Scaling Biclique Percolation for Community Detection in Large Bipartite Networks

Anonymous Author(s)

Abstract

Community detection, which uncovers closely connected vertex groups in networks, is vital for applications in social networks, recommendation systems, and beyond. Real-world networks often have bipartite structures (vertices in two disjoint sets with inter-set connections), creating unique challenges on specialized community detection methods. Biclique percolation community (BCPC) is widely used to detect cohesive structures in bipartite graphs. A biclique is a complete bipartite subgraph, and a BCPC forms when maximal bicliques connect via adjacency (sharing an (α, β)-biclique). Yet, existing methods for BCPC detection suffer from high time complexity due to the potentially massive maximal biclique adjacency graph (MBAG). To tackle this, we propose a novel partial-BCPC based solution, whose key idea is to use partial-BCPC to reduce the size of the MBAG. A partial-BCPC is a subset of BCPC. Maximal bicliques belonging to the same partial-BCPC must also belong to the same BCPC. Therefore, these maximal bicliques can be grouped as a single vertex in the MBAG, significantly reducing the size of the MBAG. Furthermore, we move beyond the limitations of MBAG and propose a novel BCPC detection approach based on (α, β)-biclique enumeration. This approach detects BCPC by enumerating all (α, β)-bicliques and connecting maximal bicliques sharing the same (α, β)-biclique, which is the condition for maximal bicliques to be adjacent. It also leverages partial-BCPC to significantly prune the enumeration space of (α, β)-biclique. Experiments show that our methods outperform existing methods by nearly three orders of magnitude.

1 Introduction

Community detection aims to uncover groups of closely connected vertices in a network and is a core task in network analysis [10]. It plays a key role in applications such as social networks [26], biological systems [25] and recommendation systems [7]. In many real-world scenarios, networks are naturally modeled as bipartite graphs, represented by $G = (U, V, E)$, where vertices are divided into two disjoint sets U and V, and connections only occur between U and V. Representative examples include user–item networks in recommendation systems [22] and author–paper networks in bibliometrics [4]. This structure introduces new challenges for community detection, motivating the development of specialized methods for bipartite graphs.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The biclique percolation community (BCPC) is a widely studied approach for detecting cohesive structures in bipartite graphs [6, 11–15, 33]. A biclique is a complete bipartite subgraph, and a biclique percolation community is defined as a maximal set of bicliques that are connected through *adjacency*. In the (α, β)-biclique percolation community, two bicliques are considered *adjacent* if they share at least α vertices on one side and β on the other (share an (α, β)-biclique). Biclique percolation community (BCPC) has diverse applications across bipartite network scenarios. In social networks (users-entities as vertices, interaction as edges), BCPC uncovers interest-based user groups to support interest or user recommendation [11]. In Wikipedia networks (articles-editors as vertices, editorial activities as edges), BCPC identifies clusters, revealing topic-driven editor aggregation and coordinated content creation [13]. For internet resource access (resources-visitors as vertices), BCPC is used to analyze student-course resource relationships and their temporal evolution [12, 33]. In enterprise heterogeneous graphs (hosts-users as vertices, interactions as edges), BCPC is used to detect clusters to aid abnormal user behavior identification and malicious activity tracing for network security [15]. BCPC allows overlaps between clusters, and this property is crucial because real-world users or entities are likely to belong to multiple communities. This property has also been extensively studied [11–13, 15].

Despite its widespread practical utility, the BCPC method faces significant computational challenges due to its high time complexity. To the best of our knowledge, the existing BCPC methods are all based on [14]. Its framework was later implemented in [27] and used to build indexes for personalized search problems [6]. The basic idea is based on *maximal biclique adjacency graph* (MBAG). This method first enumerates all maximal bicliques and then uses them as vertices to construct a maximal biclique adjacency graph. In this graph, there is an edge between two vertices if and only if the overlapping part of the corresponding two maximal bicliques contains an (α, β)-biclique. The connected components in the adjacency graph correspond one-to-one with the (α, β)-BCPC. However, real-world bipartite graphs may contain a large number of maximal bicliques (up to $2^{n/2}$ theoretically [9]), and the adjacency graph formed by these maximal bicliques could contain a significant number of edges (up to 10^{11}, see Table 1). Traversing such a large-scale adjacency graph is highly time-consuming.

To overcome the limitations of existing methods, we first propose a partial-BCPC based solution, which focuses on reducing the size of the MBAG. The key idea is to leverage the prefixes of the maximal biclique enumeration tree during the enumeration process. Specifically, we introduce a novel concept of partial-BCPC, which is computed on-the-fly while enumerating maximal bicliques. During the enumeration, if a prefix of the enumeration tree contains at least α vertices from the vertex set U and β vertices from the vertex set V, we can determine that all maximal bicliques under this branch belong to the same BCPC. These bicliques collectively form an incomplete BCPC, referred to as a partial-BCPC. Maximal bicliques in the same partial-BCPC can be regarded as a single vertex in the MBAG, thus reducing the size of the MBAG. Experiments show that our approach achieves a substantial reduction in the number of vertices in the MBAG —by two orders of magnitude (see Figure 5 (c) of Exp-3)—thereby enabling more efficient traversal and computation of BCPCs.

While the partial-BCPC solution operates on a reduced MBAG, it still requires a full traversal of this structure, which can remain prohibitively large. To overcome this limitation, we propose a novel approach based on (α, β)-biclique enumeration. The basic idea is to enumerate all (α, β)-bicliques in the bipartite graph, and for each enumerated (α, β)-biclique B, connect all maximal bicliques containing B using union-find set. The reason this idea works is that two maximal bicliques are adjacent only if they share an (α, β)-biclique. Since the number of (α, β)-bicliques increases exponentially with respect to α, β [30, 31], we leverage partial-BCPC once again to prune the (α, β)-biclique enumeration tree. During the enumeration process, for each intermediate small biclique generated, we can maintain the set of maximal bicliques containing it. If this set is empty or all the maximal bicliques in the set already belong to the same partial-BCPC, then the current enumeration branch can be safely pruned. Experiments show that using partial-BCPC can reduce the number of nodes in the (α, β)-biclique enumeration tree by more than five orders of magnitude (Exp-4). Consequently, it achieves a speedup of nearly three orders of magnitude compared to the state-of-the-art (SOTA) algorithm (Exp-1).

We summarize our contributions as follows.

Novel partial-BCPC **based solution.** We propose a novel partial-BCPC based solution, characterized by introducing a relaxed definition of BCPC, namely partial-BCPC. By leveraging partial-BCPC, we can significantly reduce the size of the MBAG used in existing methods, thereby directly accelerating the traversal process of the MBAG. We show that leveraging partial-BCPC can reduce the number of vertices in the MBAG by up to two orders of magnitude (i.e., eliminating up to 99% of the vertices in MBAG, see Exp-3).

Novel (α, β)-biclique based solution. Unlike traditional approaches that rely on traversing the MBAG, we propose a novel framework that directly enumerates (α, β)-bicliques and connects the maximal bicliques that share them utilizing union-find sets. To reduce the cost of enumerating (α, β)-bicliques, we again leverage partial-BCPC, which reduces the number of nodes in the enumeration tree by up to five orders of magnitude (Exp-4).

Extensive experiments. We conduct experiments on 10 real-world datasets, and the experimental results demonstrate the effectiveness and efficiency of our proposed algorithms. Specifically, we test the performance of our algorithms under a wide range of parameter settings. For example, on dataset Youtube, when $\alpha = 2, \beta = 2$, the SOTA method (MBAG) takes 13,376 seconds, while the partial-BCPC based solution (PBCPC+) takes 197 seconds, achieving an improvement of nearly two orders of magnitude. In contrast, the (α, β)-biclique based solution (BicliqueP) takes 34 seconds, with an improvement of nearly three orders of magnitude. Additionally, we conduct a case study of BCPC in community detection. The results show that compared with other traditional community models (e.g., biclique, bitruss, bicore), BCPC can find more practically meaningful communities, which demonstrates the effectiveness of BCPC.

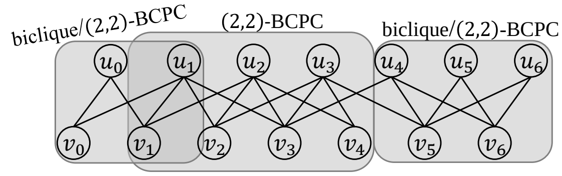


Figure 1: A bipartite graph G **and** (α, β)-BCPC **(**$\alpha = 2, \beta = 2$**)**

Reproducibility. The source code of this paper is released on https://anonymous.4open.science/r/bcpc for reproducibility purposes.

2 Preliminaries

Let $G = (U, V, E)$ be an undirected and unweighted bipartite graph, where U, V are two disjoint sets of vertices and E is a set of edges in form of $(u, v), u \in U, v \in V$. For a vertex $u \in U$, $Nei_G(u)$ is the set of neighbors of u, i.e., $Nei_G(u) = \{v | (u, v) \in E\}$. A bipartite graph $G' = (U', V', E')$ is a subgraph of G if $U' \subseteq U, V' \subseteq V, E' \subseteq E$.

Definition 1 (Biclique). *Given a bipartite graph $G = (U, V, E)$, a biclique is a pair of vertex sets $B = (X, Y)$ where $X \subseteq U, Y \subseteq V$ and $\forall u \in X, v \in Y, (u, v) \in E$.*

For simplicity, we let $B.X = X, B.Y = Y$ if $B = (X, Y)$. For two bicliques B, B', $B \subseteq B'$ indicates that $B.X \subseteq B'.X, B.Y \subseteq B'.Y$. A maximal biclique B is a biclique that there is no other biclique B' satisfying $B \subseteq B'$.

Definition 2 ((α, β)-Biclique Adjacency). *Given two integer α, β and two bicliques B_1, B_2, B_1 and B_2 are (α, β)-adjacent if $|B_1.X \cap B_2.X| \geq \alpha, |B_1.Y \cap B_2.Y| \geq \beta$.*

Definition 3 ((α, β)-Biclique Connectivity). *Given two integer α, β and two bicliques B_1, B_n, B_1 and B_n are (α, β)-connected if there is a sequence of bicliques $B_1, B_2, ..., B_n$ that $\forall i = 1, ..., n-1$, B_i and B_{i+1} are (α, β)-adjacent.*

Definition 4 ((α, β)-Biclique Percolation Community ((α, β)-BCPC)). *Given a bipartite graph G and two integer α, β, an (α, β)-biclique percolation community is a maximal set of maximal bicliques in G that any two maximal bicliques in the set are (α, β)-connected.*

Example 1. *Let $\alpha = 2, \beta = 2$, Figure 1 illustrates a bipartite graph G and some of the $(2,2)$-BCPCs. These BCPCs locates on the left, in the middle, and on the right, respectively. The $(2,2)$-BCPC in the middle is composed of three maximal bicliques: $B_1 = (\{u_1, u_2\}, \{v_1, v_2, v_3\})$, $B_2 = (\{u_1, u_2, u_3\}, \{v_2, v_3\})$, and $B_3 = (\{u_2, u_3\}, \{v_2, v_3, v_4\})$. Among these bicliques, B_1 and B_2 are adjacent because their intersection, $(\{u_1, u_2\}, \{v_2, v_3\})$, satisfies the condition in Definition 2. For the same reason, B_2 and B_3 are also adjacent. Therefore, B_1, B_2, and B_3 can together form a valid $(2,2)$-BCPC. On the left and right sides of Figure 1 are two maximal bicliques. These maximal bicliques are not adjacent (Definition 2) to any other maximal bicliques, and thus can be regarded as two special BCPCs.*

Algorithm 1: The basic maximal biclique enumeration framework in [9]

```
   Input: Bipartite graph G = (U, V, E)
   Output: All maximal bicliques in G
1  Enum (∅, ∅, U, V, ∅, ∅);
2  Procedure Enum (R_U, R_V, C_U, C_V, X_U, X_V)
3  if C_U ∪ C_V = ∅ then
4      if X_U ∪ X_V = ∅ then Output (R_U, R_V) as a maximal biclique;
5      return;
6  foreach u ∈ C_U do
7      Branch (R_U, R_V, u, C_U − {u}, C_V, X_U, X_V);
8      C_U ← C_U − {u}; X_U ← X_U ∪ {u};
9  foreach v ∈ C_V do
10     Branch (R_V, R_U, v, C_V − {v}, C_U, X_V, X_U);
11     C_V ← C_V − {v}; X_V ← X_V ∪ {v};
12 Procedure Branch (R_U, R_V, u, C_U, C_V, X_U, X_V)
13 C'_V ← C_V ∩ Nei_G(u); X'_V ← X_V ∩ Nei_G(u);
14 Enum (R_U ∪ {u}, R_V, C_U, C'_V, X_U, X'_V);
```

Problem statement. Given a bipartite graph G and two integer α, β, the goal of our work is to detect all (α, β)-BCPCs (BCPC detection).

2.1 Maximal Biclique Enumeration

The existing state-of-the-art maximal biclique enumeration methods [1, 5, 9] are all based on the set-enumeration tree structure, where vertices are iteratively traversed to build nodes of the enumeration tree, with each node representing a distinct biclique. In this paper, we adopts the algorithmic framework in [9] to perform maximal biclique enumeration and to design the partial-BCPC based method in Section 4. This is because the method in [9] achieves state-of-the-art performance in maximal biclique enumeration, and furthermore, the framework in [9] is adaptable to a wider range of cohesive subgraph models, making our partial-BCPC based method in Section 4 to also have the potential to be extended to other cohesive subgraph models in bipartite graphs.

The basic framework [9]. Algorithm 1 presents the basic framework for maximal biclique enumeration in [9]. This framework recursively explores candidate vertex sets to construct the set-enumeration tree, while leveraging exclusion sets to avoid non-maximal bicliques. Algorithm 1 begins by invoking the recursive procedure Enum in Line 1. The parameters of Enum contain the current biclique (R_U, R_V), candidate vertices (C_U, C_V) for expansion, and exclusion vertices (X_U, X_V) that have already been processed. Within Enum, the algorithm first checks the termination condition: $C_U \cup C_V = \emptyset$ (Line 3). If the termination condition is satisfied while $X_U \cup X_V = \emptyset$ (Line 4), it implies that (R_U, R_V) is a maximal biclique. Subsequently, the algorithm iterates over each vertex u in the candidate set C_U (Line 6), calling the Branch procedure to perform branch expansion (Line 7), and moves u from the candidate set C_U to the exclusion set X_U (Line 8). Symmetrically, the algorithm performs the same operation for each vertex v in C_V (Lines 9-11). In the Branch procedure, the algorithm computes the new candidate set $C'_V = C_V \cap Nei_G(u)$ and the new exclusion set $X'_V = X_V \cap Nei_G(u)$, adds u to R_U, and recursively calls the Enum procedure to further explore maximal bicliques within this branch.

Pivoting technique [9]. A notable issue with the aforementioned framework is that it generates a significant amount of unnecessary

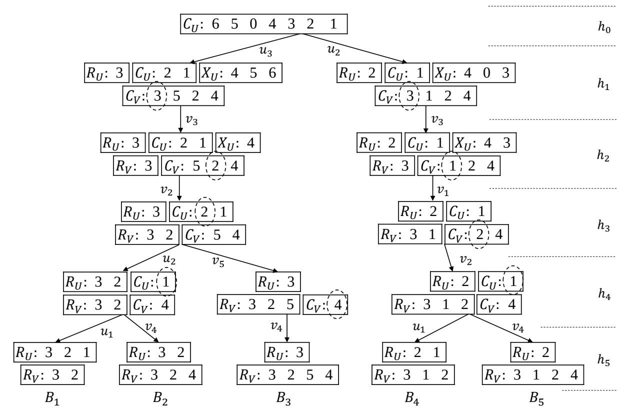


Figure 2: The partial enumeration tree of the framework in [9] on bipartite graph of Figure 1. Pivot vertices are enclosed in dashed circles. $nd(B_i, h_j)$ represents the tree node at level h_j on the B_i branch of the enumeration tree

computations that produce non-maximal bicliques. We first make the following observation: for any vertex $u \in U$ in a bipartite graph $G = (U, V)$, and for any maximal biclique $B = (X, Y)$ in the bipartite graph, exactly one of the following two conditions must hold:

(1) B contains u (i.e., $u \in X$).
(2) B contains a vertex v, where v is a non-neighbor of u in V (i.e., $\exists v \in V - Nei_G(u), v \in Y$).

Based on the above observation, we can conclude that by selecting a pivot vertex u from X_U, C_U, X_V, or C_V, and enumerating only u (if $u \notin X_U, X_V$) and vertices in C_V (or C_U, if $u \in X_V, C_V$) that are not neighbors of u in Lines 6-11 of Algorithm 1, we can ensure the completeness of all maximal bicliques while reducing unnecessary enumeration branches that produce non-maximal bicliques. An effective method for selecting a pivot vertex is to choose the vertex in $X_U \cup C_U$ (or $X_V \cup C_V$) that has the fewest non-neighbor vertices in C_V (or C_U) as the pivot vertex.

Vertex ordering [5]. The efficiency of maximal biclique enumeration algorithms is often influenced by the order in which vertices are processed. [9] applied the 2-hop degree ordering introduced in [5] to their framework. This vertex ordering is only applied at the first level of the recursive search, while the pivoting pruning technique mentioned earlier is utilized in subsequent levels of the recursive search.

Example 2. *Figure 2 illustrates a partial enumeration tree for maximal biclique enumeration [9] on bipartite graph $G = (U, V, E)$ in Figure 1. The root node, located at level h_0, initializes its candidate set C_U with vertex set U of the bipartite graph, ordered by 2-hop degree ordering. The root node only contains C_U and no C_V. The enumeration process in level h_0 follows the ordered sequence, processing u_6, u_5, u_0, and u_4 first, followed by u_3 and u_2, while only branches processing u_3 and u_2 are shown. For tree node $nd(B_1, h_1)$, u_3 is added to R_U, and C_V is updated to include $Nei_G(u_3)$ in V. The exclusion set X_U in $nd(B_1, h_1)$ contains u_4, u_5, u_6, as these vertices were enumerated in previous branches and added to X_U at level h_0 (initially, X_U at level h_0 is empty and is not shown).*

At $nd(B_1, h_1)$, the framework selects v_3 as the pivot vertex. Consequently, vertices in C_V other than v_3 and vertices in $Nei_G(v_3) \cap C_U$

Algorithm 2: The (α,β)-biclique enumeration framework in [30]

Input: Bipartite graph $G=(U,V,E)$; Two integer α,β
Output: All (α,β)-bicliques in G

1 Construct 2-hop graph $\vec{H}=(U,\vec{E_h})$ on vertex set U [30]; /* $\vec{H}=(U,\vec{E_h})$: a direct graph, $(u,v)\in\vec{E_h}$ indicates that $u<v$, u,v have common neighbors in V */
2 BicliqueListing $(\vec{H},\emptyset,V)$;
3 **Procedure** BicliqueListing $(\vec{H},R_U,C_V)$
4 **if** $|R_U|=\alpha$ **then**
5   **foreach** $R_V\subseteq C_V$, $|R_V|=\beta$ **do** Output (R_U,R_V) as an (α,β)-biclique ;
6   **return**;
7 **foreach** $u\in\vec{H}$ **do**
8     $C'_V\leftarrow C_V\cap Nei_G(u)$;
9   **if** $|C'_V|<\beta$ *or* $Nei_{\vec{H}}(u)<\alpha-|R_U|-1$ **then continue**;
10   Let $\vec{H}'$ be induced subgraph of $\vec{H}$ on $Nei_{\vec{H}}(u)$;
11   BicliqueListing $(\vec{H}',R_U\cup\{u\},C'_V)$;

(u_2,u_1) are skipped. Thus, only the branch enumerating v_3 is expanded at level h_2. At level h_2, v_3 is added to R_V, and removed from C_V. The sets C_U and X_U at level h_2 are updated similarly to further refine the search space and exclusion set. When both candidate sets C_U and C_V are empty (i.e., at the leaf nodes of the enumeration tree), and if the exclusion sets X_U and X_V are also empty, then (R_U,R_V) is a maximal biclique.

2.2 (α,β)-Biclique Enumeration

The existing state-of-the-art (α,β)-biclique enumeration method is proposed in [30]. Given a bipartite graph $G=(U,V,E)$, the core idea is to enumerate combinations of vertices in the U set while maintaining their common neighbors in the V set. It also constructs an ordinary directed graph, called the *2-hop graph*, on the vertices of the U set to accelerate the enumeration process.

Algorithm 2 presents the basic framework. It first constructs the *2-hop graph* $\vec{H}$ with U as its vertex set (Line 1), where an edge (u,v) exists if $u<v$ and u,v share at least one common neighbor in set V of the bipartite graph. In Line 4, if R_U already contains α vertices, it is sufficient to enumerate all combinations of β vertices from C_V to obtain the (α,β)-bicliques. In Line 7, the algorithm enumerates vertices u in $\vec{H}$, then in Line 8 updates the candidate set C'_V in V, and proceeds with the next-level enumeration in Line 11.

2.3 Union-Find Set [8]

The Union-Find Set, also referred to as the Disjoint-Set data structure, serves as an efficient method for managing a collection of disjoint sets. Each set is represented as a tree, where the nodes represent the elements of the set. Within this tree, every node points to its parent, with the root of the tree acting as the representative of the set. This tree structure allows for fast **union** and **find** operations, making the Union-Find Set well-suited for tasks such as graph connectivity problems. **Find(i)** is used to get the representative of the unique set containing i, **union(i, j)** is used to merge the two sets containing i and j into their union. Specifically, within the corresponding tree structure, the representative node of one set is assigned as the parent of the representative node of another set.

In all practical scenarios, the **union** and **find** operations of the union-find set achieve constant amortized time complexity. The space complexity of this structure is $O(n)$, with n denoting the total number of elements it manages [8].

Algorithm 3: Existing MBAG-based solution

Input: Bipartite graph $G=(U,V,E)$; Two integer α,β
Output: All BCPC in G

1 $\mathcal{B}\leftarrow$ set of all maximal bicliques;
2 $\mathcal{B}\leftarrow\{B|B\in\mathcal{B},|B.X|\geq\alpha,|B.Y|\geq\beta\}$;
3 $Nei(B)\leftarrow$ set of maximal bicliques that share vertices with B;
4 $UF\leftarrow$ an initial union-find set, i.e., $UF.find(B)=B$ for any input B;
5 **for** $B\in\mathcal{B}$ **do**
6   **for** $B'\in Nei(B)$ **do**
7   **if** $|B.X\cap B'.X|\geq\alpha$ *and* $|B.Y\cap B'.Y|\geq\beta$ **then** $UF.union(B,B')$;
8 **return** UF;

3 Existing Solution

The existing BCPC detection method was proposed and implemented in [6, 14, 27]. To the best of our knowledge, no other work has primarily focused on the algorithmic implementation and improvement of BCPC detection. The basic idea is based on *maximal biclique adjacency graph* (MBAG). This method first enumerates all maximal bicliques and then uses them as vertices to construct a maximal biclique adjacency graph. In this graph, there is an edge between two vertices if and only if the overlapping part of the corresponding two maximal bicliques contains an (α,β)-biclique (satisfying Definition 2). The connected components in the adjacency graph correspond one-to-one with the (α,β)-BCPC. In this method, the (α,β)-biclique connectivity of maximal bicliques in (α,β)-BCPC are maintained by the classic union-find set [8].

Algorithm 3 is the existing MBAG-based algorithm identifying all (α,β)-BCPCs in a bipartite graph $G=(U,V,E)$. It starts by enumerating all maximal bicliques in G and filters them based on size constraints (Lines 1-2). The reason is that for a maximal biclique B, if $|B.X|<\alpha$ or $|B.Y|<\beta$, then B can not form a BCPC with other maximal bicliques. A union-find set UF is initialized in Line 4. The algorithm then iterates over each maximal biclique B and its neighbors $B'\in Nei(B)$, checking if $|B.X\cap B'.X|\geq\alpha$, $|B.Y\cap B'.Y|\geq\beta$ (Lines 5-7). If the conditions are satisfied, B and B' are merged into the same BCPC by $UF.union(*)$ (Line 7). After processing all maximal bicliques, the union-find set UF represents the final BCPCs in the bipartite graph.

The major drawback of this approach is that the size of the MBAG can be extremely large (up to 10^{11}, see Table 1). Consequently, traversing such a massive MBAG to derive all BCPCs is clearly impractical.

4 Novel Partial-BCPC Based Solution

As we discussed in Section 3, current approach relying on MBAG (Algorithm 3) faces significant challenges due to the vast scale of MBAG. This immense size leads to inefficient traversal operations. Consequently, it becomes evident that optimizing the performance of MBAG-based method hinges on effectively reducing the size of the MBAG.

Key idea. We reduce the size of the MBAG by merging vertices within it. The key is to leverage the prefixes of the maximal biclique enumeration tree during the enumeration process. During the enumeration, if a prefix of the enumeration tree contains at least α

vertices in U and β vertices in V, we can determine that all maximal bicliques under this branch belong to the same BCPC. Connections between these maximal bicliques are unnecessary to be traversed. These bicliques collectively form an incomplete BCPC, referred to as a partial-BCPC.

EXAMPLE 3. *Let $\alpha = \beta = 1$. Consider Figure 2, where the subtree rooted at node $nd(B_1, h_2)$ contains three maximal biclique branches: B_1, B_2, and B_3. All three maximal bicliques include both $nd(B_1, h_2).R_U$ and $nd(B_1, h_2).R_V$. Therefore, we can regard B_1, B_2, and B_3 as forming an incomplete* BCPC, *i.e., a partial*-BCPC.

Next, we will formally define partial-BCPC, the maximal biclique enumeration tree and the special prefixes used to compute partial-BCPCs.

DEFINITION 5 (PARTIAL-BCPC). *Given a bipartite graph G and two integer α, β, a partial*-BCPC *$\mathcal{P}$ is a set of maximal bicliques satisfying $\mathcal{P} \subseteq \mathcal{B}$, where $\mathcal{B}$ is an (α, β)*-BCPC *in G.*

The distinction from Definition 4 lies in the fact that Definition 5 does not require the set of maximal bicliques in the partial-BCPC to be maximal. This reflects the characteristic of partial-BCPC as an incomplete BCPC. In practice, partial-BCPCs are also maintained by union-find set.

DEFINITION 6 (MAXIMAL BICLIQUE ENUMERATION TREE (MBE TREE)). *Given a bipartite graph $G = (U, V, E)$. The* MBE *tree $T = (N, \vec{E})$ represents the search space of the maximal biclique enumeration algorithm on G. Here, N denotes the set of nodes in the tree, where each node $nd \in N$ contains a six-tuple $(R_U, R_V, C_U, C_V, X_U, X_V)$, as defined in the basic framework of* MBE *in Section 2.1. The set $\vec{E}$ consists of directed edges of the form (nd_i, nd_j, u), where $u \in U$ or $u \in V$. An edge $(nd_i, nd_j, u) \in \vec{E}$ indicates that node nd_j is a child of node nd_i, generated by selecting vertex u from $nd_i.C_U$ or $nd_i.C_V$. A node $nd \in N$ is a leaf node if $nd.C_U \cup nd.C_V = \emptyset$. For a leaf node $nd \in N$, if $nd.X_U \cup nd.X_V = \emptyset$, then $(nd.R_U, nd.R_V)$ is a maximal biclique.*

For simplicity, we use $nd.RCX_U$ and $nd.RCX_V$ to denote $nd.R_U \cup nd.C_U \cup nd.X_U$ and $nd.R_V \cup nd.C_V \cup nd.X_V$, respectively.

DEFINITION 7 ((α, β)-PREFIX AND (α, β)-NODE). *Given a bipartite graph $G = (U, V, E)$ and its* MBE *tree $T = (N, \vec{E})$, let $nd_1 \in N$ be the root node of T, $PF(nd_l) = (nd_1, nd_2, ..., nd_l)$ is the path from nd_1 to nd_l, i.e., $\forall i = 1, 2, ..., l-1$, $(nd_i, nd_{i+1}, *) \in \vec{E}$. $PF(nd_l)$ is an (α, β)-prefix if and only if $|nd_l.R_U| = \alpha, |nd_{l-1}.R_U| < \alpha, |nd_l.R_V| \geq \beta$ or $|nd_l.R_V| = \beta, |nd_{l-1}.R_V| < \beta, |nd_l.R_U| \geq \alpha$. In this case, nd_l is an (α, β)-node.*

In short, an (α, β)-node nd is the first node that satisfies $|nd.R_U| \geq \alpha$ and $|nd.R_V| \geq \beta$ in all branches where nd resides. For all its ancestor nodes nd_i, it is not possible to simultaneously satisfy $|nd_i.R_U| \geq \alpha$ and $|nd_i.R_V| \geq \beta$.

THEOREM 1. *For any two (α, β)-nodes nd_1, nd_2, nd_1 can not be an ancestor or a descendant of nd_2.*

PROOF. Based on Definition 7, nd_1 is the first node that satisfies $|nd_1.R_U| \geq \alpha$ and $|nd_1.R_V| \geq \beta$ in all branches where nd_1 resides. The same applies to nd_2. Thus, nd_1, nd_2 are in two different branches, and nd_1 can not be an ancestor or a descendant of nd_2. □

Algorithm 4: Partial-BCPC based solution

Input: Bipartite graph $G = (U, V, E)$; Two integer α, β
Output: All BCPC in G

1 Let $T = (N, \vec{E})$ be the MBE tree of G, $\mathcal{B}$ be the set of maximal bicliques in G;
2 $Nei(B) \leftarrow$ set of maximal bicliques that share vertices with B;
3 Let UF be an initial union-find set;
4 Let $\mathcal{P} \leftarrow \emptyset$;
`/* compute partial-BCPCs */`
5 **for** (α, β)*-node* nd *in* T **do**
6   ProcessNode (nd);
7   **for** $B \in \mathcal{P}$ **do** $UF.union(B, \mathcal{P}[0])$;
8     $\mathcal{P} \leftarrow \emptyset$;
`/* traverse MBAG with partial-BCPCs */`
9 **for** $B \in \mathcal{B}$ **do**
10   **for** $B' \in Nei(B)$ *and* B, B' *are in different partial*-BCPC*s* **do**
11   **if** $|B.X \cap B'.X| \geq \alpha, |B.Y \cap B'.Y| \geq \beta$ **then** $UF.union(B, B')$;
12 **return** UF;
`/* recursively traverse the MBE tree to gather all maximal bicliques under` *node* `*/`
13 **Procedure** ProcessNode $(node)$
14 **for** $(node, node', u) \in \vec{E}$ **do**
15   **if** $node'.C_U \cup node'.C_V = \emptyset$ **then**
16   **if** $node'.X_U \cup node'.X_V = \emptyset$ **then**
17             $\mathcal{P} \leftarrow \mathcal{P} \cup \{(node'.R_U, node'.R_V)\}$; `/*` $(node'.R_U, node'.R_V)$ `is a maximal biclique */`
18   **else** ProcessNode $(node')$;

4.1 The Basic Framework

This section presents the basic framework for our partial-BCPC based solution. The framework consists of two fundamental steps: (1) Computing partial-BCPCs using the MBE tree; (2) Traversing the reduced MBAG based on the partial-BCPCs to derive the final BCPCs. When computing partial-BCPCs, the framework connects the maximal bicliques within the subtree rooted at specific nodes in the MBE tree using a union-find set. Specifically, we focus on (α, β)-nodes in MBE tree. Ancestors of an (α, β)-node are ignored as they fail to satisfy $|R_U| \geq \alpha$ and $|R_V| \geq \beta$. Descendants of an (α, β)-node, while satisfying $|R_U| \geq \alpha$ and $|R_V| \geq \beta$, are redundant since any maximal biclique branches sharing the prefix corresponding to the descendants must also share the (α, β)-prefix corresponding to the (α, β)-node. Thus, focusing solely on (α, β)-nodes enables efficient computation of partial-BCPCs.

Implementation. Algorithm 4 presents the basic framework. It begins by maximal biclique enumeration and extracting MBE tree $T = (N, \vec{E})$ (Line 1). It then initializes the neighbor list of maximal bicliques and a union-find set UF to manage the partial-BCPCs and BCPCs (Lines 2-3). In Lines 5-8, the algorithm iterates over each (α, β)-node nd in T (Line 5), invoking the ProcessNode function to gather all maximal bicliques under subtree rooted at nd into $\mathcal{P}$ (Line 6). Subsequently, it merges all maximal bicliques in $\mathcal{P}$ into the same partial-BCPC (Line 7). In Lines 9-11, the algorithm traverses a smaller MBAG based on the partial-BCPCs. Specifically, for each maximal biclique B, the algorithm only traverses maximal bicliques B' that reside in different partial-BCPCs from B (Line 10). The algorithm finally returns UF, which represents all detected BCPCs.

The storage of the MBE **tree.** In practice, we do not store all branches of the MBE tree. We first introduce the two types of nodes in the MBE tree.

Definition 8 (Real and Virtual Node). *Given a graph G and its* MBE *tree* $T = (N, \vec{E})$, *for any node* $nd \in N$, *if nd is a leaf node and satisfies* $nd.X_U = nd.X_V = \emptyset$, *then nd is a real node—in this case,* $(nd.R_U, nd.R_V)$ *is a maximal biclique (see Definition 6). Otherwise, if nd has at least one descendant that is a real node, it is also a real node. If neither of these conditions is satisfied, then nd is a virtual node.*

The ProcessNode procedure in Algorithm 4 only needs to identify the branches that contain maximal bicliques (Lines 15-17), i.e., branches composed of real nodes. Thus, in practice, we only store real nodes in the MBE tree.

Analysis. We first analyze the correctness of Algorithm 4, then examine the relationship between the number of partial-BCPCs obtained by Algorithm 4 and the numbers of maximal bicliques as well as BCPCs, and finally analyze the complexity of Algorithm 4.

Theorem 2. *Algorithm 4 correctly computes all* BCPC*s.*

Proof. We first prove that the partial-BCPCs are all correct. The procedure ProcessNode collects all maximal bicliques within the subtree rooted at the given *node* into $\mathcal{P}$. Since the parameter is always an (α, β)-node, the maximal bicliques collected in each execution of the procedure must share an (α, β)-biclique. Therefore, these bicliques correctly form a partial-BCPC. Then we prove that the final BCPCs are correct. In Lines 9-11, any two maximal bicliques that belong to different partial-BCPCs but satisfy the adjacency condition are connected via the union-find set UF. As a result, each disjoint set maintained by UF corresponds to a valid BCPC. □

Theorem 3. *Given a bipartite graph G, two integer* α, β *and* MBE *tree* $T = (N, \vec{E})$, *let* $n_{biclique}$ *be the number of maximal bicliques,* $ND \leftarrow \{nd | nd \in N, nd \text{ is an } (\alpha, \beta)\text{-node}, nd \text{ is a real node}\}$, *pbcpc be the number of partial-*BCPC*s computed by Algorithm 4, bcpc be the number of* BCPC*s, the following inequality holds:* $n_{biclique} \geq |ND| = pbcpc \geq bcpc$.

Proof. Based on Theorem 1, any two (α, β)-nodes in ND can not be in the same branch, and nodes in ND are all real nodes, thus, each node in ND has at least one maximal biclique in its subtree. As a result, $n_{biclique} \geq |ND|$. Since Algorithm 4 connects all maximal bicliques in the subtree of each (α, β)-node in ND into a partial-BCPC, and the maximal bicliques from different subtrees of (α, β)-nodes are distinct, thus, each (α, β)-node in ND corresponds to one partial-BCPC ($|ND| = pbcpc$). Partial-BCPCs are incomplete BCPCs according to Definition 5, thus, $pbcpc \geq bcpc$. □

Theorem 4. *The time and space complexity of Algorithm 4 are* $O((1 - \frac{1}{pbcpc}) n_{biclique}^2 n)$ *and* $O(n_{biclique} n)$ *respectively, where* $n = |U| + |V|$, *pbcpc is the number of partial-*BCPC*s,* $n_{biclique}$ *is the number of maximal bicliques.*

Proof. Let $m = |E|$, the time complexity of maximal biclique enumeration is $O(2^{n/2} m)$ [9]. Compared to maximal biclique enumeration, Algorithm 4 introduces additional operations: traversing (α, β)-nodes and collecting maximal biclique branches from the subtrees rooted at these nodes. Since (α, β)-nodes do not reside in the same branch (Theorem 1), the time complexity of these additional operations does not exceed $O(2^{n/2} m)$. In Lines 9-11, in the worst case, Algorithm 4 only skips the inner mutual visits of maximal bicliques in each initial partial-BCPC. Let $x = n_{biclique}$, $y = pbcpc$ and $p_1, p_2, ..., p_y$ be the number of maximal bicliques in each partial-BCPC, the time complexity of Lines 9-11 is $O(x^2 n - (p_1^2 + p_2^2 + ... + p_y^2) n) = O(x^2 n - (\frac{x}{y})^2 y n)$ (based on Cauchy-Schwarz inequality), which can be written as $O((1 - \frac{1}{pbcpc}) n_{biclique}^2 n)$. As a result, the time complexity of Algorithm 4 is $O(2^{n/2} m + (1 - \frac{1}{pbcpc}) n_{biclique}^2 n)$. Since $O(n_{biclique}) = O(2^{n/2})$ [9, 24], the time complexity can be rewritten as $O((1 - \frac{1}{pbcpc}) n_{biclique}^2 n)$.

As for the space complexity, the main space consumption in Algorithm 4 comes from the MBE tree and maximal bicliques. Since only the real nodes in the MBE tree are stored, the number of branches in the MBE tree is equal to the number of maximal bicliques, thus, the space complexity of Algorithm 4 is $O(n_{biclique} n)$. □

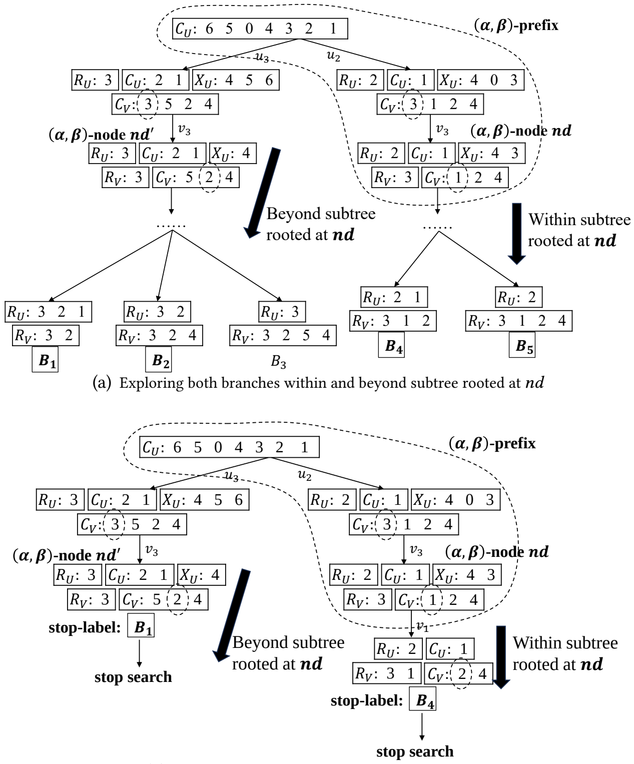
Discussion. The inequality $n_{biclique} \geq pbcpc$ in Theorem 3 forms the basis for reducing the MBAG using partial-BCPC. This is because maximal bicliques within the same partial-BCPC are treated as a single vertex in the MBAG, which leads to an MBAG with formally fewer vertices. This can also be seen from the time complexity in Theorem 4: if $pbcpc$ is replaced with $n_{biclique}$, the result is exactly the complexity of the existing method in Section 3 for traversing the complete MBAG ($O(n_{biclique}^2 n)$). Experiments (Figure 5 (c)) show that $pbcpc$ can be an order of magnitude smaller than $n_{biclique}$.

4.2 Optimizations

Beyond subtree rooted at (α, β)-node. The basic framework in the previous section directly merges the maximal bicliques in the subtree rooted at an (α, β)-node into a partial-BCPC. However, this approach is insufficient because, for an (α, β)-node nd, not all maximal bicliques sharing $nd.R_U$ and $nd.R_V$ are located within the subtree rooted at nd. For example, in Figure 2, if $\alpha = \beta = 1$, $nd(B_4, h_2)$ is an (α, β)-node, but B_1 and B_2 are not in the subtree rooted at $nd(B_4, h_2)$, even though both B_1 and B_2 share $nd(B_4, h_2).R_U$ and $nd(B_4, h_2).R_V$. This phenomenon occurs because some vertices in B_1 and B_2 (e.g., u_3) appear early in the candidate set C_U of the root node, causing B_1 and B_2 to be enumerated much earlier.

To merge all relevant maximal bicliques within and beyond the subtree of an (α, β)-node, it is essential to fully utilize all the information maintained in the (α, β)-node, i.e., R_U, R_V, C_U, C_V, X_U, and X_V, to systematically search on the MBE tree. Specifically, for each (α, β)-node nd, we use $nd.RCX_U$ and $nd.RCX_V$ to search, within the MBE tree, all branches of maximal bicliques that contain $nd.R_U$ and $nd.R_V$, and connect them into the same partial-BCPC.

Example 4. *Let* $\alpha = 1, \beta = 1$, *Figure 3 (a) illustrates that when processing* (α, β)*-node nd, both the branches within and beyond subtree rooted at nd are taken into account. It can be observed that the maximal biclique branches sharing* $(nd.R_U, nd.R_V)$ *include not only* B_4 *and* B_5 *under nd but also the maximal biclique branches* B_1 *and* B_2 *beyond the subtree rooted at nd.*



(a) Exploring both branches within and beyond subtree rooted at *nd*

(b) Using stop-labels to terminate exploration early

Figure 3: Examples of optimization

Algorithm 5: Partial-BCPC based solution with optimizations

```
   Input: Bipartite graph G = (U, V, E); Two integer α, β
   Output: All BCPC in G
1  Lines 1-3 of Algorithm 4;
2  Let 𝒫 ← ∅, root ← root node of T;
3  Let ST(nd), nd ∈ N be the stop-label of nd;
   /* compute partial-BCPCs                                  */
4  Postorder (root);
   /* traverse MBAG with partial-BCPCs                       */
5  Lines 9-11 of Algorithm 4;
6  return UF;

7  Procedure Postorder (node)
8  foreach child of node, node', in creation order of node' do Postorder (node') ;
9  if node is an (α, β)-node then
10     ProcessNode+ (root, node.RCX_U ∪ node.RCX_V, node.R_U ∪ node.R_V);
11 else if node is a real node and a descendant of an (α, β)-node then
12     ProcessNode+ (node);
13 if 𝒫 is not empty then
14     for B ∈ 𝒫 do UF.union(B, 𝒫[0]);
15     ST(node) ← 𝒫[0], 𝒫 ← ∅;

16 Procedure ProcessNode+ (node, UV, RUV)
17 foreach child of node, node', in creation order of node' do
18     Let (node, node', u) ∈ E⃗;
19     if node' is a real node then
20         if u ∈ UV then
21             if ST(node') = None then ProcessNode+ (node', UV, RUV) ;
22             else 𝒫 ← 𝒫 ∪ {ST(node')} ;
23     if u ∈ RUV then break;

24 Procedure ProcessNode+ (node)
25 if node is a leaf node then 𝒫 ← 𝒫 ∪ {(node.R_U, node.R_V)};
26 else
27     foreach node', node' is a child of node and a real node do
28         𝒫 ← 𝒫 ∪ {ST(node')}
```

Stop-Label. The above method may result in fully traversing all the relevant maximal biclique branches when processing each (α, β)-node. To reduce the overhead of traversing all maximal biclique branches that share $nd.R_U$ and $nd.R_V$ (nd is an (α, β)-node) without affecting correctness, we assign a stop-label to certain real nodes in the MBE tree. When such a real node is visited, the stop-label allows immediate retrieval of all maximal biclique information within its subtree, thus eliminating the need to further traverse its descendants. Given α, β, the following presents the details of the stop-label:

(1) Stop-labels are assigned only to real nodes in the MBE tree.
(2) For a descendant node nd of an (α, β)-node, the prerequisite for setting stop-label to nd is to connect all maximal bicliques within the subtree rooted at nd into the same partial-BCPC. The stop-label of nd is set to one of the maximal bicliques in that partial-BCPC.
(3) For an (α, β)-node nd, the prerequisite for setting stop-label to nd is to connect all maximal bicliques sharing $nd.R_U$ and $nd.R_V$ into the same partial-BCPC. The stop-label of nd is set to one of the maximal bicliques in that partial-BCPC

We now describe how stop-labels contribute to pruning in identifying branches.

(1) For a descendant node nd of an (α, β)-node, before setting a stop-label to nd, it is sufficient to merge the stop-labels of all its child nodes (also real nodes) into the same partial-BCPC using union-find set. This eliminates the need to traverse all maximal biclique branches within the subtree rooted at nd.
(2) For (α, β)-node nd, in identifying all maximal biclique branches that share $nd.R_U$ and $nd.R_V$, the search terminates as soon as it encounters a node with a stop-label and directly adds the stop-label to the set $\mathcal{P}$.

Example 5. *Let $\alpha = 1, \beta = 1$, when processing nd in Figure 3 (b):*

*During the search of subtree rooted at nd, the process terminates upon encountering the stop-label of child node of nd. This stop-label is B_4, with B_5 belonging to the same partial-*BCPC *as B_4.*

During the search beyond the subtree rooted at nd, the search process terminates upon detecting the stop-label of nd'.

*Finally, the two stop-labels (B_1 and B_4) are connected; concurrently, the partial-*BCPC*s they belong to are merged into a larger partial-*BCPC.

Implementation. Based on the above strategies, we revise the original framework and present Algorithm 5. Algorithm 5 performs an initialization step in Line 1 that is similar to that of Algorithm 4. In Line 3, $ST(nd)$ denotes the stop-label of nd. The core component of Algorithm 5 is the Postorder function (Lines 4 and 7). This function performs a post-order traversal of the MBE tree, a process that can be executed during the MBE progress. The Postorder function is responsible for handling both the (α, β)-node itself—by invoking ProcessNode+ $(node, UV, RUV)$ in Line 10—and the real nodes among its descendant nodes—by invoking ProcessNode+ $(node)$

in Line 12. Both processing routines may add certain maximal bicliques to $\mathcal{P}$. Finally, in Lines 13–15, Algorithm 5 connects all the maximal bicliques in $\mathcal{P}$ into the same partial-BCPC, and it set the first biclique in $\mathcal{P}$ as the stop-label of *node* (Line 15). Similar to Algorithm 4, Algorithm 5 traverses the MBAG based on partial-BCPCs to obtain the final BCPC (Line 5).

When processing an (α, β)-node (Line 10), the ProcessNode+ function starts traversing the MBE tree from its root node. It mainly explores real nodes (Line 19), since any branch containing a maximal biclique must be composed of real nodes. Moreover, any vertex u along the traversal path must belong to the set UV (Line 20). The recursive calls end immediately upon encountering a node that has a stop-label assigned (Lines 22). After processing a child node $node'$, if the vertex u along the traversal path belongs to the set RUV, there is no need to explore the remaining child nodes and their corresponding branches (Line 23). This is because, according to the enumeration principle of the MBE algorithm, any maximal biclique that includes RUV can not be found in those subsequent branches. Since the goal of this function is to identify maximal bicliques that contain RUV, further traversal becomes unnecessary in this case.

When processing the descendant nodes of an (α, β)-node (Line 12), if the parameter *node* is already a leaf node, the maximal biclique it represents can be directly added to $\mathcal{P}$. Otherwise, the stop-labels of all its child nodes are added to $\mathcal{P}$.

Analysis. We now proceed to analyze the correctness of Algorithm 5. The key lies in demonstrating that the partial-BCPCs computed in Postorder (Line 3) are correct.

Theorem 5. *For each (α, β)-node processed in Algorithm 5 (Line 9),* ProcessNode+ *connects all maximal bicliques in the bipartite graph that share $(node.R_U, node.R_V)$ into the same partial*-BCPC.

Proof. Consider a sequence of (α, β)-nodes $(nd_1, nd_2, \dots, nd_n)$, which follow the processing order in Postorder (i.e., post-order traversal). We first prove that during the processing of nd_1, ProcessNode+ connects all maximal bicliques that share $(nd_1.R_U, nd_1.R_V)$ into the same partial-BCPC. Since nd_1 is the first (α, β)-node, it must also be in the first enumerated maximal biclique branch (if nd_1 appears in an earlier branch, then that earlier or an even earlier branch is likely the first one containing a maximal biclique; if nd_1 appears in a later branch, then it can not be the first (α, β)-node). Therefore, all maximal biclique branches that contain $(nd_1.R_U, nd_1.R_V)$ must lie in the subtree rooted at nd_1, and can not appear in any subtree processed before nd_1.

Using mathematical induction, assume that ProcessNode+ connects all maximal bicliques that share $(nd_i.R_U, nd_i.R_V)$ when processing $nd_1, nd_2, ..., nd_i$. Next, we prove the case for $i+1$. For nd_{i+1}, let $R_U = nd_{i+1}.R_U$ and $R_V = nd_{i+1}.R_V$. According to the MBE process [9], the branches containing maximal bicliques with R_U and R_V are either generated before nd_{i+1}, or located in the subtree rooted at nd_{i+1}. For the maximal biclique branches generated before nd_{i+1}, their corresponding (α, β)-node nd_j must satisfy $j < i+1$, so we only need to collect $ST(nd_j)$, as it represents the partial-BCPCs of those maximal bicliques. For the maximal bicliques in the subtree of nd_{i+1}, we only need to collect the stop-labels of the child nodes of nd_{i+1}, because the maximal bicliques under each child node have already been connected into partial-BCPCs (since the traversal is post-order). Therefore, the case for nd_{i+1} is proved. □

Theorem 6. *Algorithm 5 correctly computes all* BCPC*s.*

Proof. The proof of this theorem is similar to that of Theorem 2. Since Theorem 5 proves that the partial-BCPCs in Algorithm 5 are obtained by connecting maximal bicliques that share the R_U and R_V of (α, β)-nodes, these partial-BCPCs are correct. Moreover, the process of computing the final BCPCs in Algorithm 5 (Line 5) is the same as in Algorithm 4, thus, Algorithm 5 can correctly compute the BCPCs. □

It should be noted that the partial-BCPCs computed by Algorithm 5 are not the same as those computed by Algorithm 4. We illustrate this with the following theorem.

Theorem 7. *Given a bipartite graph G, two integer α, β, let $pbcpc$ be the number of partial*-BCPC*s computed by Algorithm 4, $pbcpc+$ be the number of partial*-BCPC*s computed by Algorithm 5, $bcpc$ be the number of* BCPC*s, the following inequality holds: $pbcpc \geq pbcpc+ \geq bcpc$.*

Proof. We only need to prove $pbcpc \geq pbcpc+$. When processing an (α, β)-node, Algorithm 5 not only connects maximal bicliques in the subtree rooted at that node, but also further connects maximal bicliques that were enumerated in earlier branches and share R_U, R_V of the (α, β)-node. Moreover, the ProcessNode+ procedure in Algorithm 5 also processes (α, β)-nodes that belong to virtual nodes. Therefore, $pbcpc \geq pbcpc+$. □

Theorem 8. *The time and space complexity of Algorithm 5 are $O((1-\frac{1}{pbcpc+})n^2_{biclique}n)$ and $O(n_{biclique}n)$ respectively, where $n = |U| + |V|$, $pbcpc+$ is the number of partial*-BCPC*s, $n_{biclique}$ is the number of maximal bicliques.*

Proof. Let $m = |E|$, the time complexity of maximal biclique enumeration is $O(2^{n/2}m)$ [9]. Unlike Algorithm 4, Algorithm 5 handles not only (α, β)-nodes (Lines 9-10) but also their child nodes (Lines 11-12). For processing the child nodes of (α, β)-nodes, Algorithm 5 only needs to traverse one layer of their child nodes, resulting in an overall time complexity not exceeding $O(2^{n/2}m)$. Regarding the processing of (α, β)-nodes, let x be the total number of such nodes (including both real nodes and virtual nodes), d be the maximum depth of real (α, β)-nodes. When processing each of these nodes (Line 10), the procedure ProcessNode+ will access at most all real (α, β)-nodes. Since the number of child nodes of each node in the MBE tree does not exceed n, the total time complexity of the above process is $O(xn^d) = O(2^{n/2}mn^d)$. The time complexity analysis for the traversal of MBAG in Algorithm 5 is similar to that of Algorithm 4. As a result, the time complexity of Algorithm 5 is $O(2^{n/2}mn^d + (1-\frac{1}{pbcpc+})n^2_{biclique}n)$. Since $O(n_{biclique}) = O(2^{n/2})$ [9, 24], the time complexity can be rewritten as $O((1-\frac{1}{pbcpc+})n^2_{biclique}n)$.

Similar to Algorithm 4, the space complexity of Algorithm 5 is also $n_{biclique}n$. □

Algorithm 6: (α, β)-Biclique based solution

Input: Bipartite graph $G = (U, V, E)$; Two integer α, β
Output: All BCPC in G

1 Let UF be an initial union-find set;
2 $\mathcal{B} \leftarrow$ set of all maximal bicliques in G;
3 $\mathcal{B} \leftarrow \{B | B \in \mathcal{B}, |B.X| \geq \alpha, |B.Y| \geq \beta\}$;
4 Construct 2-hop graph $\vec{H} = (U, \vec{E_h})$ on vertex set U [30]; /* $\vec{H} = (U, \vec{E_h})$: a direct graph, $(u, v) \in \vec{E_h}$ indicates that $u < v$, u, v have common neighbors in V */
5 BicliqueListing $(\vec{H}, \emptyset, V)$;
6 **return** UF;

7 **Procedure** BicliqueListing $(\vec{H}, R_U, C_V)$
8 **if** $|R_U| = \alpha$ **then**
9   **foreach** $R_V \subseteq C_V, |R_V| = \beta$ **do** Connect maximal bicliques sharing (R_U, R_V) in $\mathcal{B}$ with $UF.union(*)$;
10   **return**;
11 **foreach** $u \in \vec{H}$ **do**
12     $C'_V \leftarrow C_V \cap Nei_G(u)$;
13   **if** $|C'_V| < \beta$ *or* $Nei_{\vec{H}}(u) < \alpha - |R_U| - 1$ **then continue**;
14   Let $\vec{H}'$ be induced subgraph of $\vec{H}$ on $Nei_{\vec{H}}(u)$;
15   BicliqueListing $(\vec{H}', R_U \cup \{u\}, C'_V)$;

Discussion. The inequality $pbcpc \geq pbcpc$+ in Theorem 7 forms the basis for Algorithm 5 outperforming Algorithm 4, since it implies that Algorithm 5 can traverse a smaller MBAG. Experiments (Figure 5) show that $pbcpc$+ is significantly smaller than $pbcpc$—for example, smaller by an order of magnitude, and Algorithm 5 can be an order of magnitude faster than Algorithm 4 (Table 2).

5 Novel (α, β)-Biclique based Solution

Existing BCPC detection methods are all based on the MBAG approach. In the previous section, we reduce the MBAG using partial-BCPCs, but it is still essentially based on MBAG and remains inefficient for certain datasets where MBAG is extremely large. This section introduces a novel BCPC detection framework, which can also leverage the partial-BCPCs proposed in Section 4 to accelerate the detection process. We first introduce this framework.

5.1 The Basic Framework

Key idea. Note that in the previous section, each edge in the MBAG essentially represents an (a, b)-biclique, where $a \geq \alpha$ and $b \geq \beta$. A straightforward idea is to enumerate all (α, β)-bicliques in the bipartite graph, and for each enumerated (α, β)-biclique B, connect all maximal bicliques containing B using union-find set.

Implementation. Algorithm 6 presents the basic framework of the (α, β)-biclique based solution. It is built upon the (α, β)-biclique enumeration framework proposed in [30], where the core idea is to enumerate combinations of vertices in the U set while maintaining their common neighbors in the V set. This framework constructs an ordinary directed graph, called the *2-hop graph*, on the vertices of the U set to accelerate the enumeration process.

In Lines 2–3, Algorithm 6 prepares the maximal bicliques that need to be connected. In Line 4, it constructs the 2-hop graph according to [30]. Then, in Line 5, it executes the (α, β)-biclique enumeration framework. Within BicliqueListing, for each enumerated (α, β)-biclique (Line 9), all maximal bicliques that share this biclique are connected together.

Analysis. Now we proceed to analyze the correctness and complexity of Algorithm 6.

Theorem 9. *Algorithm 6 correctly computes all* BCPC*s.*

Proof. This theorem can be easily proven as follows: Algorithm 6 is capable of enumerating all (α, β)-bicliques and connecting all maximal bicliques that share these bicliques. Consequently, the final BCPCs are correct. □

Theorem 10. *The time and space complexity of Algorithm 6 are $O(a(\vec{H})^{\alpha-2}|\vec{E_h}||V| + \Delta n_{biclique})$ and $O(n_{biclique} n)$ respectively, where $n = |U| + |V|$, $a(\vec{H})$ is the arboricity of $\vec{H}$ [30], Δ is the number of (α, β)-bicliques, $n_{biclique}$ is the number of maximal bicliques.*

Proof. The time complexity of maximal biclique enumeration is $T_{biclique} = 2^{n/2}|E|$ [9]. The time complexity of enumerating (α, β)-biclique is $O(a(\vec{H})^{\alpha-2}|\vec{E_h}||V| + \Delta)$ [30]. Algorithm 6 connects all maximal bicliques sharing an (α, β)-biclique (Line 9), thus, the time complexity of Algorithm 6 is $O(T_{biclique} + a(\vec{H})^{\alpha-2}|\vec{E_h}||V| + \Delta n_{biclique})$. Since $O(n_{biclique}) = O(2^{n/2})$ [9, 24], $O(\Delta) = O(n^{\alpha+\beta})$ [30], the time complexity can be rewritten as $O(a(\vec{H})^{\alpha-2}|\vec{E_h}||V| + \Delta n_{biclique})$.

Algorithm 6 does not store all (α, β)-bicliques, thus, the space complexity of Algorithm 6 is $O(n_{biclique} n)$. □

5.2 Pruning with Maximal Bicliques and Partial-BCPCs

Challenges in the basic framework. It is evident that the basic framework in Algorithm 6 is highly inefficient, as the number of (α, β)-bicliques grows exponentially with respect to α and β [30]. Therefore, enumerating all (α, β)-bicliques and connecting the corresponding maximal bicliques is impractical.

Pruning with Maximal Bicliques. The key to addressing the above challenge is to reduce the search space in the enumeration process of (α, β)-bicliques — that is, to prune in advance those branches that do not contribute to connecting multiple maximal bicliques.

Theorem 11. *Given a bipartite graph G, for a biclique B, if the number of maximal bicliques sharing it is 0, or all those maximal bicliques are in the same* BCPC*, then for any other biclique B' such that $B \subseteq B'$, the number of maximal bicliques sharing B' is also 0, or they are in the same* BCPC*.*

Proof. It is easy to see that any maximal biclique containing B' must also contain B, so the set of maximal bicliques containing B' is a subset of those containing B. Therefore, Theorem 11 can be easily proved. □

Theorem 11 reveals the core idea of pruning the (α, β)-biclique enumeration space: during the enumeration process, for each intermediate small biclique generated, we can maintain the set of maximal bicliques containing it. If this set is empty or all the maximal bicliques in the set already belong to the same BCPC, then the current enumeration branch can be safely pruned.

Algorithm 7: (α, β)-Biclique based solution with maximal biclique

Input: Bipartite graph $G = (U, V, E)$; Two integer α, β
Output: All BCPC in G
1 Let UF be an initial union-find set;
2 $\mathcal{B} \leftarrow$ set of all maximal bicliques in G;
3 $\mathcal{B} \leftarrow \{B | B \in \mathcal{B}, |B.X| \geq \alpha, |B.Y| \geq \beta\}$;
4 Construct 2-hop graph $\vec{H} = (U, \vec{E_h})$ on vertex set U [30]; /* $\vec{H} = (U, \vec{E_h})$: a direct graph, $(u, v) \in \vec{E_h}$ indicates that $u < v$, u, v have common neighbors in V */
5 BicliqueListing+ $(\vec{H}, \emptyset, V)$;
6 **return** UF;
7 **Procedure** BicliqueListing+ $(\vec{H}, R_U, C_V)$
8 $RB \leftarrow \{B | R_U \subseteq B.X, B \in \mathcal{B}\}$; /* maximal bicliques sharing R_U */
9 **if** $|\{UF.find(B) | B \in RB\}| \leq 1$ **then return**; /* there is only 1 or 0 UF disjoint set in RB */
10 **if** $|R_U| = \alpha$ **then**
11 **foreach** $R_V \subseteq C_V, |R_V| = \beta$ **do**
12 $RB \leftarrow \{B | R_V \subseteq B.Y, B \in RB\}$;
13 **if** $|\{UF.find(B) | B \in RB\}| \leq 1$ **then continue**; /* there is only 1 or 0 UF disjoint set in RB */
14 Connect maximal bicliques in RB with $UF.union(*)$;
15 **return**;
16 **foreach** $u \in \vec{H}$ **do**
17 $C'_V \leftarrow C_V \cap Nei_G(u)$;
18 **if** $|C'_V| < \beta$ *or* $Nei_{\vec{H}}(u) < \alpha - |R_U| - 1$ **then continue**;
19 Let $\vec{H}'$ be induced subgraph of $\vec{H}$ on $Nei_{\vec{H}}(u)$;
20 BicliqueListing+ $(\vec{H}', R_U \cup \{u\}, C'_V)$;

Implementation. Algorithm 7 presents the implementation of pruning the (α, β)-biclique enumeration process using maximal bicliques. The difference between Algorithm 7 and Algorithm 6 lies in the BicliqueListing+ function. Specifically, in Line 8, BicliqueListing+ collects maximal bicliques that share $(R_U, \emptyset)$. In Line 9, if the maximal bicliques in RB belong to only one or zero union-find sets, then the entire enumeration branch can be pruned. In Lines 12–13, the algorithm further checks whether the maximal bicliques in RB that share R_V belong to more than one union-find set. If so, it connects them using $UF.union(*)$.

Analysis. Now we proceed to analyze the correctness and complexity of Algorithm 7.

Theorem 12. *Algorithm 7 correctly computes all* BCPC*s.*

Proof. Compared with Algorithm 6, Algorithm 7 prunes the branches that can not connect multiple UF sets based on Theorem 11. Therefore, Algorithm 7 maintains the same correctness as Algorithm 6. □

Theorem 13. *The time and space complexity of Algorithm 7 are $O((a(\vec{H}_m)^{\alpha-2}|\vec{E}_{hm}||V_m| + \Delta_m)n_{biclique})$ and $O(n_{biclique}n)$ respectively. $n = |U| + |V|$. Let U_m and V_m be the vertex sets shared by any two maximal bicliques, G_m be the induced subgraph on U_m and V_m, and $\vec{H}_m = (U_m, \vec{E}_{hm})$ be the 2-hop graph based on G_m. $a(\vec{H}_m)$ is the arboricity of $\vec{H}_m$ [30], Δ_m is the number of (α, β)-bicliques in G_m, $n_{biclique}$ is the number of maximal bicliques.*

Proof. The time complexity of maximal biclique enumeration is $T_{biclique} = 2^{n/2}|E|$ [9]. Compared with Algorithm 6, Algorithm 7 prunes the branches that can not connect multiple UF sets based on Theorem 11. This implies that vertices initially contained in only one maximal biclique need not be considered during the (α, β)-biclique enumeration process. Therefore, Algorithm 7 can be regarded as performing (α, β)-biclique enumeration on G_m. Considering Lines 8-9 and Lines 12-13 of Algorithm 7, its time complexity is $O(T_{biclique} + (a(\vec{H}_m)^{\alpha-2}|\vec{E}_{hm}||V_m| + \Delta_m)n_{biclique})$. Since $O(n_{biclique}) = O(2^{n/2})$ [9, 24], $O(\Delta_m) = O((U_m + V_m)^{\alpha+\beta})$ [30], the time complexity can be rewritten as $O((a(\vec{H}_m)^{\alpha-2}|\vec{E}_{hm}||V_m| + \Delta_m)n_{biclique})$.

Similarly, the space complexity of Algorithm 7 is still $O(n_{biclique}n)$. □

Algorithm 8: (α, β)-Biclique based solution with partial-BCPC

Input: Bipartite graph $G = (U, V, E)$; Two integer α, β
Output: All BCPC in G
1 Lines 1-3 of Algorithm 5; /* partial-BCPC computation */
2 Lines 2-5 of Algorithm 7; /* connect maximal bicliques by (α, β)-biclique listing */
3 **return** UF;

Pruning with Partial-BCPC**.** As shown in Section 4, a partial-BCPC is an incomplete BCPC. Based on the partial-BCPCs, we can determine in advance that certain maximal bicliques belong to the same BCPC. Therefore, they enable the (α, β)-biclique enumeration process in Algorithm 7 to reach pruning conditions earlier.

Implementation. Algorithm 8 is the algorithm that leverages partial-BCPCs, and it consists of two main parts. First, it computes the partial-BCPC (Line 1), which are recorded in the union-find structure (UF). Then, in Line 2, these partial-BCPCs are used to further prune the (α, β)-biclique listing process.

Analysis. Now we proceed to analyze the correctness and complexity of Algorithm 8.

Theorem 14. *Algorithm 8 correctly computes all* BCPC*s.*

Theorem 15. *The time and space complexity of Algorithm 8 are $O((a(\vec{H}_p)^{\alpha-2}|\vec{E}_{hp}||V_p| + \Delta_p)n_{biclique})$ and $O(n_{biclique}n)$ respectively. $n = |U| + |V|$. Let U_p and V_p be the vertex sets shared by any two partial-*BCPC*s, G_p be the induced subgraph on U_p and V_p, and $\vec{H}_p = (U_p, \vec{E}_{hp})$ be the 2-hop graph based on G_p. $a(\vec{H}_p)$ is the arboricity of $\vec{H}_p$ [30], Δ_p is the number of (α, β)-bicliques in G_p, $n_{biclique}$ is the number of maximal bicliques.*

The proves of Theorem 14 and Theorem 15 are similar to those of Theorem 12 and Theorem 13.

6 Experiments

6.1 Experimental Setup

Datasets. We use 10 real-world bipartite graph datasets, whose detailed information is presented in Table 1. All the datasets are downloaded from http://konect.cc/.

Algorithms. We first implement the state-of-the-art algorithm in [6, 14] as MBAG.

- MBAG (Algorithm 3) is directly based on the maximal biclique adjacency graph (also abbreviated as MBAG). It first

Table 1: Datasets, $\overline{d}_U$ is the average degree of vertices in U, $\overline{d}_V$ is the average degree of vertices in V, N_m is the number of maximal bicliques, N_o is the number of edges in maximal biclique adjacent graph, K=10^3, M=10^6, B=10^9, '-' means timeout

Datasets	$\|U\|$	$\|V\|$	$\|E\|$	$\overline{d}_U$	$\overline{d}_V$	N_m	N_o
Youtube	94,238	30,087	293,360	3.11	9.75	1,769,331	2B
Bookcrossing	77,802	185,955	433,652	5.57	2.33	155,391	1B
Github	56,519	120,867	440,237	7.79	3.64	55,260,550	-
Citeseer	105,353	181,395	512,267	4.86	2.82	54,083	962K
Stackoverflow	545,195	96,678	1,301,942	2.39	13.47	2,922,148	26B
Twitter	175,214	530,418	1,890,661	10.79	3.56	5,102,542	13B
Imdb	685,568	186,414	2,715,604	3.96	14.57	1,809,175	1B
Actor2	303,617	896,302	3,782,463	12.46	4.22	3,761,666	2B
Amazon	2,146,057	1,230,915	5,743,258	2.68	4.67	5,702,211	108B
DBLP	1,953,085	5,624,219	12,282,059	6.29	2.18	1,281,831	85M

enumerates all maximal bicliques, then constructs the maximal biclique adjacency graph using these bicliques, and finally traverses this adjacency graph to compute the BCPC.

We then implement the following partial-BCPC based solution and (α, β)-biclique based solution.

- PBCPC (Algorithm 4) is the basic partial-BCPC based solution. It first enumerates maximal bicliques, and builds the MBE tree. For each (α, β)-node in the MBE tree, it identifies all maximal bicliques associated with that node and connects them into a single partial-BCPC. It then traverses the MBAG reduced by the partial-BCPCs to obtain the final BCPC.
- PBCPC+ (Algorithm 5) is an improved version of PBCPC. The key difference lies in its use of stop-labels to reduce the search overhead when identifying all maximal bicliques associated with an (α, β)-node.
- Biclique (Algorithm 6) is the basic solution based on (α, β)-biclique enumeration. Its core idea is to enumerate all (α, β)-bicliques, and for each such biclique, connect all maximal bicliques that contain it in order to construct the BCPC.
- BicliqueM (Algorithm 7) is an improved version of Biclique. It prunes (α, β)-biclique enumeration branches by checking whether the maximal bicliques associated with the currently enumerated biclique already belong to the same BCPC.
- BicliqueP (Algorithm 8) is also an improved version of Biclique. Unlike BicliqueM, it leverages the method from PBCPC+ to compute partial-BCPCs in advance, thereby identifying some maximal bicliques that already belong to the same BCPC. This enables a stronger pruning effect.

Parameters. There are two parameters used in the experiments: α and β. Their default values are both set to 4, and the possible values they can take are 2, 4, 6, and 8.

All the algorithms are implemented in C++. The experiments are conducted on a system running Ubuntu 20.04.4 LTS with an AMD Ryzen 3990X 2.2GHz CPU and 256GB of memory.

6.2 Experimental Results

Exp-1: Efficiency of various algorithms. In this experiment, we evaluated the performance of all algorithms across all datasets, and the results are presented in Table 2.

Overall, MBAG and PBCPC algorithms perform the worst in terms of runtime across most datasets and α, β. This is mainly because the MBAG algorithm traverses the original maximal biclique adjacency graph (MBAG), which is often very large (see Table 1). Although the PBCPC algorithm attempts to reduce the size of the MBAG by computing partial-BCPCs, these partial-BCPCs are not sufficiently comprehensive, leading to limited pruning effectiveness on the MBAG (we will explore the effectiveness of these partial-BCPCs in Exp-3).

PBCPC+ performs slightly better than MBAG and PBCPC, especially when α and β are small. For instance, on dataset Youtube, when $\alpha = \beta = 2$, PBCPC+ outperforms MBAG and PBCPC by more than one order of magnitude. This is because when α and β are small, there is a larger number of maximal bicliques that are larger than (α, β)-bicliques, which increases the scale of maximal biclique adjacency graph. In such cases, the effect of PBCPC+ using partial-BCPC to reduce the size of the adjacency graph is more prominently demonstrated.

Biclique achieves relatively good performance only when both α and β are small—for example, on datasets Youtube, Bookcrossing, and DBLP with $\alpha = \beta = 2$. When α or β increases, the performance of Biclique drops sharply. This is because the number of (α, β)-bicliques grows exponentially as α and β increase.

BicliqueM and BicliqueP are the two optimal algorithms in almost all cases, and they can outperform MBAG by nearly three orders of magnitude. For example, on dataset Youtube, when $\alpha = \beta = 2$, BicliqueP only takes 34 seconds, while MBAG requires 13,376 seconds. Furthermore, BicliqueP performs better than BicliqueM, and this advantage is more pronounced on dataset Github (under many parameters, only BicliqueP can complete within 7 days, i.e., 604,800 seconds). This is because BicliqueP uses partial-BCPC for pruning, which has stronger pruning capability compared to maximal biclique (see Exp-4).

Exp-2: Overhead of computing partial-BCPC. This experiment evaluates the overhead of computing partial-BCPCs (with maximal biclique enumeration included, which is a necessary step). Specifically, we compare the time spent on computing partial-BCPCs in the algorithms PBCPC and PBCPC+, and also compare it against the time spent on maximal biclique enumeration. The experimental results are shown in Figure 4.

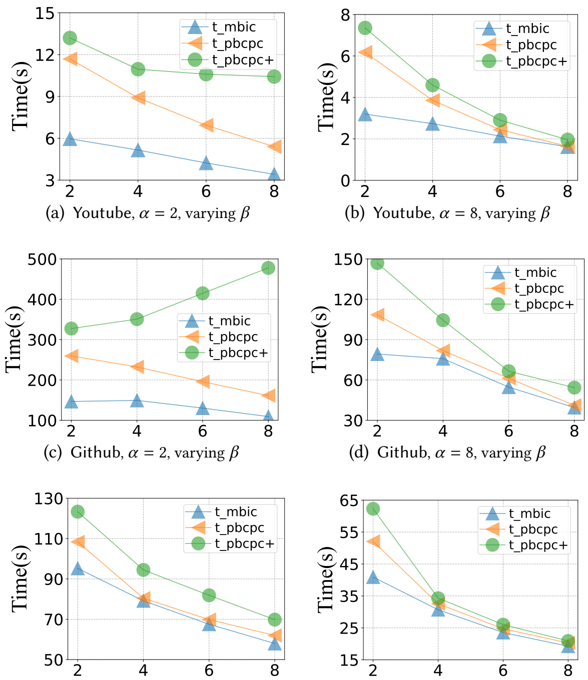
As shown in Figure 4, although computing partial-BCPCs introduces additional overhead on top of maximal biclique enumeration, the added cost is manageable. For example, on datasets Youtube and Twitter, t_pbcpc and t_pbcpc+ are only slightly higher than the time spent on maximal biclique enumeration alone (t_mbic). For the Github dataset, due to the larger number of maximal bicliques (see Table 1) and larger scale of the MBE tree, t_pbcpc+ is no longer close to t_mbic and t_pbcpc. However, this overhead is meaningful—our best-performing algorithm, BicliqueP, is built upon the partial-BCPC computed in PBCPC+ (the time spent is also t_pbcpc+). Notably, BicliqueP is the only algorithm that successfully passed all tests on the Github dataset.

Table 2: Performance (s) of all algorithms on all datasets. The best and sub-best results are bold and underlined respectively. "-" denotes out of time (≥ 7 days)

Datasets		Youtube				Bookcrossing				Github				Citeseer				Stackoverflow			
α	Algo \ β	2	4	6	8	2	4	6	8	2	4	6	8	2	4	6	8	2	4	6	8
2	MBAG	13,376	11,492	6,072	1,902	35	2	0.9	0.7	-	-	-	-	1	0.8	0.7	0.6	32,598	17,967	4,287	844
	PBCPC	4,651	7,283	6,942	3,207	42	3	1	1	-	-	-	-	1	0.8	0.7	0.6	30,051	25,220	8,811	2,177
	PBCPC+	197	1,914	2,787	1,484	20	3	1	0.8	44,276	-	-	-	1	0.9	0.8	0.6	2,564	7,144	3,467	987
	Biclique	135	358	2,995	-	3	26	93,773	-	-	-	-	-	1	1	64	6,999	2,038	3,118	-	-
	BicliqueM	59	70	43	22	**2**	**1**	**0.8**	**0.6**	24,581	-	-	-	1	**0.8**	**0.7**	0.6	486	208	**119**	**80**
	BicliqueP	**34**	**43**	**31**	**20**	3	2	1	1	**7,718**	**34,478**	**33,948**	**14,272**	**1**	0.9	0.8	**0.6**	**339**	**159**	121	103
4	MBAG	13,113	11,358	5,941	1,793	3	**0.7**	**0.5**	**0.4**	-	-	-	-	0.1	0.08	0.07	0.06	27,582	14,496	3,308	544
	PBCPC	7,748	9,568	7,802	3,197	4	0.9	0.6	0.5	-	-	-	-	0.1	0.09	0.08	0.06	37,632	25,336	7,496	1,476
	PBCPC+	1,450	2,964	3,404	1,618	4	0.8	0.6	0.5	-	-	-	-	0.1	**0.08**	0.07	0.06	13,222	9,955	3,857	864
	Biclique	3,754	5,445	5,522	2,877	2	1	0.8	0.6	-	-	-	-	0.1	0.1	0.1	0.3	1,210	5,446	748	11,595
	BicliqueM	54	235	140	50	1	0.8	0.6	0.5	-	-	-	412,886	0.1	0.09	**0.07**	**0.06**	306	539	199	56
	BicliqueP	**28**	**66**	**49**	**24**	**1**	0.9	0.6	0.5	**11,483**	**145,121**	**424,317**	**157,465**	**0.1**	0.09	0.08	0.07	**190**	**159**	**76**	**34**
6	MBAG	10,936	9,409	4,635	1,312	1	0.6	0.5	0.4	-	-	-	-	**0.03**	**0.03**	**0.03**	**0.03**	9,142	4,150	561	45
	PBCPC	7,736	8,370	5,855	2,080	1	0.7	0.5	0.4	-	-	-	-	0.05	0.04	0.04	0.04	12,482	6,893	1,134	89
	PBCPC+	2,500	3,622	3,133	1,121	1	0.7	0.5	0.4	-	-	-	-	0.04	0.03	0.04	0.03	5,758	3,833	775	76
	Biclique	-	3,289	34,084	3,352	36	0.9	0.5	0.4	-	-	-	-	0.07	0.04	0.04	0.04	-	1,764	2,724	26
	BicliqueM	44	165	178	35	1	**0.6**	0.5	0.4	13,182	444,343	-	55,394	0.04	0.04	0.04	0.04	90	131	40	12
	BicliqueP	**24**	**57**	**56**	**16**	**1**	0.7	**0.5**	**0.4**	**9,450**	**141,003**	**158,184**	**20,785**	0.05	0.04	0.04	0.04	**75**	**72**	**22**	**10**
8	MBAG	7,236	6,102	2,832	744	1	**0.5**	0.4	0.3	-	-	-	-	0.03	0.03	0.03	0.03	1,377	398	21	7
	PBCPC	5,585	5,707	3,596	1,241	1	0.6	0.4	0.3	-	-	-	-	0.03	0.03	0.03	0.03	1,721	592	32	6
	PBCPC+	2,327	2,878	1,991	713	1	0.6	0.4	0.3	-	-	-	-	0.03	0.03	0.03	0.03	1,110	447	29	7
	Biclique	-	109,345	4,728	8,555	2,315	6	0.4	0.3	-	-	401,158	23,651	0.09	0.04	0.04	0.04	-	77	20	14
	BicliqueM	33	117	101	38	1	0.6	0.4	0.3	5,219	146,064	103,754	2,324	**0.03**	**0.03**	**0.03**	0.03	30	22	9	7
	BicliqueP	**19**	**45**	**33**	**13**	**1**	0.6	**0.4**	**0.3**	**4,066**	**61,855**	**36,416**	**1,252**	0.04	0.04	0.04	**0.03**	**30**	**16**	**9**	**6**
Datasets		Twitter				Imdb				Actor2				Amazon				DBLP			
α	Algo \ β	2	4	6	8	2	4	6	8	2	4	6	8	2	4	6	8	2	4	6	8
2	MBAG	11,652	5,290	1,817	657	859	706	622	409	7,303	6,650	5,217	3,502	16,282	3,974	580	137	18	**12**	10	9
	PBCPC	47,374	24,237	6,055	2,335	323	188	293	418	1,820	2,285	2,377	2,041	20,703	6,687	1,297	342	28	19	15	13
	PBCPC+	10,580	11,843	3,162	1,337	102	58	90	160	266	524	772	780	2,413	3,067	706	228	23	15	12	10
	Biclique	685	5,339	4,584	-	55	147	58,284	-	190	875	39,764	-	964	727	-	-	20	176	296,803	-
	BicliqueM	**510**	777	208	108	**35**	**26**	**23**	**21**	88	56	48	47	618	**157**	**79**	**55**	**17**	13	**10**	**9**
	BicliqueP	518	**402**	**152**	**94**	42	33	28	25	**79**	**51**	**44**	**39**	**590**	176	108	82	25	19	16	14
4	MBAG	4,339	2,737	1,236	457	762	687	614	403	7,107	6,586	5,178	3,468	7,856	1,720	227	57	**7**	5	4	3
	PBCPC	11,436	6,764	3,176	1,482	408	355	446	514	1,622	2,251	2,462	2,125	9,234	2,496	427	100	8	5	4	3
	PBCPC+	6,282	4,237	2,004	1,001	99	74	121	185	201	516	794	840	4,219	1,508	304	78	8	5	4	4
	Biclique	-	12,028	20,544	242,804	92	1,805	1,335	25,431	626	4,996	29,168	526,141	534	2,687	474	9,676	12	7	29	1,834
	BicliqueM	134	129	94	87	**22**	21	26	21	57	131	51	44	303	161	61	**39**	8	5	4	3
	BicliqueP	**134**	**100**	**69**	**64**	23	**18**	**16**	**15**	**47**	**48**	**34**	**31**	**280**	**111**	**61**	42	9	**5**	**4**	**3**
6	MBAG	2,184	1,596	891	298	623	600	536	340	6,859	6,376	5,085	3,343	1,576	349	63	32	3	2	2	2
	PBCPC	3,124	2,404	1,677	690	557	551	605	568	1,988	2,585	2,583	2,270	1,669	499	97	44	3	2	2	2
	PBCPC+	2,666	1,469	1,073	506	169	164	204	237	326	641	933	955	1,035	331	71	36	3	3	2	2
	Biclique	-	-	134,078	27,608	1,748	835	11,021	10,968	194,453	22,912	49,668	137,995	33,212	1,595	525	806	10,632	6	17	30
	BicliqueM	**90**	81	68	52	**17**	17	22	23	51	121	62	52	**171**	80	**44**	**32**	3	2	2	2
	BicliqueP	95	**58**	**51**	**41**	19	**14**	**14**	**13**	**40**	**43**	**38**	**32**	213	**79**	47	33	**3**	**2**	**2**	**2**
8	MBAG	1,482	1,166	651	210	400	388	340	187	6,381	5,838	4,529	2,958	445	104	41	28	1	1	1	1
	PBCPC	1,479	1,297	994	369	440	437	450	331	2,631	2,922	2,867	2,398	539	143	54	34	1	1	1	2
	PBCPC+	880	729	669	271	187	186	209	193	625	894	1,093	1,023	338	101	45	31	2	1	1	1
	Biclique	-	-	-	190,333	355,545	2,375	1,985	11,003	-	-	-	-	-	29,402	3,606	355	-	25	61	93
	BicliqueM	**74**	65	53	36	**15**	13	16	16	46	100	151	66	**143**	**59**	**38**	**28**	**1**	1	1	1
	BicliqueP	79	**45**	**36**	**28**	16	**12**	**12**	**11**	**35**	**38**	**46**	**36**	188	64	40	29	2	**1**	**1**	**1**

Exp-3: Effectiveness of partial-BCPC **in partial-**BCPC **based solution.** This experiment investigates the effectiveness of partial-BCPC in PBCPC and PBCPC+. Specifically, we compare the number of partial-BCPCs obtained by the two algorithms with the number of maximal bicliques and the number of BCPCs. The results are shown in Figure 5. We can see that in all cases, the inequality in Theorem 3 and 7 holds: n_mbic ≥ n_pbcpc ≥ n_pbcpc+ ≥ BCPC. We can also observe that compared to n_mbic and n_pbcpc, n_pbcpc+ is much closer to BCPC, and in some cases, it is more than an order of magnitude smaller than n_mbic (see Figure 5 (a), (c)). In both the PBCPC and PBCPC+ algorithms, vertices in the MBAG are regrouped according to the partial-BCPCs, meaning that the number of vertices in the reduced MBAG equals the number of partial-BCPCs. As a result, PBCPC+ outperforms PBCPC on nearly all datasets.

Exp-4: Effectiveness of partial-BCPC **in (α, β)-biclique based solution.** This experiment investigates the pruning effectiveness of maximal biclique and partial-BCPC in the (α, β)-biclique enumeration process. Figure 6 shows the number of enumerated nodes in the (α, β)-biclique enumeration trees of three algorithms: Biclique, BicliqueM, and BicliqueP. It can be observed that in Biclique (without any pruning), the number of nodes in the enumeration tree is more than four orders of magnitude higher than that in the other



(a) Youtube, $\alpha = 2$, varying β (b) Youtube, $\alpha = 8$, varying β

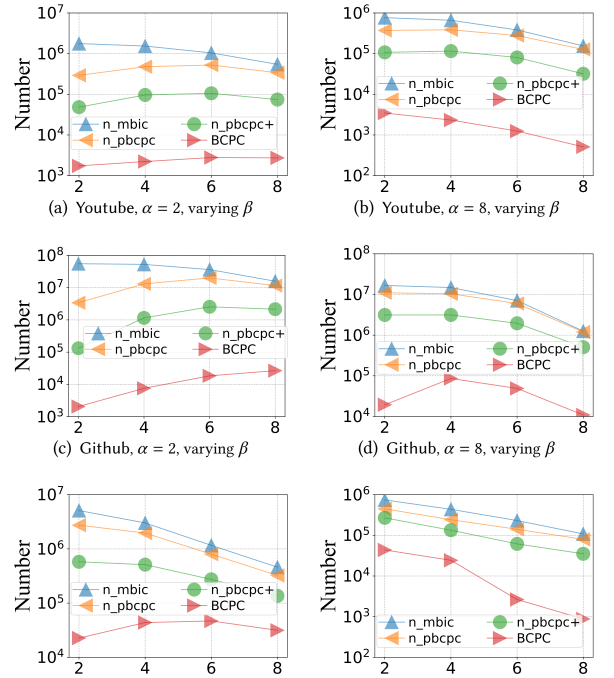
(c) Github, $\alpha = 2$, varying β (d) Github, $\alpha = 8$, varying β

(e) Twitter, $\alpha = 2$, varying β (f) Twitter, $\alpha = 8$, varying β

Figure 4: Time spent on maximal biclique enumeration (t_mbic), partial-BCPC computation (including maximal biclique enumeration as a necessary step) in PBCPC (t_pbcpc) and PBCPC+ (t_pbcpc+)

Table 3: Scalability test of PBCPC+ (s)

	Github	Imdb	Actor2	Amazon	DBLP
20%	5,779.7	3.4	1.2	8.3	0.8
40%	72,202.2	7.8	5.7	54.5	1.4
60%	119,608.0	13.5	14.7	85.2	2.8
80%	142,445.8	17.5	25.8	107.3	4.5
100%	145,121.7	18.2	49.0	111.2	5.8



(a) Youtube, $\alpha = 2$, varying β (b) Youtube, $\alpha = 8$, varying β

(c) Github, $\alpha = 2$, varying β (d) Github, $\alpha = 8$, varying β

(e) Twitter, $\alpha = 2$, varying β (f) Twitter, $\alpha = 8$, varying β

Figure 5: Number of maximal bicliques (n_mbic, where the size of X set and Y set is at least α and β respectively), partial-BCPCs obtained by PBCPC (n_pbcpc) and PBCPC+ (n_pbcpc+), and number of BCPCs

Table 4: Memory usage (MB) of various algorithms

	MBAG	PBCPC	PBCPC+	Biclique	BicliqueM	BicliqueP
Github	10,051	26,965	26,418	9,904	**9,904**	22,158
Imdb	**233**	364	361	260	260	343
Actor2	**409**	725	720	449	449	639
Amazon	**711**	1,228	1,207	797	772	1,130
DBLP	935	957	970	1,200	**935**	940

two algorithms, which sufficiently demonstrates the pruning capabilities of maximal biclique and partial-BCPC in (α, β)-biclique enumeration process. Among BicliqueM and BicliqueP, the partial-BCPC in BicliqueP exhibits the strongest pruning capability, as the number of enumerated nodes in BicliqueP is another order of magnitude smaller than that in BicliqueM (e.g., when $\beta = 2$ in Figure 6 (c)). This explains why BicliqueP achieves the best performance in most cases in Table 2, particularly with the dataset Github.

Exp-5: Scalability. This experiment explores the scalability of our best algorithm, BicliqueP. Table 3 presents the performance of BicliqueP on Github and four other large datasets. Specifically, we sampled 20%, 40%, 60%, and 80% of the edges from each dataset respectively. It can be observed that as the dataset size increases, the growth of running time of BicliqueP is manageable, demonstrating that BicliqueP has excellent scalability.

Exp-6: Memory. Table 4 presents the memory usage of all algorithms on the largest four datasets and dataset Github, which contains the maximum number of maximal bicliques. It can be observed that across all datasets, the three algorithms—MBAG, Biclique, and BicliqueM —consistently consume the least memory. This is because these algorithms do not require storing the MBE tree. However, although our best algorithm, BicliqueP, also stores the MBE tree, its memory consumption remains manageable.

Exp-7: Case study. We conduct a case study on the DBLP dataset. This dataset includes papers from major journals (TODS, TOIS, TKDE, VLDBJ) and conferences (SIGMOD, SIGKDD, ICDE, SIGIR, VLDB) in the database field, where vertices represent papers and authors respectively, and an edge (u, v) indicates that v is one of the authors of u. We compare (α, β)-BCPC with maximum biclique, k-bitruss [28], and (α, β)-core [18].

Given a bipartite graph $G = (U, V, E)$, (α, β)-core of G is a maximal subgraph of G, $G_1 = (U_1, V_1, E_1)$, that $\forall u \in U_1, |Nei_{G'}(u)| \geq \alpha, \forall v \in V_1, |Nei_{G'}(v)| \geq \beta$. k-bitruss is a maximal subgraph of G, $G_2 = (U_2, V_2, E_2)$, that each edge in E_2 is contained in at least k butterflies (butterfly is a subgraph with four vertices: $u, w \in U_2, v, x \in V_2$, $(u, v) \in E_2, (u, x) \in E_2, (w, v) \in E_2, (w, x) \in E_2$).

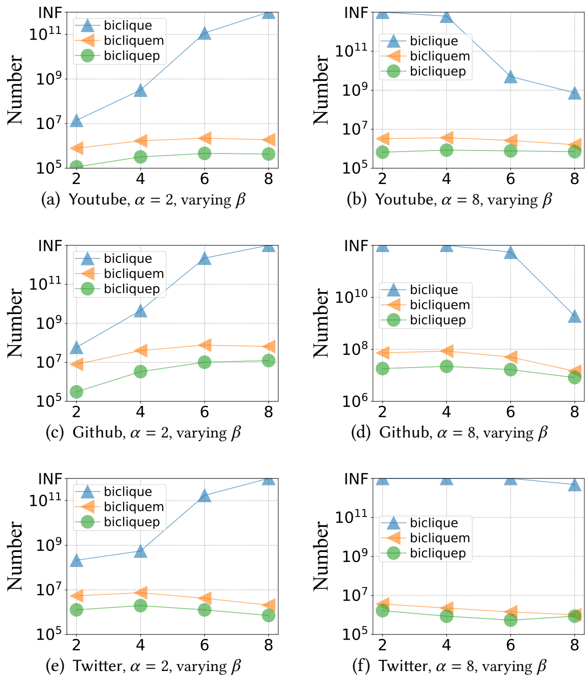


Figure 6: Number of enumerated nodes in the (α, β)-biclique enumeration trees of three algorithms: Biclique (biclique), BicliqueM (bicliquem), **and** BicliqueP (bicliquep)

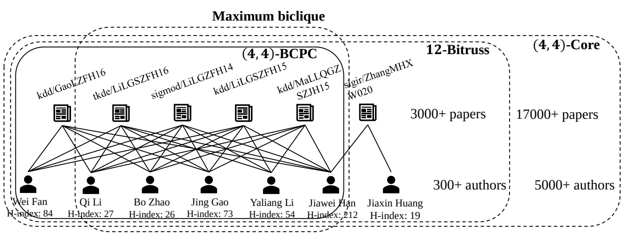


Figure 7: Case study on DBLP

In this case study, $\alpha = 4, \beta = 4, k = 12$. By Definition 4, the smallest maximal bicliques in (4, 4)-BCPC are either (5, 4)-bicliques or (4, 5)-bicliques, and the edges within these maximal bicliques exist in at least $3 \times 4 = 12$ butterflies, thus, we set $k = 12$ for k-bitruss.

Figure 7 presents the results containing *Jiawei Han* across all models. Among them, the result from maximum biclique is the smallest. Compared with (4, 4)-BCPC, it lacks one author, *Wei Fan*, who is one of *Jiawei Han*'s collaborators on Google Scholar. Although 12-bitruss and (4, 4)-core include the results of (4, 4)-BCPC, they also contain a large number of other vertices, making the usability of these two models far inferior to that of (4, 4)-BCPC.

7 Related Work

The research related to BCPC is the most relevant to our work. Initially, [14] proposed the BCPC model and introduced a method to detect BCPCs by traversing the maximal biclique adjacency graph (MBAG). Subsequently, [27] developed an online web toolkit, implementing the approach presented in [14]. In the context of social network analysis, [11] applied BCPC to identify clusters of users and entities. Regarding Wikipedia networks, [13] leveraged BCPC to explore the relationships between editors and articles, revealing that topics inherently aggregate editors. [15] applied BCPC to enterprise networks to analyze interaction patterns between hosts and users, thereby supporting the detection of abnormal user behaviors. Meanwhile, [12, 33] utilized BCPC to study the relationships between students and online learning resources, delving into the temporal evolution patterns between students and courses. Additionally, [6] investigated the problem of personalized BCPC search based on the BCPC model.

Our work is also related to maximal biclique enumeration and (p, q)-biclique enumeration. Existing methods for maximal biclique enumeration, a foundational task supporting BCPC model implementation, are primarily rooted in the idea of enumerating the powerset of vertex set [32]. Building on this framework, [1] introduced the pivot pruning strategy, which effectively reduces redundant search space and improves the efficiency. To further optimize performance, [5] proposed a method that optimizes the vertex enumeration order, enhancing computational speed. Additionally, [9] developed a unified enumeration framework suitable for hereditary cohesive subgraphs. This framework achieves state-of-the-art performance in maximal biclique enumeration problem, providing a more efficient and generalizable solution for related tasks. Key works on (p, q)-biclique enumeration include [23, 30]. [23] uses (p, q)-biclique enumeration only as a sub-process for broader graph analysis, while [30] focuses specifically on optimizing this enumeration task. Additionally, studies like [31] address the exact and approximate counting of (p, q)-bicliques.

There are also several cohesive subgraph models relevant to our work, such as (α, β)-core and k-bitruss. For (α, β)-core, key works include [17–19], which focus on efficient computation and distributed implementation of this structure. In terms of k-bitruss, studies such as [16, 28, 34] contribute to optimizing its decomposition and analysis processes. For maximum biclique, [21, 29] propose efficient methods for its identification. Beyond specific subgraph structures, modularity-based modeling approaches (e.g., [2, 3]) and label propagation methods (e.g., [20]) are also studied.

8 Conclusion

In this work, we tackle the problem of biclique percolation community (BCPC) detection in bipartite network. Confronting the high time complexity of existing BCPC detection methods, which stems from the potentially massive maximal biclique adjacency graph (MBAG), we introduce two innovative solutions. First, the partial-BCPC based solution reduces redundant computations by grouping maximal bicliques into incomplete BCPCs, lessening the burden of explicitly representing the entire MBAG. Second, our novel method based on (α, β)-biclique enumeration detects BCPCs by enumerating all (α, β)-bicliques and connecting maximal bicliques that share such bicliques. Leveraging partial-BCPC, this approach significantly prunes the (α, β)-biclique enumeration space. Experimental results demonstrate that our methods achieve remarkable performance, outperforming existing approaches by nearly three orders of magnitude. These advancements not only enhance the efficiency of BCPC detection but also pave the way for more scalable analysis of bipartite networks in diverse domains, including social networks and recommendation systems.

References

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

[2] Rudy Arthur. 2020. Modularity and projection of bipartite networks. *Physica A: Statistical Mechanics and its Applications* 549 (2020), 124341.

[3] Michael J Barber. 2007. Modularity and community detection in bipartite networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 76, 6 (2007), 066102.

[4] Chiara Carusi and Giuseppe Bianchi. 2020. A look at interdisciplinarity using bipartite scholar/journal networks. *Scientometrics* 122, 2 (2020), 867–894.

[5] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2022. Efficient maximal biclique enumeration for large sparse bipartite graphs. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1559–1571.

[6] Zi Chen, Yiwei Zhao, Long Yuan, Xuemin Lin, and Kai Wang. 2023. Index-based biclique percolation communities search on bipartite graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2699–2712.

[7] Chaitali Choudhary, Inder Singh, and Manoj Kumar. 2023. Community detection algorithms for recommendation systems: techniques and metrics. *Computing* 105, 2 (2023), 417–453.

[8] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

[9] Qiangqiang Dai, Rong-Hua Li, Xiaowei Ye, Meihao Liao, Weipeng Zhang, and Guoren Wang. 2023. Hereditary Cohesive Subgraphs Enumeration on Bipartite Graphs: The Power of Pivot-based Approaches. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.

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

[11] Tobias Hecking, Irene-Angelica Chounta, and H Ulrich Hoppe. 2015. Analysis of user roles and the emergence of themes in discussion forums. In *2015 Second European Network Intelligence Conference*. IEEE, 114–121.

[12] Tobias Hecking, Sabrina Ziebarth, and H Ulrich Hoppe. 2014. Analysis of dynamic resource access patterns in a blended learning course. In *Proceedings of the Fourth International Conference on Learning Analytics and Knowledge*. 173–182.

[13] Rut Jesus, Martin Schwartz, and Sune Lehmann. 2009. Bipartite networks of Wikipedia's articles and authors: a meso-level approach. In *Proceedings of the 5th international symposium on Wikis and open collaboration*. 1–10.

[14] Sune Lehmann, Martin Schwartz, and Lars Kai Hansen. 2008. Biclique communities. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 78, 1 (2008), 016108.

[15] Qi Liao, Aaron Striegel, and Nitesh Chawla. 2010. Visualizing graph dynamics and similarity for enterprise network security and management. In *Proceedings of the seventh international symposium on visualization for cyber security*. 34–45.

[16] Fengnian Lin, Boyu Ruan, Junhao Gan, and Lei Li. 2025. Efficient Bitruss Decomposition without Butterfly Enumeration. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 1718–1728.

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

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

[19] Qing Liu, Xuankun Liao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2023. Distributed (α, β)-core decomposition over bipartite graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 909–921.

[20] Xin Liu and Tsuyoshi Murata. 2010. An efficient algorithm for optimizing bipartite modularity in bipartite networks. *Journal of Advanced Computational Intelligence and Intelligent Informatics* 14, 4 (2010), 408–415.

[21] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum biclique search at billion scale. *Proceedings of the VLDB Endowment* (2020).

[22] Cristina Maier and Dan Simovici. 2022. Bipartite graphs and recommendation systems. *Journal of Advances in Information Technology-in print* (2022).

[23] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. 2015. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 815–824.

[24] Erich Prisner. 2000. Bicliques in graphs I: Bounds on their number. *Combinatorica* 20, 1 (2000), 109–117.

[25] Sara Rahiminejad, Mano R Maurya, and Shankar Subramaniam. 2019. Topological and functional comparison of community detection algorithms in biological networks. *BMC bioinformatics* 20, 1 (2019), 212.

[26] Karsten Steinhaeuser and Nitesh V Chawla. 2008. Community detection in a large real-world social network. In *Social computing, behavioral modeling, and prediction*. Springer, 168–175.

[27] Bei Wang, Jinyu Chen, and Shihua Zhang. 2018. BMTK: a toolkit for determining modules in biological bipartite networks. *Quantitative Biology* 6 (2018), 186–192.

[28] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient bitruss decomposition for large-scale bipartite graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 661–672.

[29] Kai Wang, Wenjie Zhang, Xuemin Lin, Lu Qin, and Alexander Zhou. 2022. Efficient personalized maximum biclique search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 498–511.

[30] Jianye Yang, Yun Peng, Dian Ouyang, Wenjie Zhang, Xuemin Lin, and Xiang Zhao. 2023. (p, q)-biclique counting and enumeration for large sparse bipartite graphs. *The VLDB Journal* 32, 5 (2023), 1137–1161.

[31] Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongchao Qin, and Guoren Wang. 2023. Efficient biclique counting in large bipartite graphs. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[32] 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 bioinformatics* 15, 1 (2014), 110.

[33] Sabrina Ziebarth, German Neubaum, Elias Kyewski, Nicole Kramer, H Ulrich Hoppe, Tobias Hecking, and Sabrina Eimler. 2015. Resource usage in online courses: Analyzing learner's active and passive participation patterns. International Society of the Learning Sciences, Inc.[ISLS].

[34] Zhaonian Zou. 2016. Bitruss decomposition of bipartite graphs. In *International conference on database systems for advanced applications*. Springer, 218–233.