enumerated bicliques* obtained by Algorithm 2, which are input into the Represent procedure to represent all bicliques. Here we illustrate how to get the first two *enumerated bicliques*. Initially, $\tilde{P}_l, \tilde{P}_r, H_l, H_r$ are empty, $C_l = \{u_1, u_2, u_3, u_4\}$ and $C_r = \{v_1, v_2, v_3, v_4\}$. First, suppose that $e(u_1, v_1)$ is selected as the pivot edge. The vertices u_1 and v_1 are added into $\tilde{P}_l$ and $\tilde{P}_r$ respectively, and the candidate set is updated to $(C_l, C_r) = (\{u_2, u_3, u_4\}, \{v_2, v_3\})$. Then, the algorithm continues the recursive calls, and another pivot edge (u_2, v_2), selected in the subgraph of G induced by (C_l, C_r), is further added into $\tilde{P}_l$ and $\tilde{P}_r$. After that, we obtain a new candidate set $(C_l, C_r) = (\{u_3\}, \{v_3\})$. In the next recursion with $(P_l, P_r) = (\{u_1, u_2\}, \{v_1, v_2\})$, there does not exist an edge in the subgraph induced by (C_l, C_r). Thus, Algorithm 2 terminates, and generates the *enumerated bicliques*. Since $C_l \neq \emptyset$ and $C_r \neq \emptyset$, the first *enumerated biclique* with $(\tilde{P}_l, \tilde{P}_r) = (\{u_1, u_2, u_3\}, \{v_1.v_2\})$ and $(H_l, H_r) = (\emptyset, \emptyset)$ is directly obtained by pushing all vertices in C_l into $\tilde{P}_l$ (line 5). Then, at least one vertex in C_r is required to add into H_r (line 7). Thus, the second *enumerated biclique* $(\tilde{P}_l, \tilde{P}_r) = (\{u_1, u_2\}, \{v_1.v_2\})$, $(H_l, H_r) = (\emptyset, \{v_3\})$ is obtained.

3.3 The EPivoter algorithm

Recall that by Theorem 3.5, each biclique of G must be contained in a uniquely *enumerated biclique* by Algorithm 2, where an *enumerated biclique* denoted by $(\tilde{P}_l \cup H_l, \tilde{P}_r \cup H_r)$ is an input biclique of the Represent procedure of Algorithm 2 (line 21). In this subsection, we propose an algorithm called EPivoter to count the (p, q)-cliques for all p and q using a combinatorial counting method based on Algorithm 2.

The EPivoter algorithm is outlined in Algorithm 3. The main framework of Algorithm 3 is basically the same as that of Algorithm 2. There are two main differences. First, when obtaining an *enumerated biclique* $(P_l \cup H_l, P_r \cup H_r)$, we count the (p, q)-bicliques contained in it using a combinatorial counting method, since each subset of P_l and P_r with size $p - |H_l|$ and $q - |H_r|$ respectively can yield a different (p, q)-biclique, i.e., the number of (p, q)-bicliques is exactly equal to $\binom{|P_l|}{p-|H_l|}\binom{|P_r|}{q-|H_r|}$ (line 7 and lines 14-15). Second, Algorithm 3 uses a degree-ordering based optimization technique to further improve the efficiency. The key idea of the degree-ordering technique is that for each *enumerated biclique*, there always exists an edge whose two end-vertices have the lowest ranks in terms of the degree ordering. That implies that every *enumerated biclique* can be identified by the enumeration algorithm that only considers the edges whose end-vertices with ranks no less than the lowest rank, instead of considering all edges in the graph. Specifically,

Algorithm 3: EPivoter

Input: A bipartite graph $G = (U, V, E)$.
Output: The count of (p, q)-bicliques ($C_{p,q}$) for every p and q in G.

1 Sort all vertices in U and V in a non-decreasing order of degree;
2 **foreach** $e(u, v) \in E$ **do**
3   BCCounting($N^{>u}(v), N^{>v}(u), \emptyset, \emptyset, \{u\}, \{v\}$);
4 **Procedure** BCCounting($C_l, C_r, P_l, P_r, H_l, H_r$)
5   **if** $E' = \emptyset$ **then**
6   **if** $C_l \neq \emptyset$ *and* $C_r \neq \emptyset$ **then**
7   Count($P_l \cup C_l, H_l, P_r, H_r$);
8   **for** $i = 1$ *to* $|C_r|$ **do**
9   /*choose i vertices of C_r to add into H_r*/
10   **foreach** $p \geq |H_l|, q \geq |H_r| + i$ **do** $C_{p,q} \leftarrow C_{p,q} + \binom{|P_l|}{p-|H_l|}\binom{|P_r|}{q-|H_r|-i}\binom{|C_r|}{i}$;
11   **else** Count($P_l \cup C_l, H_l, P_r \cup C_r, H_r$) ;
12   **return**;
13   Lines 10-22 of Algorithm 2 (replace the Represent procedure in Algorithm 2 by the Count procedure);
14 **Procedure** Count(P_l, H_l, P_r, H_r)
15   **foreach** $p \geq |H_l|, q \geq |H_r|$ **do** $C_{p,q} \leftarrow C_{p,q} + \binom{|P_l|}{p-|H_l|}\binom{|P_r|}{q-|H_r|}$;

in the top recursive call of Algorithm 3, for each edge $e(u, v)$, we only need to explore the subgraph induced by the set of ordering neighbors of $e(u, v) \in E$ (by the degree ordering) to compute the count of every biclique containing u and v, instead of exploring the whole graph G (lines 1-3). Clearly, the correctness of Algorithm 3 can be guaranteed by Theorem 3.5. The time and space complexity of Algorithm 3 is given as follows.

THEOREM 3.7. *The worst-case time and space complexity of Algorithm 3 is* $O(|E|3^{|E_{\max}|/3})$ *and* $O(|E| + |U| + |V| + d_{\max}^2)$ *respectively, where* $|E_{\max}| = \max |\vec{N}(e)|, \forall e \in E$.

Proof sketch. Algorithm 3 takes $O(3^{|\vec{N}(e)|/3})$ to process the subgraph induced by the ordering neighbors of each edge by Theorem 3.3. Thus, the total running time is $O(\sum_{e \in E} 3^{|\vec{N}(e)|/3})$. □

Note that if we only need to count the (p, q)-bicliques for a given p and q (not for all p and q), we can further improve Algorithm 3. Specifically, if the parameters satisfy one of the following conditions $|H_l| > p$, $|H_r| > q$, $|P_l \cup H_l \cup C_l| < p$ or $|P_r \cup H_r \cup C_r| < q$, we can safely terminate this recursion. Algorithm 3 can also be easily extended to count the bicliques with size constraints. We can prune the branch if $|P_l \cup H_l \cup C_l| < p$ or $|P_r \cup H_r \cup C_r| < q$ and add a preprocessing step to shrink the input graph into a (p, q)-core [19, 32].

The choice of p and q in practice. For practical applications, the values of p and q are often not very large, because small motifs are often the building blocks of large networks [20]. For example, only the count of $(2, 2)$-bicliques is useful in measuring the *Triadic Closure* of networks [29]. In application of detecting (p, q)-biclique densest subgraph [21], both p and q are often set no larger than 5. In some cases, (p, q) can be imbalanced, i.e., the value of p and q differs. For instance, [34] shows that $(5, 10)$-bicliques and $(4, 10)$-bicliques achieve the the best performance in application of optimizing the performance of graph neural network training. As a result, we recommend to set p and q no larger than 10 for many practical applications.

Discussions. The challenge of the biclique counting problem is that a (p, q)-biclique can be located in many maximal bicliques. Since the overlap relationships among the maximal bicliques are very complicated, it is extremely hard to apply the Inclusion-Exclusion technique to exactly derive the (p, q)-biclique counts from the maximal bicliques. To overcome this challenge, we propose an edge-pivoting based enumeration technique (i.e., Algorithm 3) that can uniquely encode each biclique, thus our technique can avoid repeatedly counting. Compared to the state-of-the-art algorithm which needs to exhaustively enumerate every (p, q)-biclique, Algorithm 3 only enumerates the large bicliques (i.e., the *enumerated bicliques*) and then count the (p, q)-bicliques in the *enumerated bicliques* using a combinatorial counting method. Moreover, once obtaining the *enumerated bicliques*, our algorithm can compute all biclique counts for all parameters p and q, while the state-of-the-art algorithm can only obtain one count of the (p, q)-cliques for a given (p, q) pair.

It is worth mentioning that existing motif counting techniques are not suitable for counting the bicliques because those techniques (e.g., edge-based sampling technique [2]) only work well for small motifs (e.g., the size smaller than 10). Moreover, those techniques are not tailored to biclique counting, thus they are likely worse than the existing highly-optimized biclique counting technique proposed in [34]. In addition, existing maximal biclique enumeration algorithms [1, 8, 16, 39] are also not suitable for counting bicliques because these algorithms are based on enumerating vertices of only one side. The vertex-based pivoting technique in these maximal biclique enumeration algorithms is not easily adapted to count bicliques. The reason is as follows: an edge-based pivot has neighbors on each side, thus it can easily encode vertices on both two sides. However, the vertex-based pivot is located on only one side, and it cannot encode both two sides as our edge-based pivot does. To our knowledge, our edge-pivoting based technique is novel and it has not been studied previously.

4 THE ZIG-ZAG SAMPLING ALGORITHMS

In this section, we develop several novel approximate algorithms based on a carefully-designed zig-zag sampling technique, which can obtain an unbiased estimator for all bicliques within polynomial time. Below, we give a new concept called h-zigzag which is crucial for devising our sampling algorithms. For convenience, in the rest of this paper, we assume without loss of generality that the vertices in each side of the bipartite graph $G = (U, V, E)$ are sorted in a non-decreasing ordering based on their degrees, i.e., $u_1 \prec_d u_2 \prec_d \cdots \prec_d u_{n_1}$ and $v_1 \prec_d v_2 \prec_d \cdots \prec_d v_{n_2}$.

Definition 4.1. Given a bipartite graph $G(U, V, E)$ and an integer h, a h-zigzag in G is an *ordered* simple path $P = \{u_{i_1}, v_{j_1}, u_{i_2}, v_{j_2},..., u_{i_h}, v_{j_h}\}$ that satisfies (1) its length equaling $2h - 1$, and (2) $i_1 < \cdots < i_h$ and $j_1 < \cdots < j_h$.

By Definition 4.1, an h-zigzag contains h vertices on each side of the bipartite graph, and the vertices in the h-zigzag follow the degree ordering $\prec_d$. The following lemma establishes a relationship between the h-zigzag and the (p, q)-clique.

Lemma 4.2. *Given a (p, q)-biclique of G with $p \leq q$, it exactly contains $\binom{q}{h}$ h-zigzags, if $h = p$.*

Proof sketch. It is easy to see that any h vertices selected from R will form a h-zigzag. Thus, a (p, q)-biclique contains $\binom{q}{h}$ h-zigzags. □

For convenience, in the rest of this section, we focus mainly on counting the (p, q)-bicliques with $p \leq q$, because the same proposed technique can also be used to count the (p, q)-bicliques with $p > q$. Let $\mathcal{H}$ be the set of all h-zigzags in G, and $\mathcal{T}$ be the set of uniform h-zigzags sampled from $\mathcal{H}$. Denote by Z_i the i-th h-zigzag in $\mathcal{T}$, where $1 \leq i \leq |\mathcal{T}|$. For each $Z_i \in \mathcal{T}$, the count of the (p, q)-bicliques in G that contain Z_i is denoted as $c_{p,q}(Z_i)$. Then, based on Lemma 4.2, we can derive the following lemma.

Lemma 4.3. *Let Z be an h-zigzag uniformly sampled from $\mathcal{H}$. The probability of each (p, q)-biclique of G containing Z is $\binom{q}{h}/|\mathcal{H}|$, where $h = p$.*

Proof sketch. A (p, q)-biclique contains $\binom{q}{h}$ h-zigzags (Lemma. 4.2), thus the probability is $\binom{q}{h}/|\mathcal{H}|$.□

Based on Lemma 4.3, we can construct an unbiased estimator for the count of (p,q)-biclique by uniformly sampling h-zigzag in G.

Theorem 4.4. *Denote by $\mathcal{B}_{p,q}$ the set of (p,q)-bicliques. Suppose that $\mathcal{T}$ is set of h-zigzags ($h=p$) sampled uniformly from G. Then, the unbiased estimator of the count of (p,q)-bicliques in G, denoted by $|\hat{\mathcal{B}_{p,q}}|$, is $|\hat{\mathcal{B}_{p,q}}| = \frac{|\mathcal{H}|\sum_{Z\in\mathcal{T}} c_{p,q}(Z)}{|\mathcal{T}|\binom{q}{h}}$.*

Proof sketch. We can first derive $\sum_{Z\in\mathcal{H}} c_{p,q}(Z) = \binom{q}{h}|\mathcal{B}_{p,q}|$ by definition. Then, we have $E[c_{p,q}(Z)] = \frac{\binom{q}{h}|\mathcal{B}_{p,q}|}{|\mathcal{H}|}$. Based on this, we have $E[|\hat{\mathcal{B}_{p,q}}|] = E[\frac{|\mathcal{H}|\sum_{i=1}^{|\mathcal{T}|} c_{p,q}(Z)}{|\mathcal{T}|\binom{q}{h}}] = \frac{|\mathcal{H}|}{|\mathcal{T}|\binom{q}{h}}\sum_{1=1}^{|\mathcal{T}|} E[c_{p,q}(Z)] = |B_{p,q}|$.

Armed with Theorem 4.4, we can estimate the number of (p,q)-cliques by uniformly sampling h-zigzags from G. The remaining issues are: (1) how to uniformly sample h-zigzag from the graph G; and (2) how to compute the total number of h-zigzags in G. Below, we will propose a novel and efficient dynamic programming (DP) algorithm to tackle these issues.

4.1 The DP-based zig-zag sampler

Counting the number of h-zigzags. Let $Z = \{u_1, v_1, ..., u_h, v_h\}$ be an h-zigzag in G, we have $u_i \prec_d u_{i+1}$ and $v_i \prec_d v_{i+1}$ for each $1 \le i \le h-1$. Thus, the total number of h-zigzags of G can be calculated by counting the h-zigzags starting with each vertex u_i in U. Denote the number of zig-zag paths with length i starting from the edge $e(u,v)$ by $dp[i][e(u,v)]$. The number of h-zigzags starting from u is equal to $\sum_{v\in N(u)} dp[2h-1][e(u,v)]$. As a result, the total number of h-zigzags in G is $\sum_{e(u,v)\in E} dp[2h-1][e(u,v)]$.

Interestingly, we find that we can compute $dp[i][e(u,v)]$ using a dynamic programming algorithm. The key idea of our DP algorithm is that for each zig-zag path Z of length i starting from $e(u,v)$, it must consist of the zig-zag paths of length $i-1$ starting from v. Thus, we have the following recursive equation

$$\begin{cases} dp[i][e(u,v)] = \sum_{u'\in N^{>u}(v)} dp[i-1][e(v,u')], \\ dp[i-1][e(v,u')] = \sum_{v'\in N^{>v}(u')} dp[i-2][e(u',v')], \\ dp[1][e(u,v)] = 1, \end{cases} \tag{1}$$

where $2 \le i \le 2h-1$.

Note that in Eq. (1), the notions $dp[i][e(u,v)]$ and $dp[i][e(v,u)]$ are quite different. The notion $dp[i][e(u,v)]$ corresponds to the number of paths of length of i where i is odd number, which also denotes the count of paths starting from a certain vertex in U and ending at a certain vertex in V. The notion $dp[i][e(v,u)]$ corresponds to the number of paths of even length, where both starting vertex and ending vertex are in U. When initializing $dp[1][e(u,v)]$ to 1 for all $e(u,v) \in E$, the others $dp[i][e(u,v)]$ and $dp[i][e(v,u)]$ for $i=2$ to $2h-1$ can be recursively computed by Eq. (1). The detailed implementation of this DP algorithm is outlined in Algorithm 4. It derives that the total time cost of Algorithm 4 is $O(hd_{\max}|E|)$, where $d_{\max}$ is the maximum degree in G. Such a time cost is too high for large graphs, as $d_{\max}$ can be up to $O(n)$. Below, we propose an optimization technique to reduce the time complexity to $O(h|E|)$.

Optimization technique for DPCount**.** Here we use a differential-interval updating technique to speed up Algorithm 4. Specifically, denote by $dp[i]$ the set $\{a_1, a_2, ..., a_n\}$ where $a_j = dp[i][e_j]$. We can replace $dp[i]$ with another set $\{b_1, b_2, ..., b_n\}$ such that $b_1 = a_1$ and $b_j = a_j - a_{j-1}$ for $j=2$ to n. Then, we have $a_j = \sum_{j'\le j} b_{j'}$, i.e, each a_i is the prefix sum of $\{b_1, b_2, ..., b_n\}$. Note that lines 4-5 (lines 7-8) can be regarded as updating the values of $\{a_1, a_2, ..., a_n\}$ in an interval $[l,r]$ using the same integer w ($w = dp[i-1][e(u,v)]$ in line 5 and $w = dp[i-1][e(v,u')]$ in line 8). If $dp[i]$ is replaced with $\{b_1, b_2, ..., b_n\}$, we only need to update b_l and b_{r+1} to $b_l + w$ and $b_{r+1} - w$, respectively.

Algorithm 4: DPCount(G, h)

1 Initialize $dp[i][e(u,v)] = 0$ and $dp[1][e(u,v)] = 1$ for each $e(u,v) \in E$;
2 **for** $i = 2$ **to** $2h-1$ **do**
3   **foreach** $e(u,v) \in E$ **do**
4   **foreach** $e(v,u')$ *with* $u' > u$ **do**
5             $dp[i][e(v,u')] \leftarrow dp[i][e(v,u')] + dp[i-1][e(u,v)]$;
6   **foreach** $e(v,u') \in E$ **do**
7   **foreach** $e(u',v')$ *with* $v' > v$ **do**
8             $dp[i+1][e(u',v')] \leftarrow dp[i+1][e(u',v')] + dp[i][e(v,u')]$;
9     $i \leftarrow i+2$;
10 **return** *dp;*

Algorithm 5: Sampling h-zigzags for all $h \leq h_{\max}$.

Input: A bipartite graph $G = (U, V, E)$, an array Ts and an integer $h_{\max}$.
Output: For each $h \leq h_{\max}$, uniformly sampling $Ts[h]$ h-zigzags.

Sample$(G, Ts, h_{\max}, \text{DPCount}(G, h_{\max}))$;
Procedure Sample$(G, Ts, h_{\max}, dp)$
    $zigzags[] \leftarrow \{\emptyset, \emptyset, ...\}$;
    for $h = 2$ *to* $h_{\max}$ **do**
        for $i = 1$ to $Ts[h]$ **do** $\{Z \leftarrow \text{DPSampling}(h, dp); zigzags[h] \leftarrow zigzags[h] \cup \{Z\};\}$
    return $zigzags$;
Procedure DPSampling(h, dp)
    Set the distribution D over the edges where $p(e(u,v)) = dp[2h-1][e(u,v)]/\sum_{e(u,v)} dp[2h-1][e(u,v)]$;
    Sample an edge $e(u,v)$ according to D; $Z \leftarrow \{e(u,v)\}$;
    for $i = 2h-2$ **to** $i = 2$; **do**
        $E' \leftarrow \{e(v,u') | u' \in N^{>u}(v)\}$;
        Set the distribution D over E' where $p(e) = dp[i][e]/\sum_{e \in E'} dp[i][e]$;
        Sample an edge $e(v,u')$ according to D;
        $Z \leftarrow Z \cup \{e(v,u')\}$; $E' \leftarrow \{e(u',v') | v' \in N^{>v}(u')\}$;
        Set the distribution D over E' where $p(e) = dp[i-1][e]/\sum_{e \in E'} dp[i-1][e]$;
        Sample an edge $e(u',v')$ according to D;
        $Z \leftarrow Z \cup \{e(u',v')\}$; $u \leftarrow u'$; $v \leftarrow v'$; $i \leftarrow i-2$;
    return *Z;*

After this updating, we also have $a_j = \sum_{j' \leq j} b_{j'}$. Thus, the computation of lines 4-5 (lines 7-8) can be achieved in $O(1)$ time using such a differential-interval updating trick. As a result, the total time complexity of Algorithm 4 can be reduced to $O(h|E|)$. The detailed implementation of this optimization technique can be found in our full version [4].

From counting to uniformly sampling. We propose an efficient algorithm to uniformly sample an h-zigzag based on Algorithm 4. The key point to uniformly generate an h-zigzag from G is to determine the probability distribution for all h-zigzags in G. Interestingly, we can use the counts computed by Algorithm 4 to construct the probability distribution. Specifically, by Eq. (1), the total number of h-zigzags in G is $\sum_{e(u,v) \in E} dp[2h-1][e(u,v)]$, where $dp[2h-1][e(u,v)]$ denotes the number of h-zigzags starting from the edge $e(u,v)$. Based on this, the probability of an edge $e(u,v)$ appearing in the head of an h-zigzag Z is $\frac{dp[2h-1][e]}{\sum_{e(u,v) \in E} dp[2h-1][e]}$. Moreover, for the next edge in Z, it must be contained in $\{e(v,u') | u' \in N^{>u}(v)\}$. The probability of an edge $e(v,u')$ appearing as the second edge in Z, given that $e(u,v)$ is the first edge of Z, is $\frac{dp[2h-2][e(v,u')]}{dp[2h-1][e(u,v)]}$. The same method can be recursively applied to determine the probabilities of the other edges appearing in Z. Based on these probabilities, we can uniformly sample an h-zigzag from G. The detailed implementation of our sampling algorithm is given in Algorithm 5.

In Algorithm 5, it admits two parameters $h_{\max}$ and Ts, where $h_{\max}$ limits the maximum length of h-zigzags to be sampled and Ts is the set of sample sizes of h-zigzags for each $h \leq h_{\max}$ (i.e, $Ts[h]$ denotes the sample size of the h-zigzags). First, the algorithm makes use of DPCount to get the DP table which is used to determine the sampling probabilities (line 1). For each length h, the algorithm draws $Ts[h]$ samples by invoking the DPSampling procedure (line 5). In DPSampling, it first sets a probability distribution over all edges in E based on the dp table that can be the head edge of h-zigzags (line 8). Then, the algorithm samples a head edge for an h-zigzag Z w.r.t. this probability distribution (line 9). For each remaining edge in Z, it uses a similar method to sample an edge by first setting the probability distribution of the edges using the dp table (lines 11-17). The procedure returns Z as a sampled h-zigzag if $2h-1$ edges have been added to Z (line 18). The following theorem shows that every h-zigzag can be uniformly sampled by Algorithm 5.

Theorem 4.5. *Algorithm 5 uniformly samples the set of h-zigzags in G for each* $h \leq h_{\max}$.

Proof sketch. The probability of each edge being sampled is $p(e)$ in lines 8 and 12. The probability of each h-zigzag Z being sampled is $\prod_{e \in Z} p(e) = \frac{1}{\sum_{e \in E} dp[2h-1][e]}$. □

The time and space complexity of Algorithm 5 is analyzed in the following theorem.

Theorem 4.6. *The time and space complexity of Algorithm 5 is* $O(h_{\max}|E| + h_{\max} d_{\max} |\mathcal{T}|)$ *and* $O(h_{\max}|E| + |\mathcal{T}|)$*, where* $\mathcal{T}$ *is the set of all the sampled h-zigzags for all* $h \leq h_{\max}$.

Proof sketch. The time cost of DPCount is bounded by $O(h_{\max}|E|)$. Sample one sample takes $O(h_{\max} d_{\max})$, and thus the total time costs of the algorithm is $O(h_{\max} d_{\max} |\mathcal{T}|)$. □

4.2 Biclique counts estimation via zig-zag sampling

Armed with Theorem 4.4 and Algorithm 5, we are ready to devise an estimating algorithm to approximately count the (p, q)-bicliques for all p and q.

The ZigZag **algorithm.** We develop a novel estimating algorithm which samples h-zigzags on the neighborhood subgraphs of G, instead of the whole graph. Our algorithm is based on the fact that any biclique that contains an edge $e(u, v)$ must be contained in the subgraph G' induced by the set of neighbors of $e(u, v)$ (including $e(u, v)$ itself). Therefore, we only need to sample h-zigzags on the *local neighborhood subgraph* G', instead of on the entire bipartite graph G, to estimate the counts of bicliques containing $e(u, v)$.

Specifically, for each $e(u, v)$, we first compute the subgraph G' of G induced by $\vec{N}(e(u, v))$. Then, we sample $(h-1)$-zigzags on G'; and we can obtain an h-zigzag of G by adding $e(u, v)$ to each of $(h-1)$-zigzag. To get a uniform h-zigzag sample, we also needs to take the count of the $(h-1)$-zigzags in G' into consideration. Specifically, by Eq. (1), for each $e(u, v) \in E$, the number of h-zigzags whose head edge is $e(u, v)$ is $dp[2h-1][e(u, v)]$, which indicates that the number of $(h-1)$-zigzags in G' (induced by $\vec{N}(e(u, v))$) is exactly $dp[2h-1][e(u, v)]$. Denote by T the total number of h-zigzags of G to be sampled. Then, for each G' induced by $\vec{N}(e)$, it needs to sample $T \times \frac{dp[2h-1][e]}{\sum_{e' \in E} dp[2h-1][e']}$ $(h-1)$-zigzags in G' to guarantee uniform h-zigzag samples. Clearly, the space usage of the algorithm is dominated by the maximum size of subgraphs induced by all $\vec{N}(e)$, which is often much smaller than the size of G. Moreover, each sampled h-zigzag must be contained in the *local neighborhood subgraph* which increases the probability that a sampled h-zigzag hits a biclique (i.e., the subgraph induced by the sampled h-zigzag is a biclique), thus improving the sampling performance. The detailed implementation is outlined in Algorithm 6.

Algorithm 6 uses a set of arrays ans to store the estimated counts of all (p, q)-bicliques, i.e., $ans[i][j]$ is the estimated count of (i, j)-bicliques (line 1). The algorithm then computes the counts

Algorithm 6: ZigZag: Estimating all bicliques in each edges' neighborhood subgraph

Input: A bipartite graph $G = (U, V, E)$, sample size T and an integer $h_{\max}$
Output: The approximate count of (p, q)-bicliques for all $p, q \leq h_{\max}$
1 Initialize ans as an $h_{\max} \times h_{\max}$ zeros array;
2 $G' \leftarrow$ the subgraph induced by $N^{>u}(v)$ and $N^{>v}(u)$ for each $e(u, v)$;
3 **foreach** $e(u, v) \in E$ **do**
4 $\quad dp \leftarrow DPCount(G', h_{\max} - 1)$;
5 $\quad$ **for** $h = 1$ to $h_{\max} - 1$ **do** $hzzCnt[h] \leftarrow hzzCnt[h] + \sum_{e \in \vec{N}(e(u,v))} dp[2h-1][e]$;
6 **foreach** $e(u, v) \in E$ **do**
7 $\quad dp \leftarrow$ DPCount$(G', h_{\max} - 1)$;
8 $\quad$ **for** $h = 1$ *to* $h_{\max} - 1$ **do**
9 $\quad\quad Hs[h] \leftarrow \sum_{e \in \vec{N}(e(u,v))} dp[2h-1][e]$;
10 $\quad\quad Ts[h] \leftarrow T \times Hs[h]/hzzCnt[h]$;
11 $\quad zigzags \leftarrow$ Sample$(G', Ts, h_{\max} - 1, dp)$;
12 $\quad c_{p-1,q-1}(Z) \leftarrow \binom{|N(L)|}{q-p}$ for each $Z = (L, R) \in zigzags[p-1]$ is a biclique;
13 $\quad c_{p-1,q-1}(Z) \leftarrow 0$ if $Z \in zigzags[p-1]$ does not induce a biclique;
14 $\quad ans_{p,q} \leftarrow ans_{p,q} + \frac{Hs[p-1] \sum_{Z \in zigzags[p-1]} c_{p-1,q-1}(Z)}{Ts[p-1]\binom{q-1}{p-1}}$
15 **return** ans;

of the $(h-1)$-zigzags in the subgraphs induced by each $\vec{N}(e)$, where $e \in E$ (lines 2-5). The total counts for the i-zigzags is stored in $hzzCnt[i]$ for each $1 \leq i \leq h_{\max} - 1$ (line 5). Subsequently, the algorithm determines the sample size for each h (lines 7-10) and invokes Sample to sample desired h-zigzags (line 11). For each h-zigzag sample Z, the algorithm computes $c_{p,q}(Z)$ in Theorem 4.4 by the following method. First, if Z induces a (p, p)-biclique (we assume without loss of generality that $h = p \leq q$), denoted by (L, R), then $c_{p,q}(Z) = \binom{|N(L)|}{q-p}$ (line 12).Otherwise, we have $c_{p,q}(Z) = 0$ (line 13, in this case an h-zigzag does not hit a biclique). After that, the algorithm estimates the counts of (p, q)-bicliques for all $p, q \leq h_{\max}$ based on the unbiased estimator established in Theorem 4.4 (line 14). The algorithm terminates until all *local neighborhood subgraphs* are processed.

Denote by $|E_{sum}| = \sum_{e \in E} |\vec{N}(e)|$ the total size of all subgraphs induced by $\vec{N}(e)$ for each $e \in E$. Let $|E'|$ be the maximum size among all these subgraphs. Then, the time and space complexity of Algorithm 6 is analyzed as follows.

Theorem 4.7. *The time and space complexity of Algorithm 6 is* $O(|E_{sum}|h_{\max} + |E|h_{max}^2 + Td_{\max}h_{\max}^2)$ *and* $O(h_{\max}|E'| + |E|)$ *respectively, where* T *is the total sample size for all* $h \leq h_{\max}$*, and* $d_{\max}$ *is the maximum degree of* G.

Proof sketch. By theorem 4.6, we can obtain the time and space complexity of the algorithm when handling each subgraph $\vec{N}(e)$. By summing over all these subgraphs, the total time and space complexity can be obtained. □

The ZigZag++ **algorithm.** The main limitation of Algorithm 6 is that it needs to process $O(|E|)$ neighborhood subgraphs which is often costly. To further boost the efficiency, we propose a different algorithm, called ZigZag++, that is based on sampling h-zigzags on the neighborhood subgraphs constructed by the *2-hop subgraph* of vertices.

Definition 4.8 (2-hop subgraph). Given a bipartite graph $G(U, V, E)$, we refer to $G_u(U_u, V_u, E_u)$ as the 2-hop subgraph of a vertex $u \in U$ if $V_u = N(u)$, $U_u = \cup_{v \in N(u)} N^{>u}(v)$ and $E_u = \{e(u', v') \in E | u' \in U_u, v' \in V_u\}$.

By Definition 4.8, we only need to sample h-zigzags on the 2-hop subgraphs to estimate the biclique counts for all p and q. The general steps of ZigZag++ are the same as those of Algorithm 6, except the graph construction part. Specifically, ZigZag++ first constructs the 2-hop subgraph for each vertex $u \in U$. Then, for each 2-hop subgraph, ZigZag++ computes the counts of h-zigzags contained in it using DPCount. Likewise, based on Theorem 4.4, ZigZag++ constructs an unbiased estimator using the uniform h-zigzag samples. The detailed implementation of ZigZag++ can be found in the full version [4] due to the space limit.

Let $|N_{sum}| = \sum_{u \in U} |E_u|$, where E_u is the set of edges in the 2-hop graph of u. Denote by $|E_{\max}| = \max_{u \in U} |E_u|$ is the maximum size among all 2-hop graphs. Then, similar to Theoreem 4.7, the time and space complexity of ZigZag++ is $O(|N_{sum}|h_{\max} + |V|h_{max}^2 + Td_{\max}h_{\max}^2)$ and $O(h_{\max}|E_{\max}| + |E|)$ respectively.

Comparison between ZigZag **and** ZigZag++**.** Note that ZigZag++ only needs to process $O(|U|)$ *local neighborhood subgraphs*, while ZigZag has to handle $O(|E|)$ subgraphs. Intuitively, ZigZag++ should be more efficient in terms of running time, as $|U| < |E|$.

4.3 Estimation accuracy analysis

Based on the classic Hoeffding's inequality, we can derive the estimation accuracy of our algorithms as shown in the following theorem.

THEOREM 4.9. *Denote by $\rho = \frac{|\mathcal{B}|\binom{q}{h}}{|\mathcal{H}|}$ where $p < q$ and $h = p$, $\mathcal{B}$ and $\mathcal{H}$ are the sets of (p, q)-bicliques and h-zigzags of G respectively. Let $Z = \max_{1 \le i \le T}\{c_{p,q}(Z_i)\}$, where Z_i is a sampled h-zigzag in G. Then, with probability $1 - \epsilon$, both* ZigZag *and* ZigZag++ *obtain a $1 - 2\delta$ approximation for the count of (p, q)-bicliques if $T \ge \frac{Z^2}{2\rho^2\delta^2} \ln \frac{1}{\epsilon}$, where ϵ and δ are small positive values and T is the sample size.*

Proof sketch. Since $E[\hat{\rho}] = \rho$ (by Theorem 4.4 and 4.5), we have $E[\hat{\rho}T] = \rho T$, where $\hat{\rho}T$ equals the total counts of bicliques being sampled. Then, the theorem follows by Hoeffding's inequality. □

In Theorem 4.9, we can observe that the sample size T is mainly determined by $(\frac{Z}{\rho})^2$, i.e., the larger $(\frac{Z}{\rho})^2$ is, the more samples T are required to ensure a high accuracy of our algorithms, where Z is a maximum value of $c_{p,q}(Z_i)$ for each $Z_i \in \mathcal{T}$. As shown in our experiments, $(\frac{Z}{\rho})^2$ is often small in most real-world bipartite graphs when $h_{\max}$ is not very large. Thus, the proposed algorithms generally do not require a large sample size T to achieve a good accuracy in real-world bipartite graphs as confirmed in our experiments.

Accuracy comparison between ZigZag **and** ZigZag++**.** In ZigZag, each sampled h-zigzag with the head edge $e(u, v)$ must be contained in $\vec{N}(e(u, v)) \cup \{e(u, v)\}$. However, in ZigZag++, the sampled h-zigzags with a head edge $e(u, v)$ are contained in the 2-hop graph. Intuitively, the edges in G' is closely connected to $e(u, v)$, while edges in G_u are not necessary connected to $e(u, v)$. As a result, an h-zigzag sampled from G' is more likely contained in a biclique. Therefore, ZigZag can achieve a better estimation accuracy than ZigZag++, as confirmed in our experiments. A more comprehensive analysis is shown in the full version of this paper [4].

Discussions. For the small subgraph counting problem, existing sampling-based algorithms can be classified into two categories: (1) exponential-time algorithms [2, 5, 6, 12, 14], and (2) polynomial-time algorithms [23, 33, 36]. The exponential-time algorithms either need to invoke an exponential-time procedure for sampling [5, 6, 12, 14], or need to use an exponential-time exact algorithm on the samples to compute the final results [2]. The polynomial-time algorithms are often with a theoretical guarantee based on the *pattern-hitting ratio* ρ [23, 33, 36], i.e., the ratio between the number of subgraphs (the subgraphs need to count) and the total number of *sampled patterns*. For example, for k-clique counting, [36] proposed a k-color set sampling algorithm, where the sampling performance of this algorithm depends on the ratio between the number of k-cliques and the number of k-color sets. Clearly, our h-zigzag sampling algorithms also belong to the second category, and the theoretical guarantees of our algorithms (Theorem 4.9) are very similar to that of [36]. To our knowledge, for the subgraph counting problem, no polynomial-time sampling algorithm with a theoretical guarantee independent of ρ is known.

5 A HYBRID COUNTING ALGORITHM

Intuitively, the proposed sampling-based approximation algorithms often work well when the bipartite graph is dense. This is because in a denser bipartite graph an h-zigzag is more likely to be contained in a biclique. However, for very sparse bipartite graphs, our exact EPivoter algorithm performs well as the number of *enumerated bicliques* is often not very large. These intuitions motivate us to develop a hybrid algorithm by integrating both exact and approximation algorithms. The key idea of our hybrid algorithm is that we use the exact algorithm to compute the biclique counts only in the *sparse regions* of the bipartite graph, while apply the sampling-based algorithm to estimate the biclique counts in the *dense regions* of the bipartite graph. The remaining question is how can we partition a bipartite graph into sparse regions and dense regions?

A heuristic bipartite graph partition strategy. Before introducing our heuristic partition strategy, we first give a definition as follows.

Definition 5.1. For each edge $e(u,v)$ in G, we define the weight of $e(u,v)$ as $w(e(u,v)) = |\vec{N}^{>u}(v)| \times |\vec{N}^{>v}(u)|$. For each vertex $u \in U$, we define the weight of u as $w(u) = \sum_{v \in N(u)} w(e(u,v))$.

Based on Definition 5.1, we present a heuristic approach to split the original bipartite graph into a sparse region and a dense region. Given a threshold τ, if $w(u) > \tau$, u will be added into the dense region, otherwise it is pushed into the sparse region. We can devise a peeling algorithm to compute the weight $w(u)$ for each vertex u. The detailed implementation is shown in Algorithm 7. Suppose the input bipartite graph is a degree ordered bipartite graph. The algorithm first initializes an array *deg* to maintain the degrees of vertices in G. Then, for each vertex $u \in U$, it computes the $w(u)$ by Definition 5.1 (lines 4-6). If $w(u) \leq \tau$, the vertex u is pushed into the sparse region, otherwise it is added into dense region (line 7). Note that *deg* always maintains the degree of each vertex in the remaining graph of G. Thus, for each $e(u,v)$, $deg(u) \times deg(v)$ is exactly equal to $w(e(u,v))$. It is easy to show that both the time and space complexity of Algorithm 7 are $O(|E|)$.

The hybrid algorithm for counting bicliques. After obtaining the sparse region S and the dense region D of G, the hybrid algorithm invokes the exact EPivoter algorithm to compute the subgraph induced by $\{e(u,v)|u \in S\}$ and invokes the ZigZag or ZigZag++ algorithm to estimate the counts of all bicliques in the subgraph induced by $\{e(u,v)|u \in D\}$. Based on the degree ordering, we can ensure that each biclique is counted once by the hybrid algorithm. For implementation details, we only need to modify the line 2 in Algorithm 3 by adding a constraint $u \in S$ in the "for" loop, and modify the line 3 and line 6 in Algorithm 6 by adding a constraint $u \in D$ in the "for" loop (similar process for the ZigZag++ algorithm).

Remark. Note that when we only need to count (p,q)-bicliques for a given pair (p,q), we can also use the state-of-the-art (p,q)-biclique algorithm [34] to count the (p,q)-bicliques in the subgraph induced by the edges $\{e(u,v)|u \in S\}$ (i.e., the spare region) if both p and q are not very large, because it is often very fast for small p and q given that the bipartite graph is very sparse.

6 APPLICATIONS OF BICLIQUE COUNTING

In this section, we show two applications of our (p,q)-biclique counting techniques.

Higher-order clustering coefficient. The higher-order clustering coefficient based on the k-clique counts [37, 38] is an important measurement that provides a comprehensive view of complex

Algorithm 7: Heuristic partition of the bipartite graph

Input: Degree ordered bipartite $G = (U, V, E)$ and a threshold τ
Output: The dense and sparse regions of G

```
Initialize an array deg with deg(v) = d(v) for each v ∈ U ∪ V;
foreach u ∈ U do
    w(u) ← 0;
    foreach v ∈ N(u) do
        deg(v) ← deg(v) − 1; deg(u) ← deg(u) − 1 ;
        w(u) ← w(u) + deg(v) × deg(u) ;
    if w(u) > τ then D ← D ∪ {u} else S ← S ∪ {u};
return D, S;
```

networks. As shown in [37], networks from the same domain typically have similar higher-order clustering coefficient characteristics. According to the definition in [37], we can easily extend such a higher-order clustering coefficient to bipartite graphs based on the (p, q)-biclique counts. Specifically, the higher-order clustering coefficient of a bipartite graph $hcc_{p,q}$ can be defined as $hcc_{p,q} \triangleq 2pq\frac{C_{p,q}}{W_{p,q}}$, where $C_{p,q}$ is (p, q)-biclique counts and $W_{p,q}$ is the count of *(p, q)-wedges*. Here a *(p, q)-wedge* is a connected non-induced subgraph which contains a $(p-1, q)$-biclique as the core and one extra node on the left side or a $(p, q-1)$-biclique as the core and one extra node on the right side. The extra node connects at least one node on the other side. The (p, q)-biclique is a special kind of (p, q)-wedge, namely *closed (p, q)-wedge*. The value of $hcc_{p,q}$ ranges from 0 to 1, which indicates the probability of a (p, q)-wedge being closed.

Note that we can compute $hcc_{p,q}$ by our biclique counting algorithms. This is because the value of $W_{p,q}$ depends on $C_{p-1,q}$ and $C_{p,q-1}$. Let v be a vertex on the left side. Denote the count of (p, q)-wedges containing v by $W_v(p, q)$ and the count of (p, q)-bicliques containing v by $C_v(p, q)$. We can derive that $W_v(p, q) = C_v(p, q-1)(|N_v| - q + 1)$. More specifically, we can slightly modify the Count procedure in Algorithm 3 to count $W_v(p, q)$. If v is a vertex in P_l of the Count procedure, $W_v(p, q)$ is $\binom{|P_l|-1}{p-|H_l|-1}\binom{|P_r|}{q-1-|H_r|}(|N_v| - q - 1)$. Similar formulations can be derived if v is in H_l, P_r or H_r of the Count procedure. Finally, we can obtain $W_{p,q} = \sum_{v \in U \cup V} W_v(p, q)$ which is the total counts of (p, q)-wedges.

Higher-order densest subgraph mining. Finding higher-order densest subgraph based on k-cliques on traditional graphs [11, 26, 28] and based on (p, q)-bicliques on bipartite graphs [21] is an important operator for many graph analysis applications. In bipartite graphs, the (p, q)-biclique densest subgraph problem aims to identify a subgraph S with the maximum (p, q)-biclique density (denoted by $\gamma(S)$), which is defined as the ratio between the count of (p, q)-bicliques and the number of vertices in the subgraph. The exact algorithm to find the (p, q)-biclique densest subgraph is based on a parametric max-flow procedure which is often intractable for large bipartite graphs [21].

To overcome this issue, we can devise a *peeling* algorithm based on our (p, q)-biclique counting techniques, by extending the existing peeling algorithm for the k-clique densest subgraph problem [11, 28]. Specifically, our peeling algorithm iteratively removes the vertex that has the minimum (p, q)-biclique counts and record the (p, q)-biclique density of the subgraph generated in each iteration. The algorithm outputs the subgraph with the maximum (p, q)-biclique density during the peeling procedure.

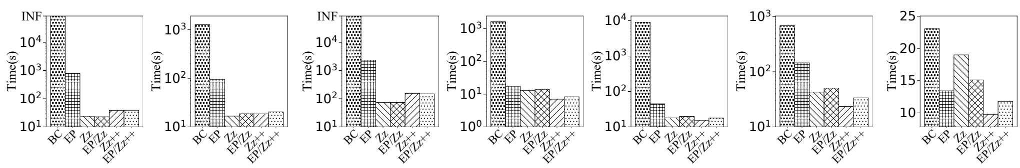
Note that we can slightly modify the Count procedure in Algorithm 3 to compute the (p, q)-biclique count for each vertex (also called local count). Specifically, if v is a vertex in P_l, then the count of (p, q)-bicliques containing node v is $\binom{|P_l|-1}{p-|H_l|-1}\binom{|P_r|}{q-|H_r|}$. If v is in H_l, the count is $\binom{|P_l|}{p-|H_l|}\binom{|P_r|}{q-|H_r|}$. Similar formulations can also be derived if v is in P_r or H_r. The following theorem shows that

Table 1. Datasets

Datasets	$\|U\|$	$\|V\|$	$\|E\|$	$\overline{d_U}$	$\overline{d_V}$
Github	56,519	120,867	440,237	7.8	3.6
StackOF	545,195	96,678	1,301,942	2.4	13.5
Twitter	175,214	530,418	1,890,661	10.8	3.6
IMDB	685,568	186,414	2,715,604	4.0	14.6
Actor2	303,617	896,302	3,782,463	12.5	4.2
Amazon	2,146,057	1,230,915	5,743,258	2.7	4.7
DBLP	1,953,085	5,624,219	12,282,059	6.3	2.2

Table 2. Comparison of the sampling algorithms on DBLP (Time (seconds), Error (%))

Algorithms	$(2,5)$		$(5,5)$		all $p=q<10$		all $(<10,<10)$	
	Time	**Error**	**Time**	**Error**	**Time**	**Error**	**Time**	**Error**
Zz	15.02	0.01	21.81	0.02	15.28	0.06	17.93	0.14
Zz++	7.14	0.03	26.13	1.76	8.65	0.07	9.80	0.49
EP/Zz	12.40	0.00	8.01	0.00	13.54	0.06	15.66	0.15
EP/Zz++	8.79	0.02	8.80	0.22	10.58	0.06	12.17	0.17
PSA	INF	-	9.79	15.01	50.73	12.52	INF	-



(a) Github (b) StackOF (c) Twitter (d) IMDB (e) Actor2 (f) Amazon (g) DBLP

Fig. 4. Runtime of different biclique counting algorithms for all $p, q \le h_{\max}$ ($h_{\max} = 10, T = 10^5$).

our *peeling* algorithm can achieve a $\frac{1}{p+q}$-approximation for the (p,q)-biclique densest subgraph problem.

Theorem 6.1. *Denote the (p,q)-biclique densest subgraph by S^*. The peeling algorithm returns a $\frac{1}{p+q}$-approximate answer $\tilde{S}$, i.e. $\gamma(\tilde{S}) \ge \frac{\gamma(S^*)}{p+q}$.*

Proof sketch. Since S^* is the (p,q)-biclique densest subgraph, we have $\gamma(S^*) \ge \gamma(S^* \setminus \{v\})$. Denote by $c(S^*)$ the total counts of S^* and $c_v(S^*)$ is the local count of v in S^*. Then, we have $\frac{c(S^*)}{|S^*|} \ge \frac{c(S^*)-c_v(S^*)}{|S^*|-1}$. Further, we can derive that $c_v(S^*) \ge \frac{c(S^*)}{|S^*|} = \gamma(S^*)$. Let v be the vertex satisfying $v \in S^*$ and $\forall u \in S^*, u \in S_v$. Then, we have $S^* \subset S_v$ and $c_v(S_v) \ge c_v(S^*)$. After that, we have

$$\gamma(S_v) = \frac{c(S_v)}{|S_v|} = \frac{\sum_{u \in S_v} c_u(S_v)}{(p+q)|S_v|} \ge \frac{\sum_{u \in S_v} c_u(S^*)}{(p+q)|S_v|} \ge \frac{\sum_{u \in S_v} \gamma(S^*)}{(p+q)|S_v|} = \frac{\gamma(S^*)}{p+q}. \tag{2}$$

7 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the proposed algorithms. For exact biclique counting, we implement the EP algorithm which is our edge-pivot based algorithm proposed in Algorithm 3. We use the state-of-the-art algorithm [34], denoted by BC, as the baseline algorithm. Since BC can only count the (p,q)-bicliques for a specific pair of (p,q), we run BC multiple times by varying p and q to get the counts of (p,q)-bicliques for all pairs of (p,q). In the experiments, we will also compare our algorithms with BC to count (p,q)-cliques with only one pair of (p,q), although we focus mainly on counting all (p,q)-bicliques for all pairs of (p,q).

For approximate biclique counting, we implement five different algorithms: Zz, Zz++, EP/Zz, EP/Zz++ and PSA [2]. Zz and Zz++ are the proposed ZigZag and ZigZag++ algorithms developed in Section 4. EP/Zz and EP/Zz++ are hybrid algorithms integrating EP. All these approximate algorithms have two parameters: the maximum h-zigzag size $h_{\max}$ and the sample size T. In our experiments, we set $h_{\max} = 10$ and $T = 10^5$ as default values. PSA is a general subgraph counting algorithm, which first samples a set of edges using a priority sampling technique and then enumerates the (p,q)-bicliques in the graph induced by the set of sampled edges using the state-of-the-art BC algorithm [34]. We use the count of $(2,2)$-biclique as the weight of edge for priority sampling as suggested in [2] and set the sample size as $T \times h_{max}$ for a fair comparison. Unless otherwise specified, when varying one parameter, the other parameters are set to their default

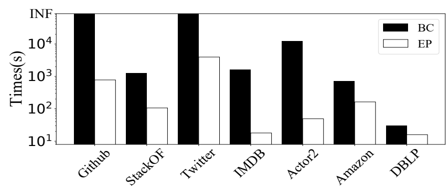
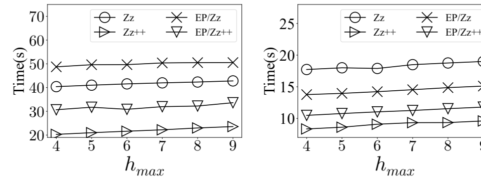
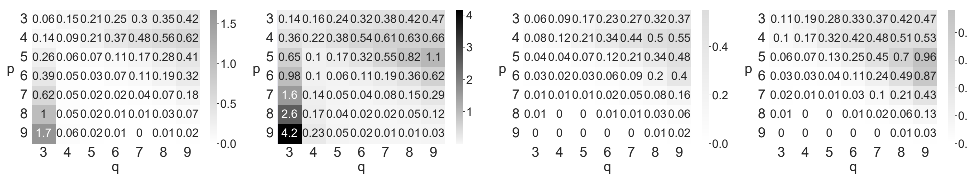


Fig. 5. Runtime of different exact biclique counting algorithms for all pairs of (p, q)



(a) Amazon (b) DBLP

Fig. 6. Runtime of different algorithms (vary $h_{\max}$)



(a) DBLP, Zz (b) DBLP, Zz++ (c) DBLP, EP/Zz (d) DBLP, EP/Zz++

Fig. 7. The heat-map of estimation errors of various algorithms with varying p and q (%).

value. We will also study how these parameters affect the performance of different algorithms. For all approximate algorithms, the results in the experiments are the average results over 20 runs. All algorithms are implemented in C++. We evaluate all algorithms on a server with an AMD 3990X CPU and 256GB memory running Linux CentOS 7 operating system.

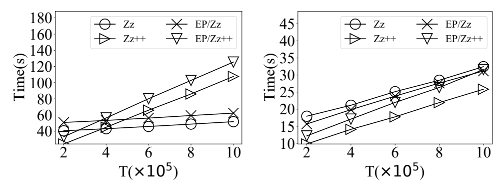
We use 7 large real-world bipartite graphs in our experiments. The detailed statistics of our datasets are summarized in Table 1.All datasets are downloaded from http://konect.cc/.

7.1 Performance results

Runtime of different algorithms. We first compare two exact algorithms EP and BC to count all bicliques for all pairs of (p, q). We use the symbol "INF" to represent that the algorithm cannot terminate within 24 hours. The results on all datasets are shown in Fig. 5. As can be seen, EP can be up to more than two orders of magnitude faster than BC on all datasets. For example, on Actor2, BC takes 12,476 seconds, while EP only consumes 49 seconds to count all bicliques.

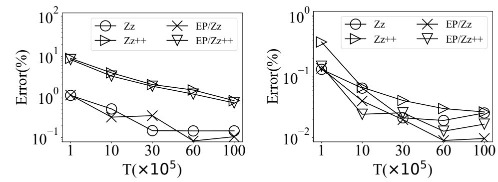
Second, we compare the runtime of all algorithms given that $h_{\max} = \min\{p, q\} \leq 10$. We set $h_{\max} = 10$ based on the following reasons. First, small biclique counts may be more useful for real-world applications [34]. Second, the baseline algorithm BC is often intractable when $h_{\max}$ is large. Third, our sampling-based algorithms may be not very accurate for a very large $h_{\max}$. As a result, it is more useful to compare the performance of different algorithms when $h_{\max}$ is not very large. Fig. 4 shows the results on all datasets. As can be seen, all our algorithms are substantially faster than the state-of-the-art algorithm BC to count the (p, q)-bicliques for all $p, q \leq h_{\max}$ on all datasets. When comparing Zz and Zz++, we find that on large graphs Zz++ is faster than Zz, while on small graphs Zz is more efficient. In general, our four sampling-based algorithms are significantly faster than our exact algorithm. For example, on Actor2, EP takes 45 seconds, while Zz, Zz++, EPivoter/ZigZag, and EPivoter/ZigZag++ consumes 18 seconds, 15 seconds, 20 seconds, and 18 seconds respectively. These results suggest that our sampling-based algorithms are very efficient to estimate the (p, q)-bicliques for all $p, q \leq h_{\max}$ given that $h_{\max}$ is not very large.

Comparison of our sampling algorithms with PSA. Here we compare the performance of our sampling algorithms with the priority sampling based algorithm PSA. The results on DBLP is shown in Table 2. Similar results can also be observed on the other datasets. As can be seen, our



(a) Amazon (b) DBLP

Fig. 8. Runtime of different algorithms (vary T)



(a) Amazon (b) DBLP

Fig. 9. Estimation errors of different algorithms (vary T)

sampling algorithms significantly outperform PSA in terms of both running time and estimation accuracy. When $p = q$, PSA achieves 15.01% error to count the $(5, 5)$-biclique and 12.52% average error for all $p = q < 10$, using 9.79 seconds and 50.73 seconds respectively. Our best algorithm, however, achieves near 0% error to count $(5, 5)$-biclique and 0.06% average error for all $p = q < 10$, using 8.01 seconds and 10.58 seconds respectively. When counting $(2, 5)$-biclique, PSA cannot terminate within 1 day. This is because there are 3×10^{11} $(2, 5)$-bicliques in the sampled graph and it is very costly to enumerate all of them using BC. Generally, the count of imbalanced biclique is very large (as shown in Fig. 1) even in the subgraph induced by the set of sampled edges. For the same reason, PSA cannot count (p, q)-bicliques for all $p < 10, q < 10$. The results further demonstrates the efficiency and superiority of our algorithms.

The effect of the parameter $h_{\max}$. Fig. 6 shows the runtime of four sampling-based algorithms with varying $h_{\max}$ on Amazon and DBLP, where the sample size T is set to 10^5. The results on the other datasets are consistent. As expected, the runtime of all sampling-based algorithms slightly increases with $h_{\max}$ increases. This is because the time complexity of sampling-based algorithms is insensitive w.r.t. the parameter $h_{\max}$ if $h_{\max}$ is small. Moreover, we can see that Zz++ is much faster than Zz with all parameters, which confirms our analysis in Section 4.2.

We define the estimation error as $\frac{|C_{p,q} - \hat{C_{p,q}}|}{C_{p,q}}$. Fig. 7 plots the heat-maps of the estimation errors of different algorithms with varying p and q on DBLP. From Fig. 7, we can see that all our sampling-based algorithms are very accurate if p and q are small (i.e., $p \leq 8$ and $q \leq 8$). Moreover, Zz is generally more accurate than Zz++ under most parameter settings, which is consistent with our analysis in Section 4.3. In general, with the increase of p or q, the estimation errors of all sampling-based algorithms also increase. This is because with $h_{\max} = \min\{p, q\}$ increases, the probability of an h-zigzag containing in a biclique will decrease, thus reducing the sampling performance.

The effect of the parameter T. Here we study the performance of our approximate algorithms with a varying sample size T. Fig. 8(b) shows the runtime of four approximate algorithms with varying T on Amazon and DBLP. The results on the other datasets are consistent. As expected, the runtime of all algorithms increases as T increases. Also, we can observe that Zz++ is significantly faster than the other algorithms, which is consistent with our previous results.

Fig. 9 shows the average errors of our approximate algorithms with varying T on Amazon and DBLP ($h_{\max} = 10$). As shown in Fig. 9, the estimation errors of all algorithms decrease as T increases. In general, Zz has a lower error than Zz++, which further confirms our analysis in Section 4.3. Additionally, we can clearly see that EP/Zz (EP/Zz++) achieves a lower error than Zz (Zz++). This result further indicates that our hybrid framework can improve the estimation accuracy of the sampling-based algorithms.

Runtime of counting (p, q)-bicliques for fixed p and q. Table 3 reports the runtime of various algorithms for specific p and q on Github. The general results on other datasets are consistent. As

Table 3. Runtime of different (p, q)-biclique counting algorithm for only a pair of (p, q) on Github ($p < 10, q < 10, T = 10^5$)

(p, q)	BC	EP	Zz	Zz++	EP/Zz	EP/Zz++
(2,3)	1.48	232.59	12.65	7.43	12.66	7.62
(2,8)	0.35	230.71	12.64	7.43	12.68	7.68
(4,5)	992.63	445.69	17.25	1.97	17.12	2.00
(4,8)	155.84	452.51	14.38	4.78	14.38	4.79
(6,4)	148.17	491.63	17.51	2.27	17.41	2.31
(6,7)	13114.24	460.14	18.96	2.42	18.81	2.49
(9,4)	56.77	457.24	17.48	2.27	17.43	2.35
(9,9)	3679.47	480.95	20.70	3.04	20.54	3.13

Table 4. The value of $\frac{Z^2}{\rho^2}$ with varying p, q (Amazon)

(p, q)	Zz	Zz++	EP/Zz	EP/Zz++
(2, 3)	3.45e+02	1.08e+02	2.94e+02	**7.69e+01**
(2, 8)	1.65e+04	5.15e+03	1.40e+04	**3.68e+03**
(4, 5)	2.52e+03	6.01e+04	**2.51e+03**	5.98e+04
(4, 8)	1.04e+04	2.49e+05	**1.04e+04**	2.48e+05
(6, 4)	2.44e+03	5.82e+04	**2.43e+03**	5.80e+04
(6, 7)	2.13e+04	5.32e+07	**2.13e+04**	5.32e+07
(9, 4)	2.24e+04	5.34e+05	**2.23e+04**	5.32e+05
(9, 9)	3.32e+04	8.01e+10	**3.32e+04**	8.01e+10

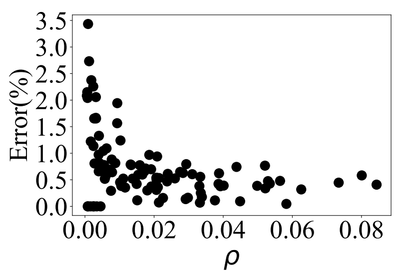


Fig. 10. Estimation errors of Zz with various ρ.

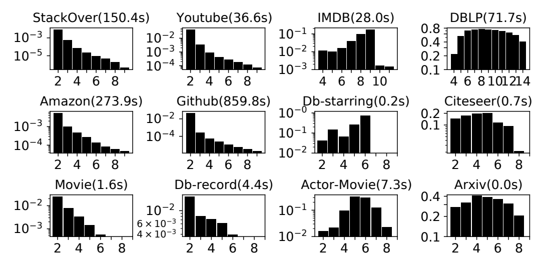


Fig. 11. The y-axis is the value of $hcc_{p,q}$. The x-axis is the value of p and q. The three networks in each column are from the same domain and they share similar properties.

can be seen, all our algorithms are substantially faster than BC. In general, the running time of EP is relatively stable when varying p and q, and EP also significantly outperforms BC when both p and q are relatively large (e.g., $p = q = 8$). We can also observe that our sampling-based algorithms are generally much faster than both EP and BC for relatively-large p and q. Zz++ achieves the lowest runtime which is up to three orders of magnitude faster than the baseline algorithm BC. For example, when $p = q = 9$, Zz++ only takes 3.04 seconds to count the $(9, 9)$-bicliques, while BC consumes 3679.47 seconds. These results further confirm the high efficiency of the proposed algorithms.

The value of $\frac{Z^2}{\rho^2}$. Theorem 4.9 shows that the sample size T depends on $\frac{Z^2}{\rho^2}$, where $Z = \max_{X \in H} c(X)$ and $\rho = \binom{\max\{p,q\}}{\min\{p,q\}}|\mathcal{B}|/|\mathcal{H}|$. Since it is very costly to compute $c(X)$ for each h-zigzag X, we sample a set of h-zigzags $\mathcal{H}'$ and use $\hat{Z} = max_{X \in \mathcal{H}'} c(X)$ as an estimation of Z. Table 4 reports the results of $\frac{Z^2}{\rho^2}$ on Amazon with varying p and q. As can be seen, the value of $\frac{Z^2}{\rho^2}$ in Zz is smaller than that of Zz++, especially when p and q is large. When integrating our exact counting technique, the value of $\frac{Z^2}{\rho^2}$ in each sampling-based algorithm decreases for all p and q, which confirms the analysis in Section 5.

Estimation error of Zz with various ρ. Here we study the accuracy of our Zz algorithm with various ρ. To this end, we first generate 100 random bipartite graphs with various edge densities by the classic Erdos-Renyi random graph model. Then, we use Zz with $T = 10^6$ to estimate the (p, q)-biclique counts on these random bipartite graphs. Fig. 10 shows a scatter-plot on estimation error of $(4, 4)$-biclique versus ρ. The results for estimating other small (p, q)-bicliques are consistent. From Fig. 10, we can see that our algorithm performs very well even when ρ is very small. For

Table 5. Results of the graph partition strategy

Networks	Sparse		Dense	
	$\|S\|$	$(2,2)$	$\|D\|$	$(2,2)$
Github	48,169	5.09e+05	8,350	5.04e+07
StackOF	533,949	8.62e+05	11,246	1.74e+07
Twitter	92,985	2.01e+06	82,229	2.04e+08
IMDB	654,744	1.81e+06	30,824	9.99e+06
Actor2	224,383	3.46e+06	79,234	1.96e+07
Amazon	2,108,102	2.71e+06	37,955	3.31e+07
DBLP	1,929,341	1.47e+07	23,744	1.70e+07

Table 6. Results of finding the (p, q)-biclique densest subgraph

Networks	(p,q)	$C_{p,q}$	Time (s)		Density	
			peeling	exact	peeling	exact
Dbpedia	$(2,2)$	1.1e+06	6.7e+01	1.8e+02	249.33	249.63
Dbpedia	$(5,5)$	2.9e+03	6.6e-01	1.6e+00	105.37	105.37
IMDB	$(2,2)$	1.2e+07	7.2e+01	4.2e+03	2638.01	2638.01
IMDB	$(5,5)$	1.8e+09	1.4e+01	INF	1.4e+07	-
Amazon	$(2,2)$	3.6e+07	6.0e+02	3.5e+04	2383.19	2383.42
Amazon	$(5,5)$	6.6e+08	1.0e+02	INF	4.7e+06	-
DBLP	$(2,2)$	3.2e+07	7.9e+03	INF	6.6e+02	-
DBLP	$(5,5)$	1.3e+08	6.1e+00	INF	2.2e+06	-

example, even when $\rho \leq 0.02$, the estimation errors of Zz are still less than 3.5% on all random bipartite graphs. This result further confirms the high efficiency of our algorithm.

Evaluation of the graph partition strategy. Table 5 shows the size of the sparse region and dense region divided by Algorithm 7, as well as the count of $(2, 2)$-bicliques in each region on all datasets. The counts of (p, q)-bicliques for other p and q are consistent. From Table 5, we can see that the sparse region occupies a large part of the entire graph, but it contains a very small portion of $(2, 2)$-bicliques on most datasets. The results indicate that our heuristic graph partition strategy is indeed very effective for partitioning the bipartite graph into sparse and dense regions.

7.2 Applications of (p, q)-biclique counting

Higher-order clustering coefficient. In this experiment, we use 12 real-life datasets (downloaded from http://konect.cc/) selected from four different domains. We plot the $hcc_{p,q}$ and the total running time for all pairs of (p, q) with $p = q < 10$ in Fig. 11. We can see that the higher-order clustering coefficient of the datasets in the same domain have similar distributions, which is also consistent with the results in [37] for traditional graphs. The $hcc_{p,q}$ in the first three columns varies by orders of magnitude with varying p and q. However, the changes in the authorship networks (the fourth column) are all within the same order of magnitude. These results indicate that the higher-order clustering coefficient can characterize the internal nature of the data. Moreover, by observing the running time in Fig. 11, we can clearly see that our biclique counting algorithm is very efficient to compute $hcc_{p,q}$. These results confirm the efficiency and effectiveness of our solutions.

Finding the (p, q)-biclique densest subgraph. We compare our peeling algorithm and the exact algorithm [21] to identify the (p, q)-biclique densest subgraph. Table 6 shows the results, where "INF" means that the algorithm runs out of memory. As can be seen, the result quality obtained by our peeling algorithm is almost the same as that returned by the exact algorithm. However, the running time of our peeling algorithm is at least one order of magnitude faster than the exact algorithm on most datasets. The exact algorithm is intractable on many datasets, since it needs to construct a flow network based on all (p, q)-bicliques. For example, on IMDB, when $p = q = 2$, our algorithm takes 72 seconds, while the exact algorithm consumes 4200 seconds. These results indicate that the proposed peeling algorithm, which is based on our biclique counting technique, is indeed very efficient and effective for mining the higher-order densest subgraph in bipartite graphs.

8 RELATED WORK

K-clique counting. Our work is closely related to the k-clique counting problem. The first exact k-clique counting (or listing) algorithm was proposed in [9] based on a backtracking enumeration technique. Such an algorithm is recently optimized by using ordering-based techniques, including the degeneracy ordering optimization proposed by Danisch et al. [10], and the color ordering optimization developed by Li et al. [17]. All these algorithms are enumeration based algorithms, thus they only work well when k is small. To improve the efficiency, Jain and Seshadhri [13] developed an elegant algorithm, called PIVOTER, based on the pivoting technique for maximal clique enumeration [7, 27]. The PIVOTER algorithm can count the k-cliques for all k using a combinatorial counting technique, instead of exhaustively enumerating all k-cliques, thus it is substantially more efficient than all the enumeration-based algorithms. However, PIVOTER may also be very costly when processing large dense graphs [13]. To overcome this limitation, sampling-based solutions are also proposed. The state-of-the-art sampling based algorithms include the TuranShadow algorithm [12] and its improved version [14] proposed by Jain and Seshadhri, as well as the color-based sampling algorithm developed by Ye et al. [36]. All these sampling-based algorithms, however, also perform poorly when k is large. It is worth mentioning that there exist many other sampling-based algorithms for counting small subgraphs [3, 5, 15, 23, 33]. All of these algorithms were shown to be less efficient than the TuranShadow algorithm [12, 14] and the color-based sampling algorithm [36] for counting k-cliques. Unlike all these studies, our work focuses mainly on counting bicliques in bipartite graph, and all the previous techniques cannot be directly extended to handle our problems.

Subgraph enumeration in bipartite graphs. Our work is also closely related to the small subgraph enumeration problem in bipartite graphs. Butterfly, a notable small subgraph in bipartite graphs, has attracted much attention in recent years. There have been many techniques to enumerate (or count) butterflies in a bipartite graph, including both exact algorithms [24, 29–31, 34, 41] and approximation algorithms [18, 24, 25, 41]. Recently, another type of small subgraph in bipartite graphs, called bi-triangle or called 6-cycle, was introduced in [35], and a wedge-based algorithm for listing all bi-triangles was also proposed in [35]. In addition to small subgraph enumeration, the problem of enumerating maximal bicliques in bipartite graphs is also widely studied in recent years. One impressive work is the iMBEA algorithm developed by Zhang et al. [40] which is mainly based on a set enumeration tree technique. Abidi et al. [1] introduced a vertex-based pivoting technique, namely PMBEA, to solve this problem. Their pivoting technique, however, needs to construct a set containment DAG (directed acyclic graph) among all vertices' neighbor-sets, which is often costly in large graphs. Chen et al. [8] further presented an algorithm called ooMBEA to speed up the enumeration of iMBEA with a total search order optimization. In this work, we develop a novel edge-pivoting technique that can also be used for maximal biclique enumeration, although we focus mainly on using the edge-pivoting technique for biclique counting.

9 CONCLUSION

In this paper, we systematically investigate the problem of counting (p, q)-bicliques. We first propose an exact algorithm, called EPivoter, based on a novel edge-pivoting technique. Then, we propose two approximate algorithms, called ZigZag and ZigZag++, based on a novel dynamic programming based h-zigzag sampling technique. We further develop a hybrid algorithm based on a carefully-designed graph partition strategy. A nice feature of all our algorithms is that they can count the (p, q)-bicliques for all pairs of (p, q), while previous algorithms are mainly tailored to count the (p, q)-bicliques with only one pair of (p, q). Extensive experiments on several real-life bipartite graphs demonstrate the high efficiency and effectiveness of the proposed solutions.

10 ACKNOWLEDGMENTS

This work was partially supported by (i) National Key Research and Development Program of China 2020AAA0108503, (ii) NSFC Grants U2241211, 62072034, U1809206 and (iii) CCF-Huawei Populus Grove Fund. Rong-Hua Li is the corresponding author of this paper.

REFERENCES

[1] Aman Abidi, Rui Zhou, Lu Chen, and Chengfei Liu. 2020. Pivot-based Maximal Biclique Enumeration. In *IJCAI*.

[2] Nesreen K. Ahmed, Nick G. Duffield, Theodore L. Willke, and Ryan A. Rossi. 2017. On Sampling from Massive Graph Streams. *Proc. VLDB Endow.* 10, 11 (2017), 1430–1441.

[3] Noga Alon, Raphael Yuster, and Uri Zwick. 1994. Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. In *STOC*.

[4] Anonymous authors. 2022. Efficient Biclique Counting in Large Bipartite Graphs. In *Full version:*. https://anonymous.4open.science/r/biclique_counting

[5] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *ACM Trans. Knowl. Discov. Data* 12, 4 (2018), 48:1–48:25.

[6] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2019. Motivo: Fast Motif Counting via Succinct Color Coding and Adaptive Sampling. *Proc. VLDB Endow.* 12, 11 (2019), 1651–1663.

[7] Coenraad Bron and Joep Kerbosch. 1973. Finding All Cliques of an Undirected Graph (Algorithm 457). *Commun. ACM* 16, 9 (1973), 575–576.

[8] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2022. Efficient Maximal Biclique Enumeration for Large Sparse Bipartite Graphs. *Proc. VLDB Endow.* 15, 8 (2022), 1559–1571.

[9] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223.

[10] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-cliques in Sparse Real-World Graphs. In *WWW*.

[11] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V. S. Lakshmanan, and Xuemin Lin. 2019. Efficient Algorithms for Densest Subgraph Discovery. *Proc. VLDB Endow.* 12, 11 (2019), 1719–1732.

[12] Shweta Jain and C. Seshadhri. 2017. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *WWW*.

[13] Shweta Jain and C. Seshadhri. 2020. The Power of Pivoting for Exact Clique Counting. In *WSDM*.

[14] Shweta Jain and C. Seshadhri. 2020. Provably and Efficiently Approximating Near-cliques using the Turán Shadow: PEANUTS. In *WWW*.

[15] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. Path Sampling: A Fast and Provable Method for Estimating 4-Vertex Subgraph Counts. In *WWW*.

[16] Jinyan Li, Guimei Liu, Haiquan Li, and Limsoon Wong. 2007. Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A One-to-One Correspondence and Mining Algorithms. *IEEE Trans. Knowl. Data Eng.* 19, 12 (2007), 1625–1637.

[17] Ronghua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering Heuristics for k-clique Listing. *Proc. VLDB Endow.* 13, 11 (2020), 2536–2548.

[18] Rundong Li, Pinghui Wang, Peng Jia, Xiangliang Zhang, Junzhou Zhao, Jing Tao, Ye Yuan, and Xiaohong Guan. 2021. Approximately Counting Butterflies in Large Bipartite Graph Streams. *IEEE Transactions on Knowledge and Data Engineering* (2021), 1–1.

[19] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2020. Efficient (α, β)-core computation in bipartite graphs. *VLDB J.* 29, 5 (2020), 1075–1099.

[20] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D Chklovskii, and U. Alon. 2010. Network Motifs: Simple Building Blocks of Complex Networks. *Science* 298, 5594 (2010), 763–764.

[21] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos E. Tsourakakis, and Shen Chen Xu. 2015. Scalable Large Near-Clique Detection in Large-Scale Networks via Sampling. In *SIGKDD, 2015*.

[22] Caixia Qin, Mingxue Liao, Yuanyuan Liang, and Changwen Zheng. 2019. Efficient Algorithm for Maximal Biclique Enumeration on Bipartite Graphs. In *ICNC-FSKD*.

[23] Mahmudur Rahman, Mansurul Alam Bhuiyan, and Mohammad Al Hasan. 2014. Graft: An Efficient Graphlet Counting Method for Large Graph Analysis. *IEEE Trans. Knowl. Data Eng.* 26, 10 (2014), 2466–2478.

[24] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyüce, and Srikanta Tirthapura. 2018. Butterfly Counting in Bipartite Networks. In *KDD*.

[25] Aida Sheshbolouki and M. Tamer Özsu. 2022. sGrapp: Butterfly Approximation in Streaming Graphs. *ACM Trans. Knowl. Discov. Data* 16, 4 (2022), 76:1–76:43.

[26] Bintao Sun, Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. 2020. KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs. *Proc. VLDB Endow.* 13, 10 (2020), 1628–1640.

[27] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.* 363, 1 (2006), 28–42.

[28] Charalampos E. Tsourakakis. 2015. The K-clique Densest Subgraph Problem. In *WWW*.

[29] Jia Wang, Ada Wai-Chee Fu, and James Cheng. 2014. Rectangle Counting in Large Bipartite Graphs. In *IEEE International Congress on Big Data*.

[30] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *Proc. VLDB Endow.* 12, 10 (2019), 1139–1152.
[31] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2022. Accelerated butterfly counting with vertex priority on bipartite graphs. *The VLDB Journal* (2022), 1–25.
[32] Kai Wang, Gengda Zhao, Wenjie Zhang, Xuemin Lin, Ying Zhang, Yizhang He, and Chunxiao Li. 2023. Cohesive Subgraph Discovery over Uncertain Bipartite Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2023).
[33] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John C. S. Lui, Don Towsley, Jing Tao, and Xiaohong Guan. 2018. MOSS-5: A Fast Method of Approximating Counts of 5-Node Graphlets in Large Graphs. *IEEE Trans. Knowl. Data Eng.* 30, 1 (2018), 73–86.
[34] Jianye Yang, Yun Peng, and Wenjie Zhang. 2021. (p, q)-biclique Counting and Enumeration for Large Sparse Bipartite Graphs. *Proc. VLDB Endow.* 15, 2 (2021), 141–153.
[35] Yixing Yang, Yixiang Fang, Maria E. Orlowska, Wenjie Zhang, and Xuemin Lin. 2021. Efficient Bi-triangle Counting for Large Bipartite Networks. *Proc. VLDB Endow.* 14, 6 (2021), 984–996.
[36] Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongzhi Chen, and Guoren Wang. 2022. Lightning Fast and Space Efficient k-clique Counting. In *WWW*.
[37] Hao Yin, Austin R. Benson, and Jure Leskovec. 2017. Higher-order clustering in networks. *Physical Review E* 97, 5 (2017), 052306.
[38] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. 2017. Local Higher-Order Graph Clustering. In *KDD*.
[39] Yun Zhang, Charles A. Phillips, Gary L. Rogers, Erich J. Baker, Elissa J. Chesler, and Michael A. Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinform.* 15 (2014), 110.
[40] Yun Zhang, Charles A. Phillips, Gary L. Rogers, Erich J. Baker, Elissa J. Chesler, and Michael A. Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinform.* 15 (2014), 110.
[41] Alexander Zhou, Yue Wang, and Lei Chen. 2021. Butterfly Counting on Uncertain Bipartite Networks. *Proc. VLDB Endow.* 15, 2 (2021), 211–223.

Received July 2022; revised October 2022; accepted November 2022