(p,q)-biclique Counting and Enumeration for Large Sparse Bipartite Graphs

Jianye Yang
Guangzhou University
Guangzhou, China
jyyang@hnu.edu.cn

Yun Peng*
Hong Kong Baptist University
Hong Kong, China
yunpeng@comp.hkbu.edu.hk

Wenjie Zhang
University of New South Wales
Sydney, Australia
zhangw@cse.unsw.edu.au

ABSTRACT

In this paper, we study the problem of (p, q)-biclique counting and enumeration for large sparse bipartite graphs. Given a bipartite graph $G = (U, V, E)$, and two integer parameters p and q, we aim to efficiently count and enumerate all (p, q)-bicliques in G, where a (p, q)-biclique $B(L, R)$ is a complete subgraph of G with $L \subseteq U$, $R \subseteq V$, $|L| = p$, and $|R| = q$. The problem of (p, q)-biclique counting and enumeration has many applications, such as graph neural network information aggregation, densest subgraph detection, and cohesive subgroup analysis, etc. Despite the wide range of applications, to the best of our knowledge, we note that there is no efficient and scalable solution to this problem in the literature.

This problem is computationally challenging, due to the worst-case exponential number of (p, q)-bicliques. In this paper, we propose a competitive branch-and-bound baseline method, namely BCList, which explores the search space in a depth-first manner, together with a variety of pruning techniques. Although BCList offers a useful computation framework to our problem, its worst-case time complexity is exponential to $p + q$. To alleviate this, we propose an advanced approach, called BCList++. Particularly, BCList++ applies a layer based exploring strategy to enumerate (p, q)-bicliques by anchoring the search on either U or V only, which has a worst-case time complexity exponential to either p or q only. Consequently, a vital task is to choose a layer with the least computation cost. To this end, we develop a cost model, which is built upon an unbiased estimator for the density of 2-hop graph induced by U or V. To improve computation efficiency, BCList++ exploits pre-allocated arrays and vertex labeling techniques such that the frequent subgraph creating operations can be substituted by array element switching operations. We conduct extensive experiments on 16 real-life datasets, and the experimental results demonstrate that BCList++ significantly outperforms the baseline methods by up to 3 orders of magnitude. We show via a case study that (p, q)-bicliques optimize the efficiency of graph neural networks.

PVLDB Reference Format:
Jianye Yang, Yun Peng, and Wenjie Zhang. (p,q)-biclique Counting and Enumeration for Large Sparse Bipartite Graphs. PVLDB, 15(2): 141-153, 2022.
doi:10.14778/3489496.3489497

*Yun Peng is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 15, No. 2 ISSN 2150-8097.
doi:10.14778/3489496.3489497

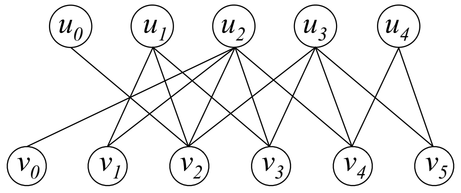


Figure 1: An example bipartite graph G.

1 INTRODUCTION

As a natural data structure to model relationships between two different types of entities [3, 18], bipartite graph has been used in many real-world applications, such as, online customer-product networks [39, 41], gene co-expression networks [49], author-paper networks [8], and graph neural networks [14], etc. Formally, a bipartite graph $G = (U, V, E)$ consists of two disjoint vertex sets U and V, where an edge $e \in E$ connects a vertex in U and another in V. An example bipartite graph is shown in Figure 1. Recently, a lot of research efforts have been devoted to many fundamental problems in analyzing bipartite graphs, such as (α, β)-core query [9, 22, 45], maximal biclique enumeration [1, 23, 24, 49], butterfly counting [30, 40, 42], and fraud detection [15, 17, 39], to name just a few.

In this paper, we introduce the concept of (p, q)-biclique. Given a bipartite graph $G = (U, V, E)$, a biclique $B(L, R)$ is a complete subgraph of G, where $L \subseteq U$, $R \subseteq V$, that is $\forall(u, v) \in L \times R$, $(u, v) \in E(G)$, and $B(L, R)$ is called a (p, q)-biclique if $|L| = p$ and $|R| = q$. We study the problem of (p, q)-biclique *counting* and *enumeration* for large sparse bipartite graphs, given two integer parameters p and q.

Motivations. Many real-world bipartite graphs are very large and sparse, such as those listed in Table 3. A special case of (p, q)-biclique (where $p = 2$ and $q = 2$) called butterfly [30, 40, 42] has demonstrated great importance in defining basic metrics such as the clustering coefficient in a bipartite graph [21, 27]. However, in many graph-based tasks, (p, q)-bicliques, where p and q are not fixed to 2, are needed. Below are a small list of examples.

(1) GNN Information Aggregation. Graph neural Network (GNN) has received much research interests in recent years [14, 38, 48] and has numerous applications. A pivotal operation in a GNN is to recursively aggregate information from vertices' neighbors in graph. A naive method simply propagates information on each pair of vertices separately, which leads to redundant computations, since vertices in a graph may often share many neighbors. Interestingly, we remark that (p, q)-biclique enumeration can optimize the task of GNN information aggregation. Our case study results in Section 6.4 report that our (p, q)-biclique based method achieves the most efficient when (p, q) settings are $(5, 10)$ and $(4, 10)$ on two

datasets IMDB and PPI, respectively, which are employed to evaluate the state-of-the-art algorithm HAG[16]. The results show that our method achieves near an order of magnitude of performance improvement over existing methods. Details about how to apply (p, q)-bicliques to GNN information aggregation are illustrated in our case study in Section 6.4.

(2) Densest Subgraph Detection. Recently, Mitzenmacher *et al.* [26] formulate the concept of (p, q)-biclique density. For a subgraph S in a bipartite graph, its (p, q)-biclique density is defined as the ratio between the number of (p, q)-bicliques in S and the size of S. Based on the (p, q)-biclique density, they study the problem of finding the (p, q)-biclique densest subgraph in a bipartite graph. They point out that (p, q)-biclique enumeration is a required procedure in their methods.

(3) Cohesive Subgroup Analysis. Borgatti *et al.* [3] study the problem of identifying cohesive subgroups in bipartite graphs. They consider using (p, q)-biclique to identify cohesive subgroups in a bipartite graph, where p and q are greater than or equal to 3. For example, in a social event bipartite graph with 18 guests and 14 events, (3, 3)-bicliques can reveal two basic groups together with some outsiders, which matches the ground truth well.

To the best of our knowledge, the problem of (p, q)-biclique counting and enumeration on large sparse bipartite graphs has not been thoroughly investigated. The closest related work is the work of finding (p, q)-biclique densest subgraph [26], where (p, q)-biclique enumeration is a necessary step. The performance of their solution is not yet satisfactory, since it is developed on top of costly maximal biclique enumeration. $(2, 2)$-biclique and $(3, 3)$-biclique based densities are adopted to make their solution more practical. However, even under the setting $p = q = 3$, their solution cannot finish in 10 hours on a medium-sized bipartite graph (with 18k vertices and 92k edges).

Challenges. The problem of (p, q)-biclique counting and enumeration is computationally challenging. Given a bipartite graph $G = (U, V, E)$, a straightforward solution is to enumerate all $\binom{|U|}{p}\binom{|V|}{q}$ combinations of vertex sets and verify whether each of them is a biclique. However, this approach is obviously cost-prohibitive, because the number of (p, q)-bicliques in a bipartite graph can be much larger than the size of the bipartite graph. Assuming $|U| = |V| = n$, then the number of (p, q)-bicliques could be up to $O(\binom{|U|}{p}\binom{|V|}{q}) \approx O(n^{p+q})$. This shows that our problem is more difficult than other related counting problems, such as butterfly counting [30, 42] and bi-triangle counting [47]. For example, the dataset *Twitter* in Table 3 contains 2.06×10^8 butterflies and 1.61×10^{12} bi-triangles, but 1.45×10^{19} (6, 2)-bicliques.

Besides, we aim to support queries with arbitrary p and q values, which makes our problem even harder. For problems such as butterfly counting and bi-triangle counting, where the answers are small and fixed patterns, one may build many intermediate structures (e.g., wedges) to facilitate query processing. This is infeasible to our problem since the intermediate result size would be extremely huge even for slightly larger p and q values. Last but not least, we aim to support both counting and enumerating (p, q)-bicliques, which is generally more difficult than counting only.

Contribution. To tackle the computation challenges, we propose efficient query processing techniques. In the database research community, algorithms that combine backtracking with branch-and-bound techniques are widely adopted to deal with graph based problems, such as maximal biclique enumeration [1, 49]. Inspired by these, we propose a competitive branch-and-bound baseline method, namely BCList, which explores the search space in a depth-first manner. Specifically, BCList maintains a partial biclique and recursively adds the candidate vertices into the partial biclique to generate (p, q)-bicliques. To improve the performance of BCList, we propose new efficient pruning techniques, such as 2-hop neighbors to reduce candidate size, size pruning to terminate search branch early, and vertex ordering to avoid redundant computation.

BCList provides a promising computation framework to solve our problem. However, observe that BCList utilizes a vertex based exploring strategy, which has a time complexity of $O((|U| + |V|)(d_{max} + d_{2max})^{p+q-2})$, where d_{max} and d_{2max} are the maximum degree and 2-hop degree of a vertex in G, respectively. It is still time costly for BCList to handle queries when p and q are large.

To alleviate these issues, we propose an advanced approach, called BCList++. In particular, BCList++ applies a layer based exploring strategy to enumerate (p, q)-bicliques by anchoring the search on either U or V only. This exploring strategy leads to a time complexity of $O(a(H)^{p-2}|E(H)|d_{max} + \triangle)$ if the left layer U is selected, where H is the 2-hop graph constructed on U, $E(H)$ is the edges in H, $a(H)$ is the arboricity of H [6], d_{max} is the maximum degree of a vertex in U, and $\triangle$ is the result size. Compared to BCList, BCList++ is more efficient w.r.t the values of p and q. It should also be remarked that many real bipartite graphs are unbalanced in practice (e.g., *Edit-en* and *Edit-fr* in Table 3), and the values of p and q may be quite different as well. Hence, the performance of BCList++ could be significantly influenced by the choice of the search layer (i.e., U or V). Consequently, a vital task is to choose a layer with the least computation cost. To this end, we develop a new cost model, built upon an unbiased estimator for the density of 2-hop graph H, to efficiently estimate the computation cost. We theoretically analyze the effectiveness of the cost model.

We use pre-allocated arrays and vertex labelling techniques to implement BCList++ such that the frequent subgraph creating operations can be substituted by array element switching operations. To further accelerate the computation, we introduce useful graph reduction techniques and extend our approach to a parallel environment, where multiple threading is available. Empirical study shows that BCList++ can significantly outperform the baseline method BCList and other competitors by up to 3 orders of magnitude. A case study about our techniques for optimizing the efficiency of GNN information aggregation is presented.

Our principle contributions are summarized as follows.

- This is the first work to systematically study the problem of (p, q)-biclique counting and enumeration for large sparse bipartite graphs. We propose BCList which combines backtracking with branch-and-bound techniques, together with a variety of pruning techniques.
- We propose a layer based approach BCList++, where a cost model is used to guide the selection of the layer with the least computation cost. To improve efficiency further, we implement BCList++ using pre-allocated arrays and vertex labelling techniques such that the frequent subgraph creating

Table 1: Frequently used notations.

Notation	Meaning
G	a bipartite graph
$U(G), V(G)$	a set of vertices in G
$E(G)$	a set of edges in G
u, v, w	a vertex in a bipartite graph
(u, v)	an edge in a bipartite graph
$u \rightarrow v$	an directed edge from u to v
$N(u, G)$	the neighbors of u in G
$d(u, G)$	the degree of u in G
$N_2(u, G)$	the 2-hop neighbors of u in G
$B(L, R)$	a biclique in a bipartite graph
$r(u)$	the rank of u
H	the 2-hop graph of a bipartite graph
$N(u, H)$	the neighbors of u in a 2-hop graph H

operations can be substituted by array element switching operations.

- The comprehensive performance evaluation on real data demonstrates the efficiency of our new techniques proposed in this paper.

Roadmap. The rest of this paper is organized as follows. In Section 2, we introduce basic concepts and problem definition. In Section 3, we propose a baseline method. In Section 4, we propose an advanced approach. Two optimizations are proposed in Section 5. We conduct extensive experiments in Section 6. Section 7 reviews the related work and Section 8 concludes the paper.

2 PRELIMINARIES

In this section, we introduce basic concepts and definitions used in this paper. Table 1 summarizes some notations frequently used throughout this paper. We consider an unweighted and undirected bipartite graph $G = (U, V, E)$, where $U(G)$ and $V(G)$ denote two disjoint vertex sets, i.e., $U(G) \cap V(G) = \emptyset$, and $E(G) \subseteq U(G) \times V(G)$ denotes the edge set of G. In this paper, we call $U(G)$ and $V(G)$ the left and right side (or *layer*) of vertices in G, respectively. An edge in G is denoted by either (u, v) or (v, u). For each vertex $u \in U(G)$, the neighbors of u is denoted as $N(u, G) = \{v | (u, v) \in E(G)\}$. The degree of u, denoted as $d(u, G)$, is the number of neighbors of u, i.e., $d(u, G) = |N(u, G)|$. We have symmetrical definitions for vertices in $V(G)$. For presentation simplicity, in the rest of the paper, we omit G in the notations when the context is self-evident.

Definition 1 (**Biclique**). *Given a bipartite graph $G = (U, V, E)$, a biclique $B(L, R)$ is a complete subgraph of G, where $L \subseteq U(G)$ and $R \subseteq V(G)$, i.e., $\forall (u, v) \in L \times R, (u, v) \in E(G)$.*

Definition 2 (**(p, q)-Biclique**). *Given a bipartite graph G, and two integer parameters p and q, a (p, q)-biclique $B(L, R)$ is a biclique of G with $|L| = p$ and $|R| = q$.*

Problem Statement. Given a bipartite graph $G = (U, V, E)$, and two integer parameters p and q, we study the problem of counting and enumerating (p, q)-bicliques in G.

Example 1. *Consider the bipartite graph in Figure 1. Assuming $p = 2$ and $q = 3$, there are two (p, q)-bicliques. They are $(\{u_1, u_2\}, \{v_1, v_2, v_3\})$ and $(\{u_2, u_3\}, \{v_2, v_3, v_4\})$.*

In the rest of the paper, we focus on the enumeration problem and we show how to extend our techniques to the counting problem.

3 THE BASELINE SOLUTION

A brute-force solution for our problem is to enumerate all $\binom{|U|}{p}\binom{|V|}{q}$ combinations of vertex sets and verify whether each of them is a biclique, which is cost-prohibitive. In the database research community, algorithms that combine backtracking with branch-and-bound techniques are widely adopted to deal with graph based problems, such as maximal biclique enumeration [1, 49]. Inspired by these, in this section, we propose a competitive branch-and-bound baseline method, called BCList. In the following, we first give the main idea of BCList together with some important pruning techniques, and then present the overall algorithm.

3.1 Solution Overview

We begin with the concept of partial biclique.

Definition 3 (**Partial Biclique**). *Given a pair of integers p and q, a partial biclique $B(L, R)$ is biclique with $|L| \leq p$ and $|R| < q$, or $|L| < p$ and $|R| \leq q$.*

Main Idea. In a nutshell, BCList maintains a partial biclique and recursively adds the candidate vertices into the partial biclique to generate full bicliques, i.e., (p, q)-bicliques. More specifically, BCList operates on the following four dynamically changing vertex sets: (i) L, a subset of U containing the left side of vertices in a partial biclique; (ii) R, a subset of V containing the right side of vertice in a partial biclique; (iii) C_L, a subset of U containing the candidate vertices that may be added to L; (iv) C_R, a subset of V containing the candidate vertices that may be added to R. In each iteration, BCList chooses one vertex from C_L or C_R to expand the partial biclique. The four vertex sets are utilized and maintained in a depth-first traversal of a recursion search tree to generate (p, q)-bicliques.

Clearly, to improve the computation efficiency, the key is to reduce the search space, i.e., the size of recursion search tree. In the following, we aim to develop efficient pruning and query processing techniques.

3.2 Pruning Techniques

Lemma 1. *Given a partial biclique $B(L, R)$, the candidate sets C_L and C_R only contain the vertices that are the common neighbors of vertices in R and L, respectively.*
(1) If $u \in C_L$, then $\forall v \in R : (u, v) \in E$; and
(2) If $v \in C_R$, then $\forall u \in L : (u, v) \in E$.

All proofs in this paper are omitted due to space limit[1]. Based on Lemma 1, we can substantially reduce the number of candidate vertices by only considering the common neighbors. In particular, when the partial bicliques are expanded, C_L and C_R are contracted. We have the following corollary based on Lemma 1.

Corollary 1. *Given a partial biclique $B(L, R)$, and the corresponding candidate sets C_L and C_R,*
(1) if $|L| = p$ and $|R| + |C_R| \geq q$, then L forms a (p, q)-biclique with each q-sized subset of $R \cup C_R$; and
(2) if $|R| = q$ and $|L| + |C_L| \geq p$, then R forms a (p, q)-biclique with each p-sized subset of $L \cup C_L$.

[1]A full version of this paper is available in [46]

Algorithm 1: Collect2HopNeighbors(G, p, q)

```
Input    : $G$ : a bipartite graph
           $p, q$ : two parameters
for each $u \in U \cup V$ do
    Initialize hashmap $C$ with zero;
    for each $v \in N(u, G)$ do
        for each $w \in N(v, G)$ do
            if $u \neq w$ then
                $C[w] \leftarrow C[w] + 1$;
    for each $w \in C$ do
        if $u \in U$ and $C[w] \geq q$ or $u \in V$ and $C[w] \geq p$ then
            $N_2(u, G) \leftarrow N_2(u, G) \cup \{w\}$;
```

Lemma 1 provides an efficient way to reduce the number of candidate vertices on the opposite side when a new vertex is added into the partial biclique. Next, we explore to reduce the number of candidate vertices on the same side as well. Before that, we introduce the important concept of 2-hop neighbor.

Definition 4 (*τ*-strength 2-Hop Neighbor). *Given a bipartite graph $G = (U, V, E)$ and an integer τ, for a vertex w in G, the τ-strength 2-hop neighbors of w, denoted as $N_2^\tau(w, G)$, contains all vertices in G, each of which has at least τ common neighbors with w, i.e., $N_2^\tau(w, G) = \{w' | w' \in U \cup V$ and $|N(w, G) \cap N(w', G)| \geq \tau\}$.*

For presentation convenience, given a bipartite graph $G = (U, V, E)$, and two integers p and q, for each $u \in U$, we define the 2-hop neighbors of u, denoted as $N_2(u, G)$, to be the q-strength 2-hop neighbors of u, i.e., $N_2(u, G) = N_2^q(u, G)$. Similarly, for each $v \in V$, $N_2(v, G) = N_2^p(v, G)$.

Example 2. *Consider u_3 in the bipartite graph in Figure 1 again. We assume $p = 2$ and $q = 3$. Since u_2 is the only 3-strength 2-hop neighbor of u_3 with common neighbors v_2, v_3, and v_4. We have $N_2(u_3, G) = \{u_2\}$.*

Lemma 2. *Given a partial biclique $B(L, R)$, the candidate sets C_L and C_R only contain the vertices that are common 2-hop neighbors of L and R, respectively.*
(1) If $u \in C_L$, then $\forall w \in L$: $u \in N_2(w, G)$; and
(2) If $v \in C_R$, then $\forall w \in R$: $v \in N_2(w, G)$.

Example 3. *Following Example 2, suppose $L = \{u_3\}$. Based on Lemma 2, we have that the candidate set $C_L \subseteq N_2(u_3, G)$, which implies that we only need to consider u_2 as the candidate vertex to expand L, rather than all vertices in $\{u_0, u_1, u_2, u_4\}$.*

Collecting 2-hop Neighbors. Algorithm 1 illustrates the details of collecting the 2-hop neighbors of vertices in a bipartite graph. For each vertex u in G, we use a hashmap C to keep the 2-hop neighbors of u along with the number of common neighbors (Line 2). In the algorithm, we first search the neighbors of u (Line 3), and then search the 2-hop neighbors (Line 4). If the possible 2-hop neighbor is a vertex rather than u itself, we increase the entry in C by 1 (Lines 5-6). After processing all neighbors of u, we check the candidate 2-hop neighbors, and only keep the q-strength (resp. p-strength) 2-hop neighbors if $u \in U$ (resp. $u \in V$) (Lines 7-9).

Table 2: The neighbor and 2-hop neighbor structure of G.

Vertex Id	$N(u, G)$	$N_2(u, G)$
u_0 :		
u_1 :	v_3, v_1	
u_2 :	v_2, v_3, v_4, v_1, v_0	u_3, u_1
u_3 :	v_2, v_3, v_4, v_5	
u_4 :	v_5	
v_0 :		
v_1 :		
v_2 :	u_1, u_0	v_3, v_4, v_1
v_3 :		v_4, v_1
v_4 :	u_4	v_5
v_5 :		

Theorem 1. *The time complexity of Algorithm 1 is $O(\sum_{u \in U} d(u, G)^2 + \sum_{v \in V} d(v, G)^2)$.*

Size Pruning. During the search processing, when the size of vertex set is relatively small, we may stop exploiting the current branch without missing any results.

Lemma 3. *Given a partial biclique $B(L, R)$, and the candidate sets C_L and C_R, if $|L| + |C_L| < p$ or $|R| + |C_R| < q$, the four sets L, R, C_L, and C_R cannot generate any (p, q)-bicliques.*

Vertex Ordering. As a frequently considered factor to improve the efficiency of many graph search algorithms [6, 7, 42], vertex ordering can be utilized to avoid generating duplicate results and thus save computation cost. In the literature, the degree ordering [6, 42] and core ordering [7] are two widely adopted vertex ordering strategies. Our experimental studies in Section 6 show that the two orderings achieve comparable performance. Next, we introduce the degree ordering since it is more computationally efficient.

Definition 5 (Vertex Rank). *Given a bipartite graph $G = (U, V, E)$, for a vertex u in G, the vertex rank $r(u)$ is an integer where $r(u) \in [1, |U \cup V|]$. For two vertices $u, v \in U \cup V$, $r(u) > r(v)$ if*

- *$d(u) > d(v)$, or*
- *$d(u) = d(v)$ and $u.id < v.id$[2].*

Graph Transformation. Given a bipartite graph G, let η be a degree ordering on G. We say the directed bipartite graph $\overrightarrow{G}$ is induced by the ordering η, if $U(\overrightarrow{G}) := U(G)$, $V(\overrightarrow{G}) := V(G)$, and there is an edge $u \to v$ in $\overrightarrow{G}$ if $r(u) > r(v)$ and $(u, v) \in E(G)$.

In the rest of the paper, we slightly abuse the notation of G to denote its induced directed graph $\overrightarrow{G}$, unless otherwise specified. For a vertex u in G, $N(u, G)$ only keeps the neighbors of u with a lower vertex rank than u. We have similar changes for $d(u, G)$ and $N_2(u, G)$. Besides, we sort vertices in $U(G)$, $V(G)$, $N(u, G)$, and $N_2(u, G)$ in descending order of their ranks by preprocessing.

Example 4. *Following the example in Figure 1. Assume $p = 2$ and $q = 3$. The sorted vertex order is as follows: $u_2, u_3, v_2, u_1, v_3, v_4, u_4, v_1, v_5, u_0, v_0$. Table 2 shows the neighbor and 2-hop neighbor structure of each vertex after graph transformation. Take u_1 as example. Although it has 3 neighbors, namely v_1, v_2, and v_3, we only keep v_3 and v_1 in $N(u_1, G)$. This is because v_2 has a higher rank than u_1. Note that*

[2]We assume $u.id < v.id$ if $u \in U$ and $v \in V$.

```
Algorithm 2: BCList($G$, $p$, $q$)
  Input  : $G$: a bipartite graph
           $p$, $q$: two parameters
  Output : $\mathcal{B}$: all ($p$, $q$)-bicliques
1  Collect2HopNeighbors($G$, $p$, $q$);
2  Compute the rank $r(u)$ for each $u \in U(G) \cup V(G)$;
3  $G \leftarrow$ directed version of $G$, where $u \rightarrow v$ if $r(u) > r(v)$;
4  VertexBasedListing($\emptyset$, $\emptyset$, $U(G)$, $V(G)$);
5  return $\mathcal{B}$;
6  procedure VertexBasedListing($L$, $R$, $C_L$, $C_R$)
7  if $|L| \geq p$ and $|R| + |C_R| \geq q$ or $|R| \geq q$ and $|L| + |C_L| \geq p$ then
8      if $|L| \geq p$ then
9          for each $S \subseteq R \cup C_R$: $|S| = q$ do
10             $\mathcal{B} \leftarrow \mathcal{B} \cup \{(L, S)\}$;
11     else
12         for each $S \subseteq L \cup C_L$: $|S| = p$ do
13             $\mathcal{B} \leftarrow \mathcal{B} \cup \{(S, R)\}$;
14 else if $|L| + |C_L| < p$ or $|R| + |C_R| < q$ then
15     return                                              /* Lemma 3 */;
16 else
17     $L' \leftarrow L$, $R' \leftarrow R$;
18     $i \leftarrow j \leftarrow 0$;
19     while $i < |C_L|$ and $j < |C_R|$ do
20         $u \leftarrow C_L[i]$, $v \leftarrow C_R[j]$;
21         if $r(u) > r(v)$ then
22             $L' \leftarrow L' \cup \{u\}$;
23             $C'_L \leftarrow C_L[i + 1 :] \cap N_2(u, G)$;          /* Lemma 2 */;
24             $C'_R \leftarrow C_R \cap N(u, G)$;                   /* Lemma 1 */;
25             VertexBasedListing($L'$, $R'$, $C'_L$, $C'_R$);
26             $L' \leftarrow L' - \{u\}$;
27             $i \leftarrow i + 1$;
28         else
29             $R' \leftarrow R' \cup \{v\}$;
30             $C'_R \leftarrow C_R[j + 1 :] \cap N_2(v, G)$;          /* Lemma 2 */;
31             $C'_L \leftarrow C_L \cap N(v, G)$;                   /* Lemma 1 */;
32             VertexBasedListing($L'$, $R'$, $C'_L$, $C'_R$);
33             $R' \leftarrow R' - \{v\}$;
34             $j \leftarrow j + 1$;
```

vertices in $N(u, G)$ and $N_2(u, G)$ are sorted in descending order of their ranks.

3.3 The Overall Algorithm of BCList

Based on the above observations, we are ready to present the BCList algorithm. We first collect the 2-hop neighbors for vertices in G (Line 1), which is described in Algorithm 1. Then, we compute the vertex rank for each vertex in the graph (Line 2) and construct the induced directed graph (Line 3). Finally, we enumerate all (p, q)-bicliques using the procedure VertexBasedListing (Line 4).

In the procedure VertexBasedListing, we maintain four vertex sets, i.e., L, R, C_L, and C_R, which are initialized as $\emptyset$, $\emptyset$, $U(G)$, and $V(G)$, respectively. During the processing of VertexBasedListing, we first check if the current branch can generate answers based on Corollary 1 (Line 7). For example, if $|L| \geq p$ and $|R| + |C_R| \geq q$, we can collect (p, q)-bicliques from L and each q-sized subset S of $R \cup C_R$ (Lines 8-10). Similarly, we might also obtain (p, q)-bicliques in Lines 11-13. Otherwise, we check if the current branch can be pruned by applying Lemma 3 (Line 14). Lastly, we search the subspaces (Lines 16-34).

More specifically, we iteratively select the vertices in C_L and C_R to expand the partial biclique, i.e., L and R (Lines 19-34). At each step, we choose from C_L and C_R the vertex with the highest rank (Line 20). Say u (i.e., $C_L[i]$) has a higher rank than v (i.e., $C_R[j]$) (Lines 21-27). Then, L' is updated by adding u to L (Line 22). Based on Lemma 2, C'_L is updated by computing the intersection of C_L and $N_2(u, G)$. Note here that we need only to consider the last $|C_L| - i$ vertices in C_L since the first i vertices have already been checked in previous iterations (Line 23). Meanwhile, based on Lemma 1, we can update C'_R (Line 24). After that, we enter the new search space formed by the vertex sets L', R', C'_L, and C'_R (Line 25). After finishing the new search space, we should remove u from L' before going to next iteration (Line 26). Lines 28-34 describe the symmetrical case where $r(C_L[i]) \leq r(C_R[j])$.

Example 5. *Following Example 4, we illustrate the overall running process of BCList. Figure 2 depicts the recursion tree of the entire search space. Note that a recursion tree contains three types of tree nodes: (i) open node marked by a dashed rectangle, which we have to explore further, e.g., s_0, s_1, s_3, and s_8; (ii) answer node marked by a green solid rectangle, where we find (p, q)-bicliques, e.g., s_2 and s_4; (iii) closed node marked by a red solid rectangle, which can be pruned safely, e.g., s_5, s_6, s_7, and s_9. Clearly, both answer and closed nodes are leaf nodes, while open node is an inner node.*

We start from the root node s_0 and iteratively search the subspaces in a depth-first manner by selecting vertices from the candidate sets following the vertex rank order. By selecting u_2, we enter node s_1, where the candidate vertex sets are updated accordingly. In particular, we have $C_L = C_L \cap N_2(u_2, G) = \{u_3, u_1\}$ and $C_R = C_R \cap N(u_2, G) = \{v_2, v_3, v_4, v_1, v_0\}$ based on Lemma 2 and Lemma 1, respectively. Clearly, we cannot prune s_1 by Lemma 3 or generate answers by Corollary 1. Therefore, we expand node s_1, and enter the child node s_2, where a (p, q)-biclique is found according to Corollary 1, i.e., $(\{u_2, u_3\}, \{v_2, v_3, v_4\})$. Continuing this processing, we can find another answer node s_4 with a (p, q)-biclique $(\{u_1, u_2\}, \{v_1, v_2, v_3\})$, while all closed nodes can be pruned by Lemma 3.

Analysis of BCList. Below we first show the correctness of BCList, and then analyze the time and space complexity of BCList.

Theorem 2. *BCList enumerates (p, q)-bicliques correctly.*

Next we focus on the main recursion procedure VertexBasedListing when analyzing the time complexity of BCList. Note that the time complexity to preprocess the graph for collecting 2-hop neighbor is shown in Theorem 1.

Theorem 3. *The time complexity of BCList is $O((|U(G)| + |V(G)|)(d_{max} + d_{2max})^{p+q-2})$, where d_{max} and d_{2max} are the maximum degree and 2-hop degree of a vertex in G, respectively.*

Theorem 4. *The space complexity of BCList is $O(|E(G)| + |U(G)|^2 + |V(G)|^2)$.*

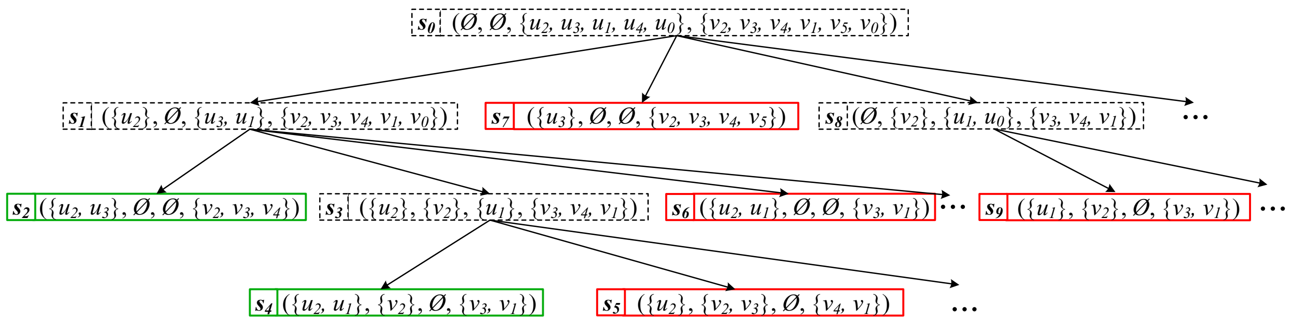

s_0 $(\emptyset, \emptyset, \{u_2, u_3, u_1, u_4, u_0\}, \{v_2, v_3, v_4, v_1, v_5, v_0\})$
s_1 $(\{u_2\}, \emptyset, \{u_3, u_1\}, \{v_2, v_3, v_4, v_1, v_0\})$
s_7 $(\{u_3\}, \emptyset, \emptyset, \{v_2, v_3, v_4, v_5\})$
s_8 $(\emptyset, \{v_2\}, \{u_1, u_0\}, \{v_3, v_4, v_1\})$
…
s_2 $(\{u_2, u_3\}, \emptyset, \emptyset, \{v_2, v_3, v_4\})$
s_3 $(\{u_2\}, \{v_2\}, \{u_1\}, \{v_3, v_4, v_1\})$
s_6 $(\{u_2, u_1\}, \emptyset, \emptyset, \{v_3, v_1\})$
…
s_9 $(\{u_1\}, \{v_2\}, \emptyset, \{v_3, v_1\})$
…
s_4 $(\{u_2, u_1\}, \{v_2\}, \emptyset, \{v_3, v_1\})$
s_5 $(\{u_2\}, \{v_2, v_3\}, \emptyset, \{v_4, v_1\})$
…

Figure 2: (Partial) Recursion tree of the running example of BCList on Figure 1 ($p = 2$ and $q = 3$).

3.4 Discussion for Counting Problem

To solve the counting problem, we only need to do minor changes for BCList. Specifically, in Lines 9-10 and Lines 12-13 of Algorithm 2, we count the number of results by using $\binom{|S|}{q}$ and $\binom{|S|}{p}$, respectively, rather than enumerating each subset of S.

4 AN ADVANCED APPROACH

In this section, we propose an advanced approach, namely BCList++, to solve the problem of (p, q)-biclique counting and enumeration.

4.1 Motivation

Although BCList offers a useful computation framework to our problem, our empirical study suggests that it still has the following two drawbacks.

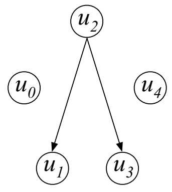
• *Drawback 1: Large Search Space.* The recursion procedure VertexBasedListing in BCList utilizes a vertex ordering based strategy to expand the search space, which may lead to a large search space in terms of both depth and width of the recursion tree when the value of $p + q$ is large. As a result, it can be inefficient since the time complexity of BCList is exponential to $p + q$ (see Theorem 3 for detials).

• *Drawback 2: Inefficient Direct Implementation.* The direct implementation of BCList does not yield an efficient algorithm because it has to produce and store a large number of intermediate subgraphs. That is, we have to produce new L, R, C_L, and C_R for each node in the recursion tree. It is costly to frequently create such data structures.

The Advanced Approach – BCList++. Based on the above analysis, we consider the following two aspects to alleviate the drawbacks of BCList.

• To resolve Drawback 1, we apply a layer based exploring strategy to enumeration (p, q)-bicliques by anchoring the search on either U or V only. This exploring strategy leads to a time complexity exponential to either p or q (see Theorem 6). To deal with the unbalanced bipartite graphs and queries with different p and q, we develop an efficient and effective cost model to choose a layer with the least computation cost.

• To tackle Drawback 2, we implement BCList++ using pre-allocated arrays and vertex labelling techniques such that the frequent subgraph creating operations can be substituted by array element switching operations. By using these data structures and operations, we only need to create them once during the entire processing of the algorithm.



(a) 2-hop graph H

Id	$N(u,H)$	$N(u,G)$
u_0 :		v_2
u_1 :		v_1, v_2, v_3
u_2 :	u_1, u_3	v_0, v_1, v_2, v_3, v_4
u_3 :		v_2, v_3, v_4, v_5
u_4 :		v_4, v_5

(b) Neighbor structures

Figure 3: 2-hop graph H and vertex neighbor structures.

4.2 BCList++ Algorithm

We begin with the important concept of 2-hop graph.

Definition 6 (2-**Hop Graph**). *Given a bipartite graph $G = (U, V, E)$, and a pair of parameters p and q, the 2-hop graph $H = (U, E)$ of G is a graph induced by G with the following properties:*
(1) $U(H) := U(G)$[3]; and
(2) $\forall u, v \in U(H)$, $(u, v) \in E(H)$ if u and v are 2-hop neighbors in G.

Given a bipartite graph G and its induced 2-hop graph H, for each $u \in U(H)$, the neighbors of u in H is denoted as $N(u, H) = \{v|(u, v) \in E(H)\}$, while $N(u, G)$ is reserved to keep neighbors of u in G. Note that, we transform H into a directed acyclic graph (DAG) and sort vertices in $U(H)$ and $N(u, H)$ in descending order of their ranks as well.

Example 6. *Continuing our running example, Figure 3(a) shows the induced 2-hop graph H of G, which contains two edges, i.e., $u_2 \to u_1$ and $u_2 \to u_3$. This is because there are only two pairs of 2-hop neighbors (i.e., $\langle u_1, u_2\rangle$ and $\langle u_2, u_3\rangle$) among vertices in $U(H)$, Figure 3(b) presents the neighbor structures of vertices in H. Note that, for each vertex $u \in U(H)$, $N(u, G)$ contains all neighbors of u in G.*

Moreover, given a 2-hop graph H, a clique c is a complete subgraph of H by ignoring the edge direction. We say a clique c is a p-clique if the number of vertices in c is p. For example, (u_1, u_2) is a 2-clique in Figure 3(a).

The Details of BCList++. Algorithm 3 illustrates the details of our advanced method BCList++. We first choose the layer with least cost as anchor layer, i.e., $U(G)$, by using a cost model (Lines 1-3), which is introduced in detail in Section 4.3. Then, we collect the 2-hop neighbors for each vertex in $U(G)$ (Line 4) and compute the

[3]The construction of H based on $V(G)$ is similar.

Algorithm 3: BCList++ (G, p, q)

```
Input   : G: a bipartite graph
          p, q: two parameters
Output  : $\mathcal{B}$: all (p, q)-bicliques
1  if Cost($U(G)$, p) > Cost($V(G)$, q) then
2      Swap($U(G)$, $V(G)$);
3      Swap(p, q);
4  Collect2HopNeighbors(G, p, q);    /* for vertices in $U(G)$ */;
5  Compute the rank $r(u)$ for each $u \in U(G)$;
6  Construct 2-hop graph H on $U(G)$;
7  $S \leftarrow p$ arrays initialized as empty;
8  LayerBasedListing(0, H, $\emptyset$);
9  return $\mathcal{B}$;
10 procedure LayerBasedListing(l, H, L)
11 if $l = p$ then
12     for each $R \subseteq S[l-1]$: $|R| = q$ do
13         $\mathcal{B} \leftarrow \mathcal{B} \cup \{(L, R)\}$;
14 for each $u \in U(H)$ do
15     if $l = 0$ then
16         $S[l] \leftarrow N(u, G)$;
17     else
18         $S[l] \leftarrow S[l-1] \cap N(u, G)$ ;
19     if $|S[l]| < q$ or $|N(u, H)| < p - l - 1$ then
20         Continue;
21     Construct subgraph $H'$ of H induced by $N(u, H)$;
22     LayerBasedListing($l + 1$, $H'$, $L \cup \{u\}$);
```

vertex ranks according to the 2-hop degree of a vertex (Line 5). Next, we construct the 2-hop graph H based on vertices in $U(G)$ (Line 6). In Line 7, we initialize a set of p empty arrays to store the common neighbors in G for vertices in $U(H)$. Finally, we enumerate all (p, q)-bicliques using LayerBasedListing (Line 8).

Generally, in LayerBasedListing, we recursively enumerate p-cliques on H, and simultaneously collect their common neighbors in G using vectors S. By combining the two parts together, we retrieve the (p, q)-bicliques.

Specifically, we use L to store the clique in H and start from depth 0 (i.e., $l = 0$). During the processing of LayerBasedListing, we first check if $l = p$ (Line 11), which implies that we have traversed p steps and a p-clique is found. We simply collect (p, q)-bicliques from L and each q-sized subset R of $S[l-1]$ (Lines 12-13). Otherwise, we iteratively select vertex $u \in U(H)$ to expand the clique L (Lines 14-22). We start by computing the common neighbors of u and previous vertices in L using $S[l]$. Particularly, if $l = 0$, we simply add all vertices in $N(u, G)$ to $S[l]$ (Line 16). Otherwise, $S[l]$ is computed by the intersection between $S[l-1]$ and $N(u, G)$ since $S[l-1]$ stores the common neighbors of existing vertices in L (Line 18). After that, we may be able to skip the current branch if the number of common neighbors is less than q (i.e., $|S[l]| < q$), or there is not enough vertices to expand L (i.e., $|N(u, H)| < p - l - 1$). Last, we construct subgraph H' of H induced by $N(u, H)$ and enter the sub-space by expanding L with u (Lines 21-22).

Example 7. *Continuing Example 6, we illustrate the overall running processing of BCList++. Since the vertices in $U(H)$ are u_2, u_1, u_3, u_0, u_4, we start from u_2. Thus, we have $S[0] = N(u_2, G) = \{v_0, v_1, v_2, v_3, v_4\}$, and the induced subgraph H' contains two isolated vertices u_1 and u_3 since $N(u_2, H) = \{u_1, u_3\}$ (See Figure 3). We search the sub-space by adding u_2 to L. Next, we consider u_1, and have that $S[1] = \{v_1, v_2, v_3\}$ by computing the intersection between $S[0]$ and $N(u_1, G)$. Meanwhile, H' is empty due to the fact that $N(u_1, H)$ is empty. After adding u_1 to L, we enter the new search space. Since we have $l = p = 2$ at this point, we begin to enumerate (p, q)-bicliques by L and $S[1]$, where $(\{u_1, u_2\}, \{v_1, v_2, v_3\})$ is found. Continuing this processing, we can find another (p, q)-biclique $(\{u_2, u_3\}, \{v_2, v_3, v_4\})$.*

Theorem 5. *BCList++ enumerates (p, q)-bicliques correctly.*

Efficient Implementation. BCList++ can be implemented efficiently using the following data structures and operations which are an adaption of the ones used in [6, 7] for listing cliques in generic graphs. Note that the only frequently constructed object is the DAG H in Line 6 and Line 21. This is because L in LayerBasedListing can be represented as an array of size p and S is a set of p arrays.

For each vertex $u \in U(H)$, we use an adjacency list $N(u, H)$ to store its out-neighbors. No other adjacency lists will be created during the processing BCList++. Specifically, given the current 2-hop graph H in the recursion, we make sure that the out-neighbors of any vertex in H always appear first in $N(u, H)$. Given H and a vertex u, the subgraph H' induced by $N(u, H)$ is built as follows.

- Assign each vertex a label initially set to 0.
- For each $v \in N(u, H)$, set its label to $l + 1$ if its current label is equal to l. It ensures that v is in the new DAG H' induced by $N(u, H)$.
- For each $v \in N(u, H)$, move all the neighbors in $N(v, H)$ with label equal to $l + 1$ in the first part of $N(v, H)$ (by swapping vertices) and compute the out-degree of vertex v in the new DAG H' to update $d(v)$. The first $d(v)$ vertices in $N(v, H)$ are thus the out-neighbors of v in H'.
- For each $v \in N(u, H)$, set its label back to l after finishing a recursion search branch.

4.3 Cost Analysis

In this section, we first analyze the time and space complexity of BCList++, and then introduce the *cost model* used by BCList++.

4.3.1 ***Time and Space Complexity Analysis.*** We focus on analyzing the time complexity of the main recursion procedure LayerBasedListing. To this end, we follow the methodology used in [6], which studies the parameterized complexity. Particularly, we use the concept of *arboricity* $a(H)$, which is defined as the minimum number of edge-disjoint spanning forests into which H can be decomposed.

Theorem 6. *The time complexity of BCList++ is $O(a(H)^{p-2}|E(H)|d_{max} + \triangle)$ where d_{max} is the maximum degree of a vertex in $U(G)$ and $\triangle$ is the result size.*

Theorem 7. *The space complexity of BCList++ is $O(|E(G)| + |E(H)|)$.*

4.3.2 ***Cost Model Analysis.*** In BCList++, the first step is to choose the layer with the least computation cost to construct the 2-hop graph H (Lines 1-3 in Algorithms 3). Theorem 6 implies that we need to compute the arboricity $a(H)$ of H, which is suggested

Algorithm 4: DegreeEstimator(G, p, q)

```
Input  : G : a bipartite graph G
         p, q : two parameters
Output : An estimate of D
Choose a vertex $u$ from $U(G)$ uniformly at random;
$D_u \leftarrow$ TwoHopDegree($u$);
return $n \cdot D_u$
procedure TwoHopDegree($u$)
$D_u \leftarrow 0$;
$C \leftarrow hashmap$;                    /* initialized with zero */;
for each $v \in N(u, G)$ do
    for each $w \in N(v, G)$ do
        if $u \neq w$ then
            $C[w] \leftarrow C[w] + 1$;
for each $w \in C$ do
    if $C[w] \geq q$ then
        $D_u \leftarrow D_u + 1$;
return $D_u$
```

as an open problem [13]. Harold [12] proposes a parametric flow based method to compute the arboricity of an undirected graph $G = (V, E)$ with a time complexity of $O(|V||E| \log(|V|^2/|E|))$, which is, however, computationally costly for large graphs.

In practice, we observe that the average degree is a good substitution for the arboricity, since both of them are used to measure the "density" of a graph. Therefore, we use average degree to estimate the computation cost of BCList++ due to its high computation efficiency. A straightforward way is to directly compute the 2-hop neighbors for vertices in both layers using Algorithm 1. However, Theorem 1 shows that it is expensive to compute the exact 2-hop degree for all vertices in G.

In the following, we resort to a random sampling method to approximate the total 2-hop degree D for a given layer, say $U(G)$. The intuition is to use the 2-hop degree of a sampled vertex to estimate the total 2-hop degree of verties in $U(G)$. Since the sampled local subgraph is typically much smaller than the original graph G, it is cost-saving to compute the 2-hop degree of sampled vertex instead.

Degree Estimation. Algorithm 4 illustrates the details of our degree estimation method. Note that we only introduce the computation for layer $U(G)$, and computation for $V(G)$ is similar. In each sampling, we choose a vertex u from $U(G)$ uniformly at random (Line 1). We then compute the 2-hop degree D_u of u using procedure TwoHopDegree (Line 2). Here, we skip introducing the details since it is basically a subroutine of Algorithm 1. Last, we return $n \cdot D_u$ as an estimation for the total 2-hop degree of vertices in $U(G)$, where $n = |U(G)|$ (Line 3). It is easy to verify that the time complexity of DegreeEstimator is $O(d(u)d_{max})$, where d_{max} is the maximum degree of vertices in $V(G)$.

Next, we show that Algorithm 4 yields an unbiased estimation for the total degree of H. Let Y denote the value returned by Algorithm 4. Let D denote the true value of total degree of H. Let p_s denote the number of degree pairs that attaching the same vertex. We have the following lemma.

Lemma 4. $\mathbf{E}[Y] = D$, *and* $\mathbf{Var}[Y] \leq n(D + p_s) - \binom{D}{2}$.

Let Z be the average of r independent instances of Y. According to the fact that $\mathbf{Var}[Z] = \mathbf{Var}[Y]/r$ and Chebyshev's inequality, we have $\mathbf{Pr}[|Z - D| \geq \epsilon D] \leq \frac{\mathbf{Var}[Z]}{\epsilon^2 D^2} = \frac{\mathbf{Var}[Y]}{r\epsilon^2 D^2}$. Next, we show that by running Algorithm 4 multiple independent instances, we can obtain an (ϵ, δ)-estimator using standard method.

Lemma 5. *There is an algorithm that runs* $r = \frac{n}{\epsilon^2 \delta}(\frac{1}{D} + \frac{p_s}{D^2}) - \frac{1}{2\epsilon^2 \delta}$ *independent times of Algorithm 4 and provides an* (ϵ, δ)-*estimator of* D.

Overall Cost Model. We now proceed to estimate the overall computational cost of LayerBasedListing. According to Theorem 6, for layer $U(G)$, we have

$$\text{Cost}(U(G), p) = (D/|U(G)|)^{p-2} D d_{max} \tag{1}$$

where D is estimated by Algorithm 4, and d_{max} is the maximum degree of vertices in $U(G)$. The cost on $V(G)$ can be computed similarly. We omit $\triangle$ for both $Cost(U(G), p)$ and $Cost(V(G), q)$.

5 OPTIMIZATIONS

In this section, we develop optimizations to further boost the performance of our proposals.

5.1 Graph Reduction

We proceed to show how to reduce the bipartite graph by exploiting some properties of our problem.

Core Reduction. To reduce the size of the bipartite graph, a promising way is to remove vertices having degrees that are small enough. This is because a vertex u contributing a (p, q)-biclique must have degree at least q if $u \in U(G)$, otherwise p. We can repeat this operation until all remaining vertices satisfy this condition. Particularly, the remaining subgraph can be formally formulated by the so called (α, β)-core, which is defined as below [9, 22].

Definition 7 (**(α, β)-core**). *Given a bipartite graph* $G = (U, V, E)$, *and two integers* α *and* β, *the* (α, β)-*core of* G, *denoted by* $C_{\alpha,\beta}(G)$, *is a maximal subgraph of* G *induced by two vertex sets* $U_C \subseteq U$ *and* $V_C \subseteq V$, *in which all vertices in* U_C *have degree at least* α *and all vertices in* V_C *have degree at least* β, *i.e.,* $\forall u \in U_C, d(u) \geq \alpha \wedge \forall v \in V_C, d(v) \geq \beta$.

Given a bipartite graph G, the state-of-the-art algorithm computes its (α, β)-core in linear time (i.e., $O(|E(G)|)$) [9]. Intuitively, it computes (α, β)-core by iteratively removing vertices in $U(G)$ with degree smaller than α and vertices in $V(G)$ with degree smaller than β until no more vertices can be removed. We omit the details of the algorithm for space constraint. Based on the (α, β)-core, we derive the following lemma.

Lemma 6. *Given a bipartite graph* $G = (U, V, E)$, *and two integers* p *and* q, *let* C *be the* (q, p)-*core of* G. *Then, for any* (p, q)-*biclique* $B(L, R)$ *exists in* G, B *must be in* C, *i.e.,* $\forall B(L, R) \subseteq G$, *we have* $B \subseteq C$.

5.2 Parallelization

One appealing property of BCList is that it can enumerate (p, q)-bicliques in parallel when multiple threading is available. That is because the entire search space of (p, q)-bicliques is materialized by

Algorithm 5: BCList++ IN PARALLEL

```
1 Line 1-Line 7 of Algorithm 3;
2 for each $u \in U(H)$ in parallel do
3     Construct subgraph $H'$ of $H$ by $N(u, H)$;
4     LayerBasedListing(1, $H'$, $\{u\}$);
```

a recursion tree, in which the answers fall only in the leaf nodes. BCList can use a thread to process a subtree of the recursion tree to enumerate a sub-space independently.

Our empirical studies show that BCList++ is much more efficient and scalable on large datasets. Therefore, in this paper, we focus on parallelizing BCList++. Particularly, we initiate a thread for each subgraph H' of the 2-hop graph H induced by each vertex $u \in U(H)$. Algorithm 5 illustrates the details of parallel version of BCList++. Specifically, we only need to replace Line 8 of Algorithm 3 with Lines 2-4 of Algorithm 5.

6 EXPERIMENTAL STUDY

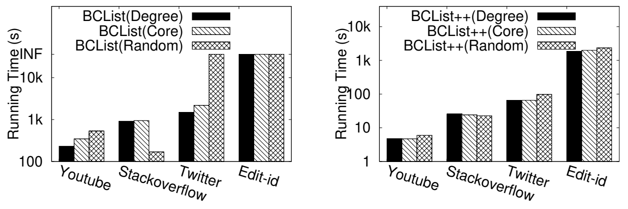
In this section, we empirically evaluate the performance of the proposed techniques. All experiments are conducted on PCs with Intel Xeon 2 × 2.4GHz CPU containing 40 cores and 128GB RAM running Ubuntu 20.04.2 LTS. Unless otherwise specified, we run algorithms against a single core. We terminate an algorithm if the running time is more than 10 hours which is denoted as INF.

6.1 Experimental Setup

Algorithms. In the experiments, we evaluate the following algorithms for counting by default.

- **BCList**. The baseline method proposed in Section 3.
- **BCList++**. The advanced approach devised in Section 4.
- **PMBE**. Adapted algorithm from the state-of-the-art for maximal biclique enumeration proposed in [1].
- **BFC-VP^{++}**. The state-of-the-art butterfly counting algorithm proposed in [42].
- **BFC**. Butterfly counting algorithm proposed in [30].

It is easy to verify that a (p, q)-biclique must be contained by maximal bicliques with at least p and q vertices in the left and right side, respectively. With this property, we can immediately come up with another baseline method as follows. First, applying the off-the-shelf maximal biclique enumeration algorithms to list all maximal bicliques with size constraint, and then enumerating all (p, q)-bicliques from the obtained maximal bicliques and removing the duplicates. The state-of-the-art algorithm, called PMBE [1], offers a threshold based solution for maximal biclique enumeration, which is much more efficient than the version without threshold. For presentation convenience, we use the name of PMBE to denote the adapted algorithm for our problem. We obtain the source codes of PMBE from the authors of [1]. To ensure the fairness, in our experiments, we *only* record the time cost of PMBE for enumerating maximal bicliques, while ignoring the time for enumerating (p, q)-bicliques and removing duplicates. For BFC-VP^{++} and BFC, we obtain the source code from the authors of [42] and [30], respectively. All algorithms are implemented in standard C++ with STL library support and compiled with GNU GCC, except PMBE, which



(a) Tuning BCList (b) Tuning BCList++

Figure 4: Effect of vertex orderings (p = 4 and q = 4).

is implemented in Java and the JVM maximum heap size is set to large enough for all datasets.

Datasets. We use 16 real datasets selected from different domains with various data properties. All datasets are obtained from KONECT[4], some of which are used to evaluate the algorithms for related problems [23, 30, 42, 47]. The detailed characteristics of the 16 datasets are shown in Table 3. We choose 4 representative datasets from Table 3 as default datasets, including *Youtube*, *Stackoverflow*, *Twitter*, and *Edit-id*, which cover different types of datasets, and various graph scales.

Queries. To better evaluate our proposals, we generate multiple (p, q) settings by fixing the value of $p + q = 8$ and varying values of p and q from 2 to 6. Unless otherwise specified, experiments are conducted with $p = 4$ and $q = 4$ by default.

6.2 Performance Tuning

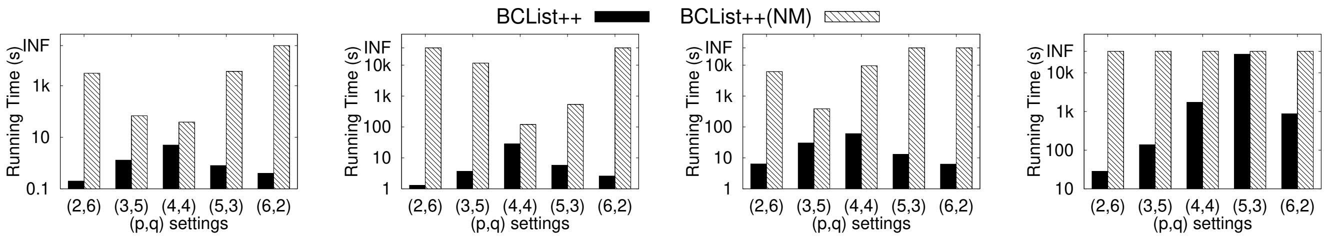
Exp-1: Effect of vertex ordering. We start by evaluating the effect of vertex ordering. We evaluate three vertex orderings, including degree, core, and random. Note that, we use the combination of algorithm name and order name. For example, BCList(Degree) stands for BCList using degree ordering. Figure 4(a) shows the results for BCList. Generally, both BCList(Degree) and BCList(Core) run quite stably compared with BCList(Random), and BCList(Degree) is slightly faster than BCList(Core). Figure 4(b) reports the results for BCList++. On one hand, we observe that BCList++(Degree) achieves the best performance under most datasets. On the other hand, the performance gap between BCList++(Degree) and others is very marginal. This implies that the performance improvement brought by vertex ordering is limited. In the following experiments, we use degree vertex order for both BCList and BCList++.

Exp-2: Effect of cost model. We evaluate the effectiveness of the cost model, which is proposed in Section 4.3. By BCList++(NM), we denote BCList++ without the cost model. The number of sampling iterations is set to $0.01 \times |U(H)|$, where $U(H)$ is the vertex set in the selected layer. Figure 5 reports the experiment results. Not surprisingly, the cost model has a huge impact on the performance of BCList++. For example, BCList++(NM) cannot even handle the smallest dataset *Youtube* when $p = 6$ and $q = 2$, while BCList++ can finish in less than one second. It is generally observed that BCList++ can achieve at least one order of magnitude performance improvement by using the cost model. Besides, the performance improvement enlarges quickly as the difference between p and q increases. This is because the time complexity of BCList++ is exponential to the value of p or q. Overall, our cost model can judiciously choose the layer with the least cost by taking both the graph size and values of p and q into consideration.

[4]http://konect.cc/

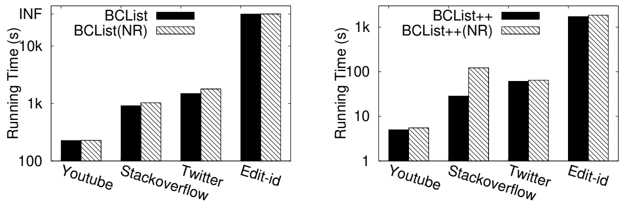
Table 3: Some characteristics of datasets.

Dataset	Category	$\|U\|$	U Type	$\|V\|$	V Type	$\|E\|$	E Type	$\overline{d}_U$	$\overline{d}_V$	$\overline{d}$
Record	Affiliation	168, 337	Artist	18, 421	record	233, 286	Membership	1.39	12.66	2.50
Youtube	Affiliation	94, 238	User	30, 087	Group	293, 360	Membership	3.11	9.75	4.72
Bookcrossing	Rating	77, 802	User	185, 955	Book	433, 652	Rate	5.57	2.33	3.29
Github	Authorship	56, 519	User	120, 867	Project	440, 237	Membership	7.79	3.64	4.96
CiteSeer	Authorship	105, 353	Author	181, 395	Publication	512, 267	Authorship	4.86	2.82	3.57
Stackoverflow	Rating	545, 196	User	96, 680	Post	1, 301, 942	Favorite	2.39	13.47	4.06
Actor-movie	Affiliation	127, 823	Actor	383, 640	Movie	1, 470, 404	Appearance	11.50	3.83	5.75
Twitter	Interaction	175, 214	User	530, 418	Hashtag	1, 890, 661	Usage	10.80	3.56	5.36
IMDB	Affiliation	303, 617	Actor	896, 302	Movie	3, 782, 463	Appearance	12.46	4.22	6.30
Edit-en	Authorship	18, 038	User	2, 192, 849	Article	4, 129, 231	Edit	363.75	2.99	5.94
Edit-fr	Authorship	6, 666	User	2, 402, 444	Article	4, 408, 423	Edit	978.60	2.72	5.42
Amazon	Rating	2, 146, 057	User	1, 230, 915	Product	5, 743, 258	Rate	2.72	4.74	3.46
Edit-id	Authorship	125, 481	User	2, 183, 494	Article	6, 126, 592	Edit	48.82	2.81	5.31
Edit-fa	Authorship	134, 986	User	3, 597, 380	Article	10, 011, 147	Edit	74.16	2.78	5.36
Edit-ar	Authorship	209, 373	User	2, 943, 711	Article	10, 489, 226	Edit	50.10	3.56	6.65
DBLP	Authorship	1, 953, 085	Author	5, 624, 219	Publication	12, 282, 059	Authorship	6.29	2.18	3.24



(a) *Youtube* (b) *Stackoverflow* (c) *Twitter* (d) *Edit-id*

Figure 5: Effect of cost model.



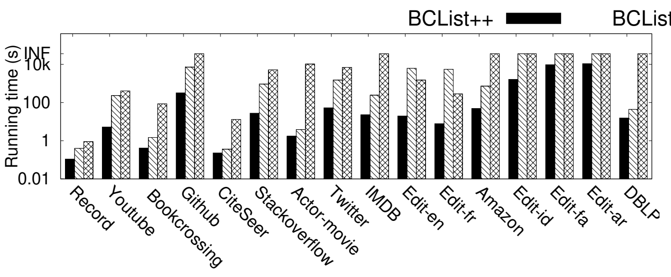
(a) Tuning BCList (b) Tuning BCList++

Figure 6: Evaluating graph reduction (p = 4 and q = 4).

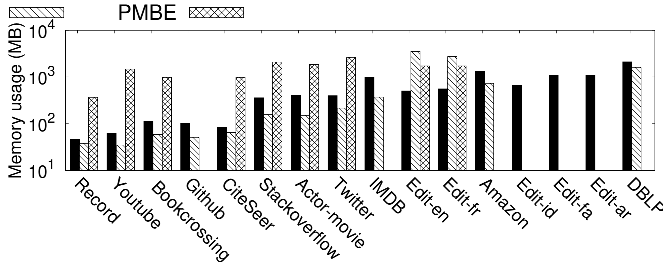
Exp-3: Evaluating graph reduction. Figure 6 evaluates the effectiveness of the graph reduction techniques proposed in Section 5.1, In particular, Figure 6(a) reports the experimental results for BCList, where BCList(NR) stands for BCList without graph reduction techniques. We observe that, by using graph reduction, BCList can achieve roughly 10% of performance improvement on *Stackoverflow* and *Twitter*. Figure 6(b) reports the experimental results for BCList++. It is observed that BCList++ runs 4 times faster on *Stackoverflow* than BCList++(NR). However, the benefit brought by graph reduction seems limited for both BCList and BCList++ on *Youtube* and *Edit-id*. The reason is that the most time-consuming part is the recursion search for enumerating (p, q)-bicliques.

6.3 Performance Evaluation

Exp-4: Experiments over all datasets. We now proceed to compare BCList++ with the two baseline algorithms BCList and PMBE on all 16 datasets in terms of both processing time and memory usage. **Processing Time**. It is reported in Figure 7(a) that BCList++ significantly outperforms the compititors on most datasets regarding processing time. Among the two baseline methods, BCList runs much faster than PMBE does on most datasets. For example, it is at least 2 orders of magnitude faster than PMBE on *Actor-movie*, *IMDB*, or *DBLP*. We observe that BCList is beaten by PMBE on two datasets *Edit-en* and *Edit-fr*. The reason is that these datasets are rather unbalanced, and the number of 2-hop neighbors of vertices on the "heavy" layer is quite large. Generally, PMBE can only process relatively small datasets, e.g., *Record*, *Youtube*, and *CiteSeer*, while neither of the two baseline methods can handle large datasets, e.g., *Edit-id*, *Edit-fa*, and *Edit-ar*. PMBE even runs into INF on *Github*, which only contains 440 thousands of edges. From the results shown in Figure 7(a), we notice that BCList++ is much more efficient and scalable than its compititors, and outperforms the compititors by more than one order of magnitude on datasets such as *Youtube*, *Github*, *Stackoverflow*, and *Twitter* etc. This is because BCList++ adopts an efficient layer based search strategy equipped with a highly effective cost model as shown Figure 5. **Memory Usage**. Figure 7(b) reports the memory usage of the three algorithms. Note that we do not show the memory usage for an algorithm if its running time is INF on the corresponding dataset. In general, BCList consumes the least amount of memory, and BCList++ consumes a bit more memory than BCList. This is because we implement BCList++ with extra arrays to avoid creating the subgraphs frequently. The memory usage of PMBE is much larger than that of our algorithms. For example, PMBE could consume up to an

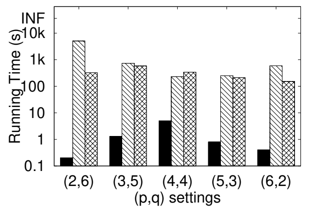


(a) Running time

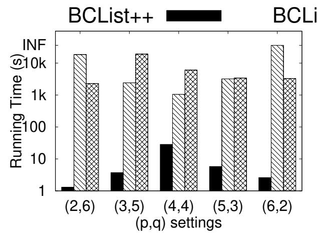


(b) Memory usage

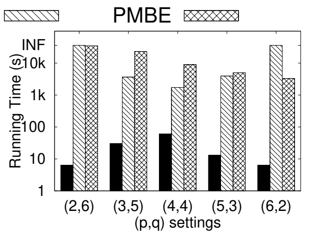
Figure 7: Evaluation over all datasets ($p = 4$ and $q = 4$).



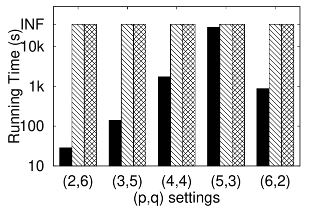
(a) *Youtube*



(b) *Stackoverflow*



(c) *Twitter*



(d) *Edit-id*

Figure 8: Varying values of p and q.

order of magnitude more memory on datasets such as *Youtube* and *CiteSeer*. The reason is that PMBE needs to build a heavy index structure to facilitate the enumeration. The only exceptions occur on *Edit-en* and *Edit-fr* where BCList consumes the most amount of memory. This is still because the amount of 2-hop neighbors of vertices on the "heavy" layer of such unbalanced bipartite graphs is large, which however can be avoided by BCList++ by selecting the other layer using the cost model.

Exp-5: Varying values of p and q. To evaluate the effect of p and q values, we conduct experiments on the default (p, q) settings where $p + q = 8$, and p and q vary from 2 to 6. Figures 8 reports the experiment results. It is reported that, compared to the baseline methods, BCList++ is much more friendly to queries when the ratio between p and q is large. For example, on *Twitter* shown in Figure 8(c), when $p = 2$ and $q = 6$, BCList runs into INF and PMBE spends near 10 hours, while it only takes less than 7 seconds for BCList++ to finish. This is because BCList++ is equipped with a cost model, which can judiciously select the layer with the least computation cost as our search layer by considering the values of p and q. Even under queries with the same values of p and q (i.e., $p = q = 4$), BCList++ is still more than one order of magnitude faster than its competitors on all datasets.

Exp-6: Evaluating butterfly counting. In this experiment, we evaluate the performance of algorithms for butterfly counting (i.e., $p = 2$ and $q = 2$). PMBE is excluded from the evaluation because it failed to give response within 10 hours on most of datasets. Figure 9 shows the experiment results. Interestingly, we observe that although BCList++ is not particularly designed for butterfly counting, it is still quite competitive compared with the state-of-the-art butterfly counting algorithms, such as BFC-VP^{++} and BFC. As we can see, BCList++ even ranks the first on dataset *Youtube* and *Twitter*. BCList++ is relatively slow on *Edit-id*. This reason is that *Edit-id* is a large graph, and BCList++ needs to execute the step of cost estimation, which deteriorates the performance of BCList++.

We conduct experiments on evaluating the scalability and parallelization of our proposals in the full version of the paper [46].

6.4 Case Study

(p, q)-biclique can be used to optimize the efficiency of Graph Neural Network (GNN) which has been one of the most successful and extensively studied research topics in recent years [14, 38, 48]. A pivotal operation in a GNN is to recursively aggregate information from vertices' neighbors in graph. A naive method simply propagates information on each pair of vertices separately. Observing that vertices in a graph usually share many neighbors, this naive method leads to redundant computations. To improve the computation efficiency, Jia *et al.* [16] propose a new GNN representation technique called Hierarchically Aggregated computation Graphs (HAGs) aiming at reducing the number of sum operations. Interestingly, we observe that (p, q)-biclique enumeration can be applied on the task of GNN information aggregation.

Example 8. *A running example for information aggregation in a GNN under different methods is shown in Figure 10, where Figure 10(a) shows the input graph. Figure 10(b) shows just a single layer of the GNN-graph of the input graph in Figure 10(a). For instance, for vertex A, its new activation h_A^k at layer k is computed by aggregating its neighbors' activations h_B^{k-1}, h_C^{k-1}, and itself h_A^{k-1} at layer $k-1$. Thus,* 2 *sum operations occur on A. The new activations of other vertices can be computed similarly. Clearly, the naive method generates* 8 *sum operations in total. In Figure 10(c), HAG considers merging the common neighbors of A and C, including A, B, and C, where* 2 *sum operations are needed for merging them to a_1. Similarly, we have that* 1 *and* 2 *sum operations occur on a_2 and B, respectively. Therefore, the total number of sum operations for HAG is* 5*. Last, in our (p, q)-biclique*

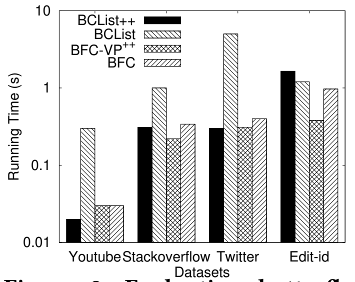


Figure 9: Evaluating butterfly counting

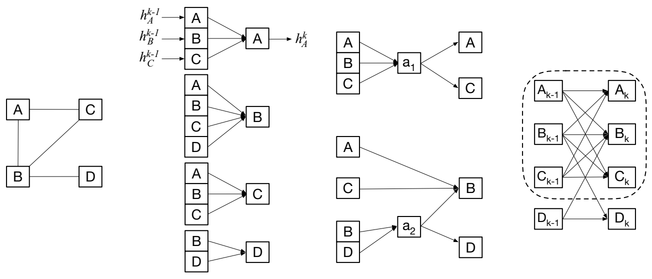


Figure 10: Information aggregation in GNN.

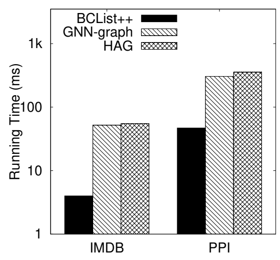


Figure 11: Evaluating GNN information aggregation

based method, the two sides of vertices of the aggregation layer are naturally considered as a bipartite graph (Figure 10(d)), where both sides contain exactly the same number of vertices as input graph. In the $(3, 3)$*-biclique (marked by dashed rectangle),* 2 *sum operations are required, that is merging* A_{k-1}, B_{k-1}, *and* C_{k-1} *together. Besides, both* B_k *and* D_k *need an extra sum operation. Thus, our (p, q)-biclique based method needs the least amount of* 4 *sum operations.*

We conduct experiments on two datasets, namely IMDB (with 19,502 vertices and 197,806 edges) and PPI (with 56,944 vertices and 1,612,348 edges), both of which are employed to evaluate the performance of HAG [16]. The experiment results shown in Figure 11 report that our (p, q)-biclique based method achieves the best performance when (p, q) are $(5, 10)$ and $(4, 10)$ on IMDB and PPI, respectively, and outperforms the competitors by near an order of magnitude for GNN information aggregation. We discuss the algorithm applying details in the full version of the paper [46].

7 RELATED WORK

Motif Counting in Bipartite Graphs. As a special case of our problem, butterfly counting has attracted many research efforts recently. Wang *et al.* [40] for the first time present exact algorithm for butterfly (rectangle) counting in a bipartite graph, which avoids enumerating all the butterflies. First, a layer is selected at random. Then, for each vertex u in the selected layer, we compute its 2-hop neighbors, and for each 2-hop neighbor w, we count the number of common neighbors between u and w denoted as n_{uw}. The number of butterflies starting from u is simply $\binom{n_{uw}}{2}$. Finally, we add all the counts together, and the added counts divided by two is the total number of butterflies. Under the same computation paradigm, Sanei-Mehri *et al.* [30] propose a layer-based method to improve the efficiency of [40] by selecting the layer with the least computation cost. Later, Wang *et al.* [42] propose a vertex priority based method to further accelerate the computation. Apart from these exact algorithms, research efforts have also been devoted to approximate approaches [20, 30, 34]. Recently, Yang *et al.* [47] investigate the problem of bi-triangle counting in bipartite graphs, where a bi-triangle is defined as a 6-cycle. Butterfly based bitruss decomposition has also been studied in the literature [43, 44].

Maximal Biclique Enumeration in Bipartite Graphs. A closely related problem is maximal biclique enumeration in bipartite graphs. A biclique is said to be maximal if it is not contained in any larger bicliques. David Eppstein [10] provides a linear algorithm to list maximal bicliques in any graph of bounded arboricity (i.e., $a(G) = O(1)$). In [29], maximal bicliques are enumerated by exhaustively enumerating subsets of vertices in one layer, and obtaining the vertices in other layer as their common neighbors, and then checking the maximality of the obtained bicliques. Li *et al.* [19] enumerate the maximal bicliques by using efficient algorithms for mining closed patterns, which have been extensively studied in the data mining field. Inspired by the classical BK algorithm [4], Zhang *et al.* [49] propose algorithm iMBEA, which combines backtracking with branch-and-bound framework, where useful pruning techniques are employed to filter out the branches that cannot lead to maximal bicliques. The state-of-the-art approach [1] utilizes pivot pruning to improve the efficiency of maximal biclique enumeration. A variant problem is maximum biclique search, which has also been extensively studied recently [5, 11, 23, 32, 33].

Listing k-Cliques in Unipartite Graphs. The problem of listing k-cliques in unipartite graphs has a long research history and a wide range of applications [2, 28, 31]. The seminal work is the algorithm of Chiba and Nishizeki [6], which provides an efficient implementation of a branch-and-bound approach. Under the same framework, recently, Danisch *et al.* [7] propose a core ordering based method to resolve this problem. The advantages of their proposal are demonstrated by both theoretical and empirical analysis. Other algorithms initially devised for counting and listing maximal cliques can also be adapted to deal with k-clique listing [25, 36]. Recently, the problem of k-clique densest subgraph search has received increasing attention [26, 31, 35, 37].

8 CONCLUSION

In this paper, we study the problem of (p, q)-biclique counting and enumeration for large sparse bipartite graphs. To efficiently solve this problem, we propose a competitive branch-and-bound baseline method, called BCList, which offers a useful computation framework to the problem. To improve the computation efficiency, we propose an advanced approach, namely BCList++, by anchoring the search on a single layer of the bipartite graph. Effective cost model and optimization techniques are proposed to enhance the performance of BCList++. Extensive experiments on 16 real datasets demonstrate the superior performance of BCList++ compared with the baseline methods.

ACKNOWLEDGMENTS

This work was supported by NSFC 62002108, NSF of Hunan Province 2021JJ40123, the ARC DP200101116, GRFs HKBU12201518, and NSF of Shandong Province ZR2019LZH006. We also thank Byron Choi for contributing valuable advice on writing of the paper.

REFERENCES

[1] A. Abidi, Rui Zhou, Lu Chen, and Chengfei Liu. 2020. Pivot-based Maximal Biclique Enumeration. In *IJCAI*. 3558–3564.

[2] A. R. Benson, D. F. Gleich, and J. Leskovec. 2016. Higher-order organization of complex networks. *Science* (2016), 163–166.

[3] S. P. Borgatti and M. G. Everett. 1997. Network analysis of 2-mode data. *Social networks* (1997), 243–269.

[4] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* (1973), 575–577.

[5] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2021. Efficient Exact Algorithms for Maximum Balanced Biclique Search in Bipartite Graphs. In *SIGMOD*. 248–260.

[6] N. Chiba and T. Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM Journal on computing* (1985), 210–223.

[7] M. Danisch, O. Balalau, and M. Sozio. 2018. Listing k-cliques in sparse real-world graphs. In *WWW*. 589–598.

[8] Hongbo Deng, Michael R Lyu, and Irwin King. 2009. A generalized co-hits algorithm and its application to bipartite graphs. In *SIGKDD*. 239–248.

[9] Danhao Ding, Hui Li, Zhipeng Huang, and Nikos Mamoulis. 2017. Efficient Fault-Tolerant Group Recommendation Using Alpha-beta-core. In *CIKM*. 2047–2150.

[10] David Eppstein. 1994. Arboricity and Bipartite Subgraph Listing Algorithms. *Inform. Process. Lett.* (1994).

[11] Q. Feng, S. Li, Z. Zhou, and J. Wang. 2017. Parameterized algorithms for edge biclique and related problems. *Theoretical Computer Science* (2017).

[12] Harold N. Gabow. 1999. Algorithms for Graphic Polymatroids and Parametric s-Sets. *Journal of Algorithms* (1999), 48–86.

[13] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. 1985. A fast parametric maximum flow algorithm and applications. *SIAM Journal on computing* (1985), 30–55.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*. 1024–1034.

[15] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In *SIGKDD*.

[16] Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. 2020. Redundancy-Free Computation for Graph Neural Networks. In *SIGKDD*.

[17] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and V.S. Subrahmanian. 2018. REV2: Fraudulent User Prediction in Rating Platforms. In *WSDM*. 333–341.

[18] M. Latapy, C. Magnien, and N. Del Vecchio. 2008. Basic notions for the analysis of large two-mode networks. *Social networks* (2008), 31–48.

[19] 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. *TKDE* (2007), 1625–1637.

[20] 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. *TKDE* (2021), 1–1.

[21] Pedro G. Lind, Marta C. G., and Hans J. Herrmann. 2005. Cycles and clustering in bipartite networks. *Physical Review E* (2005), 814–818.

[22] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (α, β)-core computation: An index-based approach. In *WWW*. 1130–1141.

[23] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum biclique search at billion scale. In *PVLDB*. 1359–1372.

[24] Ziyi Ma, Yuling Liu, Yikun Hu, Jianye Yang, Chubo Liu, and Huadong Dai. 2021. Efficient maintenance for maximal bicliques in bipartite graph streams. *WWWJ* (2021).

[25] Kazuhisa Makino and Takeaki Uno. 2004. New algorithms for enumerating all maximal cliques. In *Algorithm Theory-SWAT*. 260–272.

[26] 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*.

[27] T. Opsahl. 2013. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks* (2013), 159–167.

[28] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* (2005), 814–818.

[29] M. J. Sanderson, A. C. Driskell, R. H. Ree, O. Eulenstein, and S. Langley. 2003. Obtaining maximal concatenated phylogenetic data sets from large sequence databases. *Molecular biology and evolution* (2003), 1036–1042.

[30] S. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura. 2018. Butterfly counting in bipartite networks. In *KDD*. 2150–2159.

[31] A. E. Sariyuce, C. Seshadhri, A. Pinar, and U. V. Catalyurek. 2015. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*. 927–937.

[32] E. Shaham, H. Yu, and X. Li. 2016. On finding the maximum edge biclique in a bipartite graph: a subspace clustering approach. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. 315–323.

[33] S. Shahinpour, S. Shirvani, Z. Ertem, and S. Butenko. 2017. Scale reduction techniques for computing maximum induced bicliques. *Algorithms* (2017).

[34] A. Sheshbolouki and M. T. Özsu. 2021. sGrapp: Butterfly Approximation in Streaming Graphs. *arXiv preprint arXiv:2101.12334* (2021).

[35] Bintao Sun, M. Danisch, TH Chan, and M. Sozio. 2020. KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs. In *PVLDB*. 1628–1640.

[36] UNO Takeaki. 2012. Implementation issues of clique enumeration algorithm. *Special issue: Theoretical computer science and discrete mathematics, Progress in Informatics* (2012), 25–30.

[37] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *WWW*. 1122–1132.

[38] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. 2017. IGraph attention networks. In *ICLR*.

[39] Haibo Wang, Chuan Zhou, Jia Wu, Weizhen Dang, Xingquan Zhu, and Jilong Wang. 2018. Deep Structure Learning for Fraud Detection. In *ICDM*. 567–576.

[40] J. Wang, A. W. C. Fu, and J. Cheng. 2014. Rectangle Counting in Large Bipartite Graphs. *IEEE International Congress on Big Data*, 17–24.

[41] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR*. 501–508.

[42] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex priority based butterfly counting for large-scale bipartite networks. In *PVLDB*. 1139–1152.

[43] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient bitruss decomposition for large-scale bipartite graphs. In *ICDE*. IEEE, 661–672.

[44] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2021. Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs. *VLDBJ* (2021), 1–24.

[45] Kai Wang, Zhang Wenjie, Xuemin Lin, Ying Zhang, Lu Qin, and Yuting Zhang. 2021. Efficient and Effective Community Search on Large-scale Bipartite Graphs. In *ICDE*. IEEE, 85–96.

[46] Jianye Yang, Yun Peng, and Wenjie Zhang. 2021. *(p,q)-biclique counting and enumeration for large sparse bipartite graphs*. https://github.com/Jianye1hnu/bclist_vldb/blob/main/bclist_full.pdf

[47] Yixing Yang, Yixiang Fang, M. E. Orlowska, Wenjie Zhang, and Xuemin Lin. 2021. Efficient bi-triangle counting for large bipartite networks. In *PVLDB*. 984–996.

[48] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*. 4801–4811.

[49] Yun Zhang, C. A. Phillips, G. L. Rogers, E. J. Baker, E. J. Chesler, and M. A. Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics* (2014), 1–18.