Maximal Biclique Enumeration with Improved Worst-Case Time Complexity Guarantee: A Partition-Oriented Strategy

KAIXIN WANG, College of Computer Science, Beijing University of Technology, China
KAIQIANG YU*, State Key Laboratory of Novel Software Technology, Nanjing University, China
CHENG LONG, College of Computing and Data Science, Nanyang Technological University, Singapore

The maximal biclique enumeration problem in bipartite graphs is fundamental and has numerous applications in E-commerce and transaction networks, particularly in areas such as fraud detection and anomaly behavior identification. Most existing studies adopt a branch-and-bound framework, which recursively expands a partial biclique with a vertex until no further vertices can be added. Equipped with a basic pivot selection strategy, all state-of-the-art methods have a worst-case time complexity no better than $O(m \cdot (\sqrt{2})^n)$, where m and n are the number of edges and vertices in the graph, respectively. In this paper, we introduce a new branch-and-bound (BB) algorithm IPS. In IPS, we relax the strict stopping criterion of existing methods by allowing termination when all maximal bicliques within the current branch can be outputted in the time proportional to the number of maximal bicliques inside, reducing the total number of branches required. Second, to fully unleash the power of the new termination condition, we propose an improved pivot selection strategy, which well aligns with the new termination condition to achieve better theoretical and practical performance. Formally, IPS improves the worst-case time complexity to $O(m \cdot \alpha^n + n \cdot \beta)$, where $\alpha(\approx 1.3954)$ is the largest positive root of $x^4 - 2x - 1 = 0$ and β represents the number of maximal bicliques in the graph, respectively. This result surpasses that of all existing algorithms given that α is strictly smaller than $\sqrt{2}$ and β is at most $(\sqrt{2})^n - 2$ theoretically. Furthermore, we apply an inclusion-exclusion-based framework to boost the performance of IPS, improving the worst-case time complexity to $O(n \cdot \gamma^2 \cdot \alpha^\gamma + \gamma \cdot \beta)$ for large sparse graphs (γ is a parameter satisfying $\gamma \ll n$ for sparse graphs). Finally, we conduct extensive experiments on 15 real datasets and the results demonstrate that our algorithms can run several times faster than the state-of-the-art algorithms on real datasets.

CCS Concepts: • **Mathematics of computing → Graph algorithms**.

Additional Key Words and Phrases: Graph mining; branch-and-bound; maximal biclique enumeration

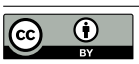
ACM Reference Format:
Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2026. Maximal Biclique Enumeration with Improved Worst-Case Time Complexity Guarantee: A Partition-Oriented Strategy. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 209 (June 2026), 24 pages. https://doi.org/10.1145/3802086

1 Introduction

A bipartite graph is a special type of graph where all vertices can be divided into two disjoint sets such that every edge connects a vertex in one set with another vertex in the other set. Bipartite graphs are widely used to model interactions between two different classes of objects in many

*Kaiqiang Yu is the corresponding author.

Authors' Contact Information: Kaixin Wang, College of Computer Science, Beijing University of Technology, China, kaixin.wang@bjut.edu.cn; Kaiqiang Yu, State Key Laboratory of Novel Software Technology, Nanjing University, China, kaiqiang.yu@nju.edu.cn; Cheng Long, College of Computing and Data Science, Nanyang Technological University, Singapore, c.long@ntu.edu.sg.



ACM 2836-6573/2026/6-ART209
https://doi.org/10.1145/3802086

real-world systems, such as purchasing networks between buyers and sellers [32], collaboration networks between authors and publications [15, 45], and recommendation networks between users and items [17]. Given a bipartite graph $G = (L \cup R, E)$, a biclique is a complete subgraph in which every vertex on one side is connected to all vertices on the other side. Among all bicliques, maximal bicliques (MBCs) are of particular interest and play a central role in applications including community analysis [14, 18], recommendation systems [21, 22], fraud detection [41], and graph neural network inference acceleration [12].

Motivations and Applications. Although the number of maximal bicliques can be exponentially large in the worst case, complete enumeration remains essential for many real-world data analysis tasks, as some examples illustrate below.

Community Discovery and Fraud Pattern Mining. In many applications, maximal bicliques serve as fundamental structural patterns for discovering dense and meaningful interactions. A common modeling paradigm in such applications is to construct a bipartite graph between two types of entities (e.g., users and attributes in social systems, or entities and activities in transaction networks), where a maximal biclique corresponds to a largest group of objects on one side that are tightly associated with the same group of objects on the other side. These structures naturally capture coherent behavioral patterns, such as communities of users sharing common attributes in social computing, or groups of entities involved in the same set of suspicious activities in fraud and anomaly detection. In both cases, the analytical objective is not merely to identify a single dominant pattern but to uncover all maximal such structures, since each may represent a distinct, meaningful interaction pattern.

Redundant Message Passing in Graph Neural Networks. Maximal biclique mining also arises in graph neural networks during message passing between consecutive layers. Here, a bipartite graph is formed between the source nodes in one layer and the target nodes in the next, where edges represent the dependencies of the message. A maximal biclique corresponds to a largest set of target nodes that receive messages from the same set of source nodes, exposing highly redundant computation patterns. Identifying such structures enables factorizing message propagation, reducing memory access, and accelerating training and inference while preserving model accuracy. Complete maximal biclique enumeration ensures that all compressible interaction patterns are discovered, providing systematic opportunities for optimizing large-scale GNN systems.

Existing Methods and Challenges. Quite a few algorithms have been proposed for maximal biclique enumeration [2, 5, 7, 9]. These approaches all adopt a similar *branch-and-bound* (BB) framework, which involves recursively dividing the problem of enumerating all MBCs into several sub-problems through branching operations. Specifically, each problem corresponds to a search space (or a branch), represented by a triplet $B = (S, C, X)$, where S induces a partial biclique found so far, C contains the vertices that can be added to S to form a larger biclique and X contains the vertices that have already been considered and is used for checking duplications. All MBCs in an input graph $G = (L \cup R, E)$ can be enumerated via the above BB framework by solving the initial search space $B = (\emptyset, L \cup R, \emptyset)$. Given a branch $B = (S, C, X)$, a set of sub-branches rooted at B would be created, where each sub-branch expands S by adding a vertex v from one side of C to S and updates C and X by removing from C and X those vertices on the other side that are not connected with v, respectively. The recursive procedure would terminate when the branch can be *trivially solvable*, i.e., C is empty, at which point S is reported as a MBC if X is also empty; otherwise, S is not a MBC.

We term the above BB framework, which conducts branching until it reaches trivially solvable branches, as TS. We note that the theoretical time complexity and empirical runtime of TS-based algorithms are dominated by the branching process, as the number of branches can become

exponential in the worst case [2, 5, 7, 9]. Therefore, existing studies of TS target two directions to improve performance by reducing the number of branches created. One direction is based on the observation that all MBCs in a bipartite graph $G = (L \cup R, E)$ can be enumerated only by visiting some subsets in L's powerset [5, 44]. The rationale is that given a vertex set $S_L \in P_L$ where P_L is the powerset of L and let S_R contain all common neighbors of the vertices in S_L. Then $S_L \cup S_R$ induces an MBC if S_L also contains all common neighbors of the vertices in S_R. Therefore, it is common practice to choose the vertex set with the minimum number of vertices inside as L and visit its powerset recursively [5, 44]. The other direction is to prune more branches effectively by *pivoting* strategy [2, 7, 9]. The main idea is to first choose a vertex from $C \cup X$ as the pivot, and then prune those branches that would not contain any MBC inside based on some topological information of the graph, such as the containment relationship [2, 7] or the neighborhood information [9]. For example, given a branch $B = (S, C, X)$, the state-of-the-art pivoting strategy first chooses a vertex $v \in C \cup X$ as the pivot and the sub-branches produced by including (1) the vertex at the same side as that of v and (2) v's neighbors at the other side can be safely pruned [9]. To prune the maximum number of branches, the common practice is to choose the vertex with the maximum degree from $C \cup X$ as the pivot. By adopting the above techniques, existing algorithms achieve the state-of-the-art worst-case time complexity of $O(m \cdot (\sqrt{2})^n)$, where m and n denotes the number of edges and vertices of the graph, respectively. We note that it is challenging to further improve beyond this worst-case time complexity with this framework since a graph may contain up to $(\sqrt{2})^n - 2$ MBCs, which implies that the total number of trivially solvable branches within TS could be $O((\sqrt{2})^n)$ in the worst case.

Our Methods. We observe that the termination condition of TS, which requires the branch to be trivially solvable, may be overly strict and could lead to significant computational overhead during branching. In fact, it would be sufficient if the branch can be solvable in optimal time - referred to as being *optimally solvable*. Specifically, a problem instance with input graph G is considered optimally solvable if it can be solved in $O(scan(G) + scan(\mathcal{B}_G))$ time, where $O(scan(G))$ denotes the time required to load the graph G, $\mathcal{B}_G$ represents the set of all MBCs in G, and $O(scan(\mathcal{B}_G))$ corresponds to the time needed to output all MBCs in G. Motivated by the above observation, we propose a new BB framework termed OS, which conducts branching until it reaches optimally solvable branches. Given the fact that an optimally solvable branch can contain multiple MBCs, while a trivially solvable branch can only contain at most one MBC, it allows OS to (1) produce much less total number of branches during enumeration, making it possible to improve the worst-case time complexity theoretically and (2) reduce the computational overhead across the branching process, making it more efficient practically compared to the TS framework. To develop a practical algorithm based on the OS framework, two key questions must be addressed: (1) what types of graphs satisfy the optimally-solvable condition? (2) how should the BB process be designed to ensure it fully unleashes the power of the optimally solvable condition?

To address the first question, we observe that there exists a kind of bipartite subgraphs known as 2-biplexes [1], which allows direct construction and enumeration of all MBCs inside in nearly optimal time, without requiring further branching. The core idea is to convert the identified 2-biplex H into its complementary bipartite graph, $\overline{H}$ [2]. Since $\overline{H}$ contains only isolated vertices, simple paths, and simple cycles of even length (since its maximum degree is at most 2), each MBC in H corresponds to a maximal independent set in $\overline{H}$. These independent sets can be enumerated iteratively, ensuring that no additional computational cost is incurred during enumeration. We note that it still remains

[1] A bipartite graph H is a k-biplex if each vertex on one side has at most k non-neighbors on the other side.

[2] Given a bipartite graph $H = (L \cup R, E)$, its complementary bipartite graph $\overline{H} = (L \cup R, \overline{E})$ shares the same vertex sets as H, with an edge between $u \in L$ and $v \in R$ if and only if $(u, v) \notin E$, i.e., $\overline{E} = (L \times R) \setminus E$.

an open question whether the termination from 2-biplexes can be further relaxed to other bipartite subgraphs (e.g., k-biplexes with $k \geq 3$) that can be optimally solved.

To address the second question, we aim to integrate the new termination condition into the BB framework. An intuitive idea is to directly build upon the state-of-the-art BB algorithms. Unlike the existing studies that only terminate the recursive process when both vertex sets C and X are empty, the new design, which follows OS framework, would terminate the process as soon as the vertices in $S \cup C$ form a 2-biplex and X becomes empty. This ensures that every MBC identified within the 2-biplex induced by $S \cup C$ is also a MBC in the whole input graph G. Nevertheless, the above design may still generate much more branches than necessary. The reason is that the pivoting strategy in [9] is not well-aligned with the early-termination technique, especially when the vertices that are chosen as the pivots have only 1 or 2 non-neighbor(s) on the other side. These vertices may be possibly contained in a 2-biplex, and if a different vertex were chosen as the pivot, the termination condition could have been applied earlier to the produced branches. Motivated by this, we propose an improved pivot strategy based on a new partition-based framework to solve the above issue. The idea is to *skip* some vertices in C and choose the pivot vertex from the remaining vertices. We determine the partition in each branch B such that when no pivot can be selected in B, B is optimally-solvable and the new termination condition can be naturally applied. We present an algorithm called IPS with this new pivoting strategy. Formally, we prove that IPS would produce at most $O(\alpha^n)$ branches, where $\alpha \approx 1.3954$ is the largest positive root of $x^4 - 2x - 1 = 0$ and enumerate all MBCs in $O(m \cdot \alpha^n + n \cdot \beta)$ time under the worst case, where β represents the number of MBCs in the graph, i.e., $\beta = |\mathcal{B}_G|$. It is strictly better than that of the existing algorithms since both $O(m \cdot \alpha^n)$ and $O(n \cdot \beta)$ are bounded by $O(m \cdot (\sqrt{2})^n)$ given the fact that β is at most $(\sqrt{2})^n - 2$ for any graph with n vertices.

In addition, we adapt an existing technique, called *inclusion-exclusion* (IE), for boosting the efficiency and scalability of the BB algorithms including IPS. The idea is to construct multiple problem instances each on a smaller subgraph, which has been widely used for enumerating and finding subgraph structures [6, 38, 40].

Contributions. We summarize our contributions as follows.

- We propose a new branch-and-bound algorithm called IPS for maximal biclique enumeration problem, which is based on our newly developed termination condition and the new pivoting strategy. IPS has a strictly better worst-case time complexity than that of the state-of-the-arts, i.e., the former is $O(m \cdot \alpha^n + n \cdot \beta)$ and the latter is $O(m \cdot (\sqrt{2})^n)$. (Section 3)
- We further introduce an inclusion-exclusion framework (IE) to boost the performance of IPS. When IE is applied to IPS, the worst-case time complexity becomes $O(n \cdot \gamma^2 \cdot \alpha^\gamma + \gamma \cdot \beta)$, where γ is a parameter satisfying $\gamma \ll n$ for sparse graphs. This is better than that of IPS when the graph satisfies the condition $n \geq \gamma + 6.01 \ln \gamma$. (Section 4)
- We conduct extensive experiments on both real and synthetic datasets to evaluate the effectiveness of our proposed techniques, and the results show that our algorithm can consistently and largely outperform the state-of-the-art algorithms. (Section 5)

The rest of the paper is organized as follows. Section 2 reviews the problem and presents some preliminaries. Section 6 reviews the related work and Section 7 concludes the paper.

2 Problem and Preliminaries

Let $G = (L \cup R, E)$ be an unweighted and undirected bipartite graph, where L and R are two disjoint vertex sets and E is the edge set. Given $V_L \subseteq L$ and $V_R \subseteq R$, we use $H = G[V_L \cup V_R]$ to denote the subgraph of G induced by the vertices in $V_L \cup V_R$. For a subgraph H of a bipartite graph G, we use $L(H)$, $R(H)$ and $E(H)$ to denote the left side, right side and set of edges of H, respectively.

We note that all subgraphs considered in this paper are induced subgraphs. Given a vertex $u \in L$ and a vertex set $V_R \subseteq R$, we use $N(u, V_R)$ (resp. $\overline{N}(u, V_R)$) to denote the set of neighbors (resp. non-neighbors) of u in V_R. We define $d(u, V_R) = |N(u, V_R)|$ and $\overline{d}(u, V_R) = |\overline{N}(u, V_R)|$. Let $V_L \subseteq L$ be a set of vertices, we use $N(V_L, V_R)$ to denote the set of common neighbors of V_L in V_R, i.e., $N(V_L, V_R) = \cap_{u \in V_L} N(u, V_R)$. We have symmetric definitions for the vertices in R.

Definition 2.1 (Biclique [3]). *Given a graph $G = (L \cup R, E)$, a subgraph $H = G[V_L \cup V_R]$ of G is said to be a biclique if $\forall u \in V_L$ and $\forall v \in V_R$, there is an edge $(u, v) \in E$.*

Given a bipartite graph G, a biclique H is said to be maximal if there is no other biclique H' that contains H. Among all maximal bicliques of G, those maximal bicliques with the more edges are of more interests since they carry more useful information in fraud detection or community search tasks [28]. Therefore, in this paper, we target the maximal biclique enumeration problem but also consider some additional constraints that can be specified by the users. In specific, we consider two size constraints τ_L and τ_R such that for each returned maximal biclique H, it satisfies $|L(H)| \geq \tau_L$ and $|R(H)| \geq \tau_R$. These constraints help filter out those skewed bicliques, i.e., the number of vertices on one side is significantly larger than that on the other side, and those small-size bicliques. We formally present the problem definition as follows.

Problem 2.1 (Maximal Biclique Enumeration [5, 9]). *Given a bipartite graph $G = (L \cup R, E)$ and two integers $\tau_L \geq 1$ and $\tau_R \geq 1$, the maximal biclique enumeration problem aims to return a set of maximal bicliques such that each maximal biclique H inside satisfies $|L(H)| \geq \tau_L$ and $|R(H)| \geq \tau_R$.*

3 The Branch-and-Bound Algorithms

3.1 Overview of the Branch-and-bound Algorithms

Many algorithms have been proposed for maximal biclique enumeration problem in the literature [2, 5, 7, 9]. Most of them [2, 5, 9] adopt a conventional and widely-used branching strategy that we call Bron-Kerbosch (BK) branching [4]. Specifically, it recursively partitions the search space (i.e., the set of all maximal bicliques in G) into several sub-spaces via branching until some stopping criterion are satisfied, e.g., C and X become empty sets [6, 9]. Each search space, which corresponds to a branch B, can be represented as a triplet $B = (S, C, X)$, where

- **Set** S induces a partial biclique found so far;
- **Set** C contains all candidate vertices that can be included to S within the search space for expanding the biclique; and
- **Set** X contains all exclusion vertices that must be excluded from S within the search space.

We further denote the left side and right side of S (resp. C and X) by S_L (resp. C_L and X_L) and S_R (resp. C_R and X_R), respectively.

The recursive process starts from the initial branch $B = (S, C, X)$ with $S = \emptyset$, $C = L \cup R$ and $X = \emptyset$. Consider the branching step at a branch $B = (S, C, X)$. Let $\langle v_1, v_2, \cdots, v_{|C|} \rangle$ be an ordering of the vertices in C. Then the branching step would create $|C|$ sub-branches, where the i-th branch, denoted by B_i, includes v_i to S and excludes $v_1, v_2, \cdots v_{i-1}$ from S. Formally, for $1 \leq i \leq |C|$, the sub-branch B_i can be represented by

$$S_i = S \cup \{v_i\} \quad C_i = C \setminus \{v_1, \cdots, v_i\} \quad X_i = X \cup \{v_1, \cdots, v_{i-1}\} \tag{1}$$

We note that C_i and X_i can be further refined by removing those vertices that are not v_i's neighbors on its opposite side since they cannot form bicliques with v_i. The above branching step corresponds to a binary search process, where one searches those maximal bicliques including v_i by adding v_i from C to S and the other searches those maximal bicliques excluding v_i by removing v_i from C to X. Consequently, the total number of branches can grow to $O(2^n)$ in the worst case for a graph G

Algorithm 1: The general framework: BasicBB

Input: A bipartite graph $G = (L \cup R, E)$, τ_L and τ_R.
Output: All maximal bicliques in G that satisfy the size constraints.

```
1  BB_Rec(∅, L ∪ R, ∅);
2  Procedure BB_Rec(S, C, X)
       /* Termination                                              */
3      if C and X satisfy any stopping criterion then
4          Output all maximal bicliques in G[S ∪ C]; return;
       /* Pruning                                                  */
5      if any of pruning rule can be applied then
6          return;
       /* Branching with Pivoting                                  */
7      Choose a pivot vertex v* from C ∪ X;
8      Create a set of branches B_i based on the pivot vertex v*;
9      for each created branch B_i do
10         BB_Rec(S_i, C_i, X_i);
```

with n vertices. To reduce the number of explored branches during the search process, existing branch-and-bound algorithms for maximal biclique enumeration problem adopt different design strategies, which mainly differ in the following aspects.

- **Stopping Criterion** terminates the recursive procedure and outputs all maximal bicliques within the branch (if any), i.e., those within the subgraph induced by $G[S \cup C]$;
- **Pivoting Strategy** selects a vertex from one side of $C \cup X$ as the pivot and produces sub-branches only on the pivot itself and its non-neighbors on the other side, which ensures that all maximal bicliques are enumerated exactly once;
- **Pruning Technique** eliminates the unpromising branch, i.e., the branch not containing any maximal bicliques, and all sub-branches rooted at B.

Specifically, state-of-the-art algorithms [9] follow the BasicBB framework. In particular, they terminate the recursive procedure when both C and X become empty and output $G[S]$ as a maximal biclique. Moreover, for a branch $B = (S, C, X)$ during recursion, the pivot v^* is selected from $C_L \cup X_L$ (or $C_R \cup X_R$) as the vertex with the minimum number of non-neighbors in C_R (or C_L), so as to produce as few sub-branches as possible. By adopting these techniques, the algorithms achieve a worst-case time complexity of $O(m \cdot (\sqrt{2})^n)$, where m and n denote the number of edges and vertices of the graph, respectively [9]. This worst case occurs when the pivot selected in every branch B has only one non-neighbor. Figure 1 illustrates such an example. Notably, the example graph contains exactly $(\sqrt{2})^n - 2$ maximal bicliques, indicating that the existing bound of the time complexity under the basic framework is nearly worst-case optimal and it is difficult to further improve it.

In this paper, we propose new techniques for maximal biclique enumeration, including a new stopping criterion and a new pivoting strategy. Armed with a new partition-based branch-and-bound framework, these two techniques collectively contribute to significant theoretical advancements. Specifically, we first elaborate on how the partition-based branch-and-bound framework operates and how our proposed techniques would be seamlessly integrated into the framework (Section 3.2). We further summarize all possible pruning techniques to the problem setting of enumerating size-constrained maximal bicliques. Finally, we present that an algorithm, incorporating these

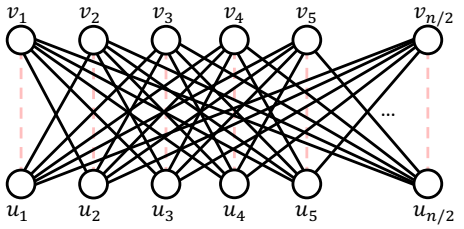


Fig. 1. An example graph with n vertices (n is even), in which each u_i is disconnected only from v_i for $i = 1, 2, \ldots, n/2$, as indicated by the dashed edges. In total, the graph contains exactly $(\sqrt{2})^n - 2$ maximal bicliques. To see this, observe that any non-empty subset of vertices chosen from U uniquely determines a maximal biclique by taking its common neighbors in V. Since $|U| = |V| = n/2$, the number of such choices is $2^{n/2} - 2$ (equiv. $(\sqrt{2})^n - 2$), excluding the empty set and the full set so that both sides of the biclique are non-empty.

techniques, achieves a worst-case time complexity of $O(m \cdot \alpha^n + n \cdot \beta)$, where $\alpha(\approx 1.3954)$ is the largest positive root of $x^4 - 2x - 1 = 0$ and β is the number of maximal bicliques in the graph. This result outperforms existing state-of-the-art branch-and-bound-based algorithms (Section 3.3).

3.2 A Partition-based Framework: ParBB

3.2.1 Overview. In this section, we introduce a new branch-and-bound algorithm called ParBB. Different from BasicBB, given a branch $B = (S, C, X)$, ParBB partitions the candidate set C into two disjoint sets C' and $C \setminus C'$ where $C' \subsetneq C$. The key advantage of this partitioning lies in the ability to defer the computation of maximal bicliques involving only vertices in C'. Specifically, instead of processing all vertices in one step, ParBB allows the enumeration of maximal bicliques in C' at a later time, in a batch-based manner. This flexibility, enabled by partitioning, ensures that the maximal bicliques in C' are handled efficiently without unnecessary recomputation. By contrast, non-partition-based frameworks, like BasicBB, do not provide such deferred handling and must process all candidates immediately. We also emphasize that this partition-based framework generalizes BasicBB. If C' is set to be empty at every branch, ParBB reduces to BasicBB. However, the partitioning strategy provides significant benefits by allowing deferred, batch-based enumeration and pruning, leading to a more efficient handling of maximal bicliques compared to non-partition-based frameworks, which must process each biclique individually. In the following, we explore two partitioning strategies that reduce computational overhead by enabling batch-based output of maximal bicliques and batch-based pruning of non-maximal ones. We summarize the procedure following the partition-based framework in Algorithm 2.

3.2.2 Candidate Set Partition Strategies. The intuition behind our partition strategies stems from a key non-trivial observation, which we summarize in the following lemma.

Lemma 3.1. *Given a bipartite graph $G = (L \cup R, E)$ being a 2-biplex, all maximal bicliques in G can be enumerated in $O(m + n \cdot \beta)$ time, where $m = |E|$, $n = |L| + |R|$ and β is the exact number of maximal bicliques in G, respectively.*

Proof. To maintain coherence, we omit the detailed proof here. The idea is that each connected component of the complementary graph $\overline{G}$ of a 2-biplex G is either a single vertex, a simple path, or a simple cycle. This follows naturally, as the maximum degree of $\overline{G}$ is at most 2. Consequently, each maximal biclique in G corresponds to a maximal independent set in $\overline{G}$, which can be constructed in a batch-based manner by merging maximal independent sets from the single vertices, simple paths, and simple cycles. □

Algorithm 2: The partition-based framework: ParBB

Input: A bipartite graph $G = (L \cup R, E)$, τ_L and τ_R.
Output: All maximal bicliques in G that satisfy the size constraints.

```
1  ParBB_Rec(∅, L ∪ R, ∅);
2  Procedure ParBB_Rec(S, C, X)
       /* Termination                                                    */
3      if C and X satisfy any stopping criterion then
4          Output all maximal bicliques in G[S ∪ C]; return;
       /* Pruning                                                        */
5      if any of pruning rules can be applied then
6          return;
       /* Partition-based Branching with Pivoting                        */
7      Partition C into two sets C' and C \ C', where C' ⊊ C;
8      Choose a pivot vertex v* from C ∪ X;
9      Create a set of branches B_i based on the pivot vertex v* and the partition;
10     for each created branch B_i do
11         ParBB_Rec(S_i, C_i, X_i);
```

This lemma tells us that there exists such a group of graphs that all maximal bicliques inside can be enumerated in nearly optimal time since it needs at least $O(n \cdot \beta)$ to output all maximal bicliques in G and the algorithm only takes extra $O(m)$ time. To equip this lemma into our algorithm, we propose a new stopping criterion shown as follows.

Stopping Criterion. If a branch $B = (S, C, X)$ satisfies $G[S \cup C]$ induces a 2-biplex and $X = \emptyset$, we output all maximal bicliques in $G[S \cup C]$ and return.

Consider an arbitrary branch $B = (S, C, X)$ that does not satisfy the stopping criterion. It must fall into at least one of the following two cases: (1) there exist vertices in C_L with at least 3 non-neighbors in C_R (or symmetrically, vertices in C_R with at least 3 non-neighbors in C_L), or (2) there exist vertices in X. Therefore, to minimize the number of produced branches, it is essential to apply the stopping criterion as early as possible, which aims to exhaustively eliminate vertices belonging to these cases as quickly as possible. To achieve this, we divide all non-terminable branches into two categories based on whether X is empty, and we propose a candidate partition strategy and enumeration procedure for each category as follows.

Category 1 - X is empty. When X is empty, each maximal biclique in the subgraph $G[S \cup C]$ is also a maximal biclique of G. We partition the candidate set into two disjoint sets such that C' always induces a 2-biplex. Formally, for each $v \in C_L$, we collect the number of disconnections with the vertices in C_R, i.e., $\overline{d}(v, C_R)$. Symmetrically, for each $v \in C_R$, we collect the number of disconnections with the vertices in C_L, i.e., $\overline{d}(v, C_L)$. We define the partition $C' = C'_L \cup C'_R$ as follows, where

$$\begin{aligned} C'_L &= \{v \in C_L \mid \overline{d}(v, C_R) \leq 2\} \\ C'_R &= \{v \in C_R \mid \overline{d}(v, C_L) \leq 2\} \end{aligned} \tag{2}$$

When C'_L and C'_R are not empty, it can be easily verified the graph induced by C' is a 2-biplex since C'_R is a subset of C_R, for a vertex $v \in C'_L$, we have $\overline{d}(v, C'_R) \leq \overline{d}(v, C_R) \leq 2$. Similarly, since C'_L is a subset of C_L, for a vertex $v \in C'_R$, we have $\overline{d}(v, C'_L) \leq \overline{d}(v, C_L) \leq 2$.

Recall that (1) our goal is to reduce the presence of vertices with at least 3 non-neighbors in C_L and C_R (since X is empty in this category), i.e., those vertices in $C_L \setminus C'_L$ and $C_R \setminus C'_R$, and (2) the pivoting strategy in [9] says that the branching steps conducted on the pivot vertex v^* and its non-neighbors are already enough to enumeration all maximal biclique within a branch, We have two choices. One is to choose a vertex $v^* \in C_L \setminus C'_L$ as the pivot vertex. The other is to choose a vertex v^* that has some non-neighbors in $C_L \setminus C'_L$ as the pivot vertex. Symmetrically, we can also choose a pivot vertex from C_R. Following this direction, we adapt the existing pivoting strategy [9] to our setting and summarize our pivoting strategy (when $X = \emptyset$) as follows. Let $B = (S, C, X)$ be a branch. Given a partition C' and $C \setminus C'$ according to Eq. (2), a vertex $v \in C$ is a candidate pivot if any of the following is satisfied: (1) $v \in C \setminus C'$; or (2) $v \in C'_L$ (resp. C'_R) has at least one non-neighbor $u \in C_R$ (resp. C_L) in $C_R \setminus C'_R$ (resp. $C_L \setminus C'_L$). The pivot v^* is selected with the minimum value of $\overline{d}(v^*, C_R)$ (if $v^* \in C_L$) or $\overline{d}(v^*, C_L)$ (if $v^* \in C_R$) among all candidate pivots. If there exists multiple vertices that have the same minimum value, we select the pivot arbitrarily. Once the pivot vertices v^* is selected, the branching steps will be conducted on v^* and its non-neighbors on v^*'s opposite side. The correctness of the pivoting strategy can be easily verified [9].

Category 2 - X is not empty. When X is non-empty, we observe that X contains the vertices that have been explored in earlier branches and are typically used to check maximality. Therefore, we leverage the number of connections and/or disconnections between the vertices in C and X to determine our partition. We consider the vertices in one side of C and those in the other side of X. Specifically, for each vertex $v \in C_L$, we collect the number of disconnections with the vertices in X_R, i.e., $\overline{d}(v, X_R)$. Symmetrically, we collect $\overline{d}(v, X_L)$ for each vertex $v \in C_R$. Besides, for each $v \in C$, we also collect the number of disconnections with the vertices on v's opposite side. Based on these values, we define the partition $C' = C'_L \cup C'_R$ as follows, where

$$\begin{aligned} C'_L &= \{v \in C_L \mid \overline{d}(v, X_R) = 0 \wedge \overline{d}(v, C_R) \leq 2\} \\ C'_R &= \{v \in C_R \mid \overline{d}(v, X_L) = 0 \wedge \overline{d}(v, C_L) \leq 2\} \end{aligned} \tag{3}$$

Recall that our goal is to eliminate the presence of vertices in X and the vertices in C having at least 3 non-neighbors. The above definition seeks to identify a subset of vertices C' from C such that the branching steps produced by including the vertex $v \in C'$ will not reduce the presence of those vertices. To demonstrate this, assume without loss of generality that $v \in C'_L$. When $X_R \neq \emptyset$, C'_L consists of vertices that are fully connected to X_R, i.e., $N(v, X_R) = X_R$. In the case where $X_R = \emptyset$, we observe that (1) the size of X_R can no longer be reduced, and (2) $\overline{d}(v, \emptyset) = 0$ since there are no vertices to disconnect from. Thus, based on this partition, we have three options for selecting a pivot vertex. First, we can choose a vertex v^* directly from X_L. Second, we can select v^* from $C_L \setminus C'_L$, as these vertices have disconnections with some members of X_R. Third, we can pick v^* from C'_L if it has non-neighbors in $C_R \setminus C'_R$. Symmetrically, a vertex can also be chosen from $C_R \cup X_R$. We summarize our pivoting strategy (when $X \neq \emptyset$) as follows. Let $B = (S, C, X)$ be a branch. Given a partition C' and $C \setminus C'$ according to Eq. (3), a vertex $v \in C \cup X$ is a candidate pivot if any of the following is satisfied: (1) $v \in (C \setminus C') \cup X$; or (2) $v \in C'_L$ (resp. C'_R) has at least one non-neighbor in $C_R \setminus C'_R$ (resp. $C_L \setminus C'_L$). The pivot v^* is selected with the minimum value of $\overline{d}(v^*, C_R)$ (if $v^* \in C_L \cup X_L$) or $\overline{d}(v^*, C_L)$ (if $v^* \in C_R \cup X_R$) among all candidate pivots. If there exists multiple vertices that have the same minimum value, we select the pivot from X if possible; otherwise, we select one from C. Similarly, once the pivot vertices v^* is selected, the branching steps will be conducted on v^* (if $v^* \in C$) and its non-neighbors in C on v^*'s opposite side. The correctness of the pivoting strategy can be easily verified [9].

3.3 Implementation Details and Analysis

Before we introduce the implementation details of the algorithm, we first elaborate on the possible pruning rules that can be applied as follows.

Pruning Rules. Let $B = (S, C, X)$ be a branch. Define two variables $\theta_L = |S_L| + |C_L|$ and $\theta_R = |S_R| + |C_R|$. We can prune the branch B if any of the following is satisfied:

(P1) $\theta_L < \tau_L$ or $\theta_R < \tau_R$;

(P2) There exists a vertex $v \in X_L$ (resp. X_R) such that v connects with all vertices in C_R (resp. C_L).

The rationale of the first rule is that θ_L and θ_R can be verified to be the upper bound of the number of vertices at the left side and that at the right side of a maximal biclique covered by the branch B, respectively. If either of them does not comply with the size constraint, the maximal biclique enumerated from this branch would violate the problem definition. The rationale of the second rule is that all maximal bicliques in subgraph $G[S \cup C]$ would not be globally maximal, since an additional vertex v can be included in each of them. In addition, we observe that the partition strategies and the pivoting strategies can be naturally combined when $X = \emptyset$ and $X \neq \emptyset$. We summarize as follows.

Partition-based Pivoting Strategy. Let $B = (S, C, X)$ be an arbitrary branch during the recursion. We partition C into two disjoint sets $C' = C'_L \cup C'_R$ and $C \setminus C'$ by setting

$$\begin{aligned} C'_L &= \{v \in C_L \mid \overline{d}(v, X_R) = 0 \wedge \overline{d}(v, C_R) \leq 2\} \\ C'_R &= \{v \in C_R \mid \overline{d}(v, X_L) = 0 \wedge \overline{d}(v, C_L) \leq 2\}. \end{aligned} \tag{4}$$

A vertex v is a candidate pivot if any of the following is satisfied:

(C1) $v \in (C \setminus C') \cup X$;

(C2) $v \in C'_L$ (resp. C'_R) and it has at least one non-neighbor in $C_R \setminus C'_R$ (resp. $C_L \setminus C'_L$).

The pivot v^* is selected among all candidates with the minimum number of non-neighbors in C. We first select the pivot from X if possible; otherwise, we select one from C.

Based on the all possible techniques, including (1) the partition-based framework, (2) the candidate set partition strategies, (3) the pivoting strategy and (4) pruning techniques, we develop a new branch-and-bound algorithm called `IPS`. The pseudocode is presented in Algorithm 3.

Worst-case Time Complexity Analysis. The overall time complexity can be expressed as $O\big(P(n) \cdot T(n)\big)$, where $P(n)$ denotes the running time per recursive call and $T(n)$ denotes the total number of recursive calls. For the former, given a branch $B = (S, C, X)$, we note that this new partition-based pivoting strategy will not incur more overhead than that of the existing pivoting strategy, i.e., the partition-based pivoting strategy can also select a pivot v^* in $O(m)$ time. To see this, our pivot selection procedure can be decomposed into three steps. The first step is to calculate necessary values for partition, i.e., $\overline{d}(\cdot, \cdot)$. The second step is to collect the candidate pivots based on the partition. The last step is to choose the pivot among candidates. Consider the first step. We take a vertex $v \in C_L \cup X_L$ as an example. To calculate $\overline{d}(v, X_R)$ (resp. $\overline{d}(v, C_R)$), we first initialize $\overline{d}(v, X_R)$ (resp. $\overline{d}(v, C_R)$) to be $|X_R|$ (resp. $|C_R|$). Then we substract one from $\overline{d}(v, X_R)$ (resp. $\overline{d}(v, C_R)$) if there exists an edge (v, u) between v and a vertex $u \in X_R$ (resp. C_R), which can be done in $O(m)$. Symmetric procedure can be done for a vertex $v \in C_R \cup X_R$. Consider the second step. It can be done in $O(|C|)$, which is bounded by $O(m)$. Consider the third step. The critical point is to calculate a vertex v's non-neighbors. We only collect the non-neighbor information for the vertices that have

Algorithm 3: Pivot-based branch-and-bound algorithm with improved candidate set partition strategy: IPS

Input: A bipartite graph $G = (L \cup R, E)$, τ_L and τ_R.
Output: All maximal bicliques in G that satisfy the size constraints.

```
1  ParBB_Rec(∅, L ∪ R, ∅);
2  Procedure ParBB_Rec(S, C, X)
       /* Termination                                                    */
3      if G[S ∪ C] induces a 2-biplex and X = ∅ then
4          Output all maximal bicliques in G[S ∪ C];
5          return;
       /* Pruning                                                        */
6      if any of pruning rules is satisfied then
7          return;
       /* Partition-based Branching with Pivoting                        */
8      v* ← the pivot selected by Partition-based Pivoting Strategy;
9      Create branches B_i on v* (if v* ∈ C) and its non-neighbors on v*'s opposite side;
10     for each created branch B_i do
11         ParBB_Rec(S_i, C_i, X_i);
```

at most two non-neighbors, which can be done in $O(m + 2 \cdot |C|)$. Thus, $P(n)$ is bounded by $O(m)$. For the latter, we present the detailed analysis of $T(n)$ and the resulting time complexity in the following theorem, which shows that the worst-case bound of IPS is strictly better than that of existing algorithms.

Theorem 3.2. *Given a bipartite graph $G = (L \cup R, E)$ with $n = |L| + |R|$ and $m = |E|$, IPS enumerates all maximal bicliques in G in $O(m \cdot \alpha^n + n \cdot \beta)$ time, where α (≈ 1.3954) is the largest positive real root of $x^4 - 2x - 1 = 0$.*

Proof. Let $B = (S, C, X)$ be a branch and let $n = |C| + |X|$. The running time of ParBB_Rec is dominated by the size of its recursion tree, i.e., the total number of generated branches. At each branch, the pivot v^* is selected and the algorithm creates sub-branches on v^* (if $v^* \in C$) and on its k non-neighbors on the opposite side, where $k = \overline{d}(v^*, C)$. Let $T(n_0)$ denote the number of recursive calls involving v^* (when applicable), and let $T(n_1), \cdots, T(n_k)$ denote the number of recursive calls involving each non-neighbor of v^*, respectively. Accordingly, the number of recursive calls satisfies

$$T(n) \leq \begin{cases} \sum_{i=1}^{k} T(n_i) & \text{if } v^* \in X, \\ T(n_0) + \sum_{i=1}^{k} T(n_i) & \text{if } v^* \in C, \end{cases} \tag{5}$$

where each n_i denotes the size of the remaining instance in the corresponding sub-branch, determined by the number of vertices removed from $C \cup X$. Each branching step yields a linear recurrence, and the exponential growth rate of its solution is determined by the largest real root of the corresponding characteristic equation[3].

The analysis systematically characterizes all possible branching cases. Without loss of generality, assume the pivot v^* is selected from the left side, i.e., $v^* \in C_L \cup X_L$. Depending on the specific

[3]For a linear recurrence $T(n) \leq \sum_i c_i \cdot T(n - a_i)$, one assumes a solution of the form $T(n) = x^n$, which leads to the characteristic equation $x^n - \sum_i c_i \cdot x^{n-a_i} = 0$. The largest real root of this equation determines the asymptotic growth rate of $T(n)$ [13].

position where v^* is selected, each branch falls into one of the following three categories. For each category, we derive the recurrence by tracking the reduction of $C \cup X$ in each sub-branch and upper bound the total number of recursive calls accordingly.

Case (1) - $v^* \in X_L$. If $k = 0$, the branch is pruned by Rule (P2); hence $k \geq 1$. The algorithm creates k sub-branches on v^*'s non-neighbors in C_R. It can be verified that each sub-branch removes v^*, the chosen non-neighbor, and at least k additional vertices on the opposite side, giving a size decrease of at least $k + 2$. Therefore,

$$T(n) \leq k \cdot T\big(n - (k+2)\big) \qquad (k \geq 1). \tag{6}$$

This recurrence implies $T(n) = O(n)$ when $k = 1$, and an exponential bound with base $\alpha_1 = \max_{k\geq 2} k^{1/(k+2)} < 1.26$ when $k \geq 2$.

Case (2) - $v^* \in C_L \setminus C'_L$. Given $v^* \notin C'_L$, either $\overline{d}(v^*, X_R) \neq 0$ or $\overline{d}(v^*, X_R) = 0 \wedge \overline{d}(v^*, C_R) > 2$ holds. We consider both cases.

Case (2.1) - $\overline{d}(v^*, X_R) \neq 0$. It follows that (i) v^* has at least one non-neighbor in X_R and (ii) all non-neighbors of v^* in C_R are candidate pivots. The algorithm branches on v^* and on its k non-neighbors in C_R. The branching step on v^* removes v^*, all k non-neighbors in C_R, and one excluded non-neighbor in X_R, resulting in a decrease of at least $k + 2$. Each branching step on a non-neighbor of v^* removes the chosen vertex and, due to the pivot selection rule, enforces the removal of at least k additional vertices on the opposite side, resulting in a decrease of at least $k + 1$. Thus,

$$T(n) \leq T\big(n - (k+2)\big) + k \cdot T\big(n - (k+1)\big) \qquad (k \geq 0). \tag{7}$$

The worst case occurs at $k = 2$, giving the exponential bound with base $\alpha_2 \approx 1.3954$, where α_2 is the largest real root of $x^4 - 2x - 1 = 0$.

Case (2.2) - $\overline{d}(v^*, X_R) = 0$. It follows that (i) $k \geq 3$ and (ii) all non-neighbors of v^* in C_R are candidate pivots. The algorithm creates $k + 1$ sub-branches on v^* and its k non-neighbors in C_R. By the pivot selection rule, branching on either v^* or its non-neighbors removes at least $k + 1$ vertices from $C \cup X$, yielding a decrease of at least $k + 1$ in each sub-branch. This gives the coarse bound

$$T(n) \leq (k+1) \cdot T\big(n - (k+1)\big) \qquad (k \geq 3). \tag{8}$$

The worst case occurs at $k = 3$. To achieve a tighter bound, a finer-grained analysis at $k = 3$ leads to the following recurrence

$$T(n) \leq 2 \cdot T(n-4) + 2 \cdot T(n-6) + 2 \cdot T(n-7), \tag{9}$$

whose branching factor is strictly smaller than α_2. For $k \geq 4$, the coarse bound applies, with exponential base $\max_{k\geq 4}(k+1)^{1/(k+1)} < 1.38 < \alpha_2$. We put the detailed analysis for $k = 3$ as follows.

The algorithm produces four sub-branches, one on v^* and one on each of its non-neighbors u_1, u_2 and u_3, denoted by B_0, B_1, B_2 and B_3, respectively. Consider $B_i = (S_i, C_i, X_i)$ produced on u_i for $1 \leq i \leq 3$. It can be verified that $u_i \in C_R \setminus C'_R$. Let n_1, n_2 and n_3 be the size of $|C_i| + |X_i|$ for $1 \leq i \leq 3$, respectively. We first have $n_i = |C_i| + |X_i| \leq |C| + |X| - 4 = n - 4$. We skip the analysis for B_1 since n_1 cannot achieve a tighter bound. Consider the branch $B_2 = (S_2, C_2, X_2)$ produced on u_2. Note that $u_1 \in X_2$ and $v^* \notin C_2 \cup X_2$. Since $\overline{d}(u_1, C_L) = 3$ under the worst case, we have $\overline{d}(u_1, C_{2L}) = 2$, where C_{2L} is the left side of C_2. Then consider the pivot selected in the branch B_2, which we denote by v_2^*. Either of the following case holds: (i) if $v_2^* \in X_2$, then $\overline{d}(v_2^*, C_2) \leq 2$ or (ii) if $v_2^* \in C_2$, then $\overline{d}(v_2^*, C_2) \leq 1$; since otherwise u_1 will be the pivot.

For the former case, two sub-branches would be produced from B_2 under the worst case, i.e., those produced upon v_2^*'s two non-neighbors in C_2. We have

$$T(n_2) \leq 2 \cdot T(n_2 - 4) \leq 2 \cdot T(n - 8) \tag{10}$$

For the latter case, two sub-branches would be produced from B_3 under the worst case, i.e., one produced upon v_2^* and the other produced upon v_2^*'s non-neighbor in C_2. We have

$$T(n_2) \leq T(n_2 - 2) + T(n_2 - 3) \leq T(n - 6) + T(n - 7) \tag{11}$$

It is easy to see that the worst case would happen in the latter one. Similar to the analysis of B_2, we can have the same recurrence for the sub-branch B_3. Thus, we omit the details of the recurrence for $T(n_3)$. To sum up, we have

$$T(n) \leq 2 \cdot T(n - 4) + 2 \cdot T(n - 6) + 2 \cdot T(n - 7) \tag{12}$$

Case (3) - $v^* \in C'_L$. It follows $k \in \{1, 2\}$. We analyze both cases.

Case (3.1) - $k = 1$. The algorithm produces two sub-branches, one on v^* and the other on its non-neighbor u_1, denoted by B_0 and B_1, respectively. For B_0, the branching step removes v^* and u_1, and we have $|C_0| + |X_0| \leq (|C| - 2) + |X| = n - 2$. For B_1, since $u_1 \in C_R \setminus C'_R$ by Rule (C2), either $\overline{d}(u_1, C_L) > 2$ or $\overline{d}(u_1, X_L) > 0$ holds, and in both cases, $|C_1| + |X_1| \leq |C| + |X| - 3 = n - 3$. Therefore,

$$T(n) \leq T(n - 2) + T(n - 3). \tag{13}$$

Case (3.2) - $k = 2$. The algorithm produces three sub-branches, one on v^* and one on each of its non-neighbors u_1 and u_2, denoted by B_0, B_1, and B_2, respectively. By Rule (C2), we assume $u_2 \in C_R \setminus C'_R$. Let $l = \overline{d}(u_1, C_L) \geq 1$, since u_1 is non-adjacent to at least one vertex, i.e., v^*. We distinguish the following cases based on the value of l.

Case (3.2.1) - $l = 1$. We claim that no maximal biclique exists in B_2 since (i) $u_1 \in X_2$ and (ii) for any biclique found in B_2, the vertex u_1 can always be added to form a strictly larger biclique, contradicting maximality. Therefore, the sub-branch B_2 can be safely pruned. When $k = 2$ and $l = 1$, we have $|C_0| + |X_0| \leq (|C| - 3) + |X| = n - 3$ and $|C_1| + |X_1| \leq (|C| - 2) + |X| = n - 2$. Hence,

$$T(n) \leq T(n - 2) + T(n - 3). \tag{14}$$

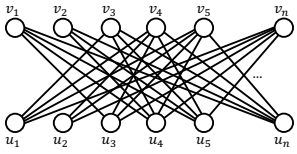
Case (3.2.2) - $l \geq 2$. Consider B_0. Given $k = 2$, we have $|C_0| + |X_0| \leq (|C| - 3) + |X| = n - 3$. Consider B_1. Given $l \geq 2$, we have $|C_1| + |X_1| \leq (|C| - 3) + |X| = n - 3$. Consider B_2. u_2 is a candidate pivot since $u_2 \in C_R \setminus C'_R$. Thus, either (i) $\overline{d}(u_2, C_L) > 2$ or (ii) $\overline{d}(u_2, C_L) = 2 \wedge \overline{d}(u_2, X_L) > 0$ holds. Then $|C_2| + |X_2| \leq |C| + |X| - 4 = n - 4$. Formally, we have

$$T(n) \leq 2 \cdot T(n - 3) + T(n - 4) \tag{15}$$

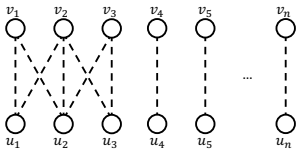
By solving all recurrences, the worst case is attained in Case (3.2.2), whose branching factor α_3 is the same as that of Case (2.1).

Conclusion. Let $\alpha = \max\{\alpha_1, \alpha_2, \alpha_3\}$. Then the total number of recursive calls is bounded by $O(\alpha^n)$. Since each call can be done in $O(m)$ time, all maximal bicliques can be enumerated in $O(m \cdot \alpha^n + n \cdot \beta)$ time, where β is the number of maximal bicliques. □

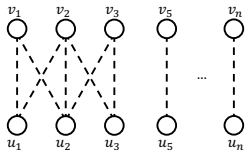
Remarks. First, compared with existing algorithms that target the maximal biclique enumeration problem [2, 5, 7, 9], our algorithm offers several key advantages. Theoretically, while existing branch-and-bound algorithms [2, 7, 9] all exhibit a worst-case time complexity of $O(m \cdot (\sqrt{2})^n)$, our approach provides a strictly better theoretical bound since both $O(m \cdot \alpha^n)$ and $O(n \cdot \beta)$ are bounded by $O(m \cdot (\sqrt{2})^n)$. We note that β is at most $(\sqrt{2})^n - 2$ in the worst case [26], yet empirically, it is



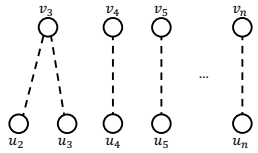
(a) A 3-biplex graph G



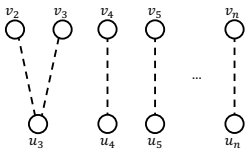
(b) G's complementary graph $\overline{G}$



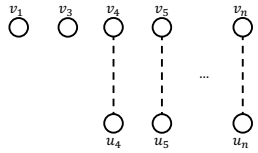
(c) The complementary of the candidate graphs after branching on u_4 and/or v_4 with the basic pivot strategy, i.e., u_4 is the pivot vertex.



(d) The complementary of the candidate graph after branching on u_1 with another pivot strategy, e.g., u_1 is the pivot vertex.



(e) The complementary of the candidate graph after branching on v_1 with another pivot strategy, e.g., u_1 is the pivot vertex.



(f) The complementary of the candidate graph after branching on v_2 with another pivot strategy, e.g., u_1 is the pivot vertex.

Fig. 2. An illustration of the reason why the conventional pivoting strategy with the new stopping criterion would still produce much more branches than necessary.

much smaller. Practically, our algorithm outperforms all existing algorithms, as confirmed by the experimental results.

We emphasize that the termination condition and the pivoting strategy *jointly* contribute to the theoretical improvement in the worst-case performance. Although each technique independently contributes to the overall empirical performance, neither is sufficient alone to achieve a substantial reduction in complexity. We also acknowledge that the excellent empirical performance of state-of-the-art algorithms and ours on real-world graphs primarily arises from the structural properties of such datasets (e.g., sparsity), rather than from the worst-case behavior. However, measuring such average-case complexity is challenging and even infeasible in many instances, which is a common issue in the field. Thus, identifying algorithmic bottlenecks and providing a solid foundation for future improvements is still essential.

Second, recall that in Section 1, we can directly build upon the existing BB algorithm [9] by just integrating the our new stopping criterion with the existing BB algorithms and retaining the conventional pivoting strategy. However, the conventional pivoting strategy is not well-aligned with the new stopping criterion and would still generate much more branches than necessary, i.e., $O((\sqrt{2})^n)$ branches. Therefore, this design still has the same worst-case time complexity as that of [9]. We illustrate this with the following example.

Example 3.3. Consider a bipartite graph G in Figure 2(a) whose complementary graph $\overline{G}$ is shown in Figure 2(b). We cannot directly apply the termination technique to G since G is a 3-biplex (i.e., u_2 and v_2 have 3 non-neighbors on their opposite side, respectively). Therefore, the conventional pivoting strategy would choose a vertex with the minimum number of non-neighbors as the pivot, e.g., u_4 is selected as the pivot vertex. Branching steps would be conducted on u_4 and its non-neighbor v_4. We show the complementary graph of the candidate graphs after branching

on u_4 and/or v_4 in Figure 2(c). We note that these two candidate graphs are identical. We still cannot apply the early-termination technique on either of them. The procedure would continue recursively, choosing $u_5, u_6, \cdots, u_n$ as pivots, until the candidate graph only has u_1, u_2, u_3, v_1, v_2 and v_3 inside. Then we can apply the early-termination technique on the candidate graphs after the branching steps on either of a vertex and its non-neighbors. Thus, the algorithm with the conventional pivoting strategy would produce $O(2^n)$ branches in this example graph. However, our proposed pivoting strategy, would create $O(1)$ branches, regardless of how large the value of n is. That is, we skip all those u_i and v_i for $i \geq 4$ and choose u_1 as the pivot vertex. Branching steps would be conducted on u_1 and its non-neighbors v_1 and v_2. We show the complementary of the candidate graphs after branching on u_1, v_1 and v_2 in Figure 2(d), Figure 2(e) and Figure 2(f), respectively. Since all these three branches are 2-biplexes, we can directly apply the early-termination technique and enumerate the maximal bicliques inside in nearly optimal time.

Third, the delay, i.e., the maximum running time between two consecutive outputs, is a standard measure for evaluating enumeration algorithms [41, 44]. In the literature on maximal subgraph enumeration, including the problem studied in this paper, most state-of-the-art solutions adopt a backtracking (a.k.a. branch-and-bound) framework [2, 5, 36]. This framework is widely used because it runs much faster on real-world graphs and does not require expensive overheads for maintaining complex auxiliary data structures. Our algorithm follows this paradigm and therefore enjoys the same characteristics. As a result, it cannot guarantee polynomial delay in theory. In particular, the early-termination mechanism outputs maximal bicliques in batches, and the delay between two consecutive batches can be exponential in the worst case. Moreover, under non-trivial size constraints (i.e., $\tau_L > 1$ and/or $\tau_R > 1$), no existing algorithm can achieve polynomial delay in the worst case. Although polynomial-delay algorithms for maximal subgraph enumeration do exist [7, 9], such methods typically incur more overhead and are rarely competitive in practice.

Finally, our partition-based pivoting strategy can also be applied to other enumeration task, such as maximal clique enumeration. The idea of the pivot selection is similar but the difference comes from the fact that the graph is unipartite in maximal clique enumeration. Therefore, it would not be necessary to distinguish which side of a vertex and its non-neighbors. Similarly, the theoretical result can also be achieved for maximal clique enumeration. We shall leave this as a future work.

4 Efficiency Boosting Techniques

In this section, we apply a widely-used decomposition technique, namely *inclusion-exclusion based framework* [6, 8, 40] (`IE` in short) to further boosts the efficiency and scalability of the algorithms introduced in the Section 3.

Given a subgraph enumeration task on a graph G, the idea of the framework is that it first decomposes the whole graph into multiple smaller subgraphs, then the algorithm runs on each graph instance, and finally returns the answer by considering the results from each graph instance. For our problem, given a graph $G = (L \cup R, E)$, we consider an arbitrary ordering over the vertices in L, which we denote by $\langle v_1, v_2, \cdots, v_{|L|} \rangle$. Let $N_1(v_i)$ and $N_2(v_i)$ denote the sets of 1-hop and 2-hop neighbors of v_i, respectively. All maximal bicliques in G can be enumerated exactly once from each induced graph $G_i = G[L_i \cup R_i]$ for $i = 1, \ldots, |L|$, where L_i is the set of v_i's 2-hop neighbors, excluding $\{v_1, \ldots, v_{i-1}\}$, and R_i is the set of v_i's 1-hop neighbors. The correctness of the formulation can be found in [7]. With this formulation, our branch-and-bound algorithms can solve each graph instance if the sets S, C and X are initialized properly. Formally, for the graph instance G_i for $1 \leq i \leq |L|$,

$$\begin{aligned} S_i = \{v_i\} \qquad C_i = (N_1(v_i) \cup N_2(v_i)) \setminus \{v_1, \cdots, v_i\} \\ X_i = N_2(v_i) \cap \{v_1, \cdots, v_{i-1}\} \end{aligned} \tag{16}$$

We note that with this framework, all algorithms introduced in Section 3 can be improved. The reason is as follows. It can be verified that C_i and X_i are mutually exclusive, and their union is given by $(N_1(v_i) \cup N_2(v_i)) \setminus \{v_i\}$. We define a variable

$$\gamma = \max_i \big(|C_i| + |X_i|\big) = \max_{v_i \in L} \big(|N_1(v_i)| + |N_2(v_i)| - 1\big), \tag{17}$$

which represents the size of the largest instance produced from the root branch. Theoretically, γ is strictly smaller than n since $\gamma \leq \min\{n-1, \Delta^2 + \Delta - 1\}$, where Δ is the maximum degree of a vertex in G. Practically, $\gamma \ll n$ due to the sparse nature of real-world graphs, as verified in the experiments. We summarize the time complexity improvements as follows.

Theorem 4.1. *Given a bipartite graph G with n vertices and m edges. The time complexity of* IPS *with* IE*-based framework would be $O(n\gamma^2 \cdot \alpha^\gamma + \gamma \cdot \beta)$, where γ can be computed by Eq. (17).*

Remark. (1) Compared with the algorithm without inclusion-exclusion based technique, when the graph satisfies $n \geq \gamma + 6.01 \cdot \ln \gamma$ (this condition is satisfied by all existing real-world bipartite graphs due to their sparse nature), the time complexities of the algorithms with IE is strictly bounded by those of the algorithms without IE. To see this,

$$\begin{aligned} n \geq \gamma + 6.01 \cdot \ln \gamma \quad &\Rightarrow \quad n \ln \alpha \geq \gamma \ln \alpha + 6.01 \cdot \ln \gamma \ln \alpha \\ &\Rightarrow \quad n \ln \alpha > \gamma \ln \alpha + 2 \ln \gamma \\ &\Rightarrow \quad n \ln \alpha + \ln \frac{m}{n} > \gamma \ln \alpha + 2 \ln \gamma \\ &\Rightarrow \quad m \cdot \alpha^n > n \cdot \gamma^2 \cdot \alpha^\gamma \end{aligned} \tag{18}$$

The first inequality is an equivalent transformation because a positive value, $\ln \alpha$, is multiplied by both sides of the inequality. The second inequality follows from the fact that $6.01 \cdot \ln \alpha > 2$. The third inequality holds since a positive value, $\ln(m/n)$, is added to the left side of the inequality. The last inequality is also an equivalent transformation by taking the exponents on both sides of the inequality. (2) As indicated in several subgraph enumeration problems [7, 9, 35], the theoretical and practical performance of the branch-and-bound algorithm are usually sensitive to the vertex ordering. However, it depends in our case. Consider the theoretical performance. The vertex ordering would not affect our theoretical outcomes since the exponents mainly depend on $|C| + |X|$ but not solely on $|C|$ in our analysis. Therefore, those vertex ordering algorithms, which target minimizing the largest size of the candidate set after branching on the initial branch, e.g., degeneracy ordering [24] or unilateral ordering [7], will not improve the theoretical performance of our algorithm. Consider the practical performance. As verified in the experiments, we observe that running time varies as different orderings are applied. Therefore, we consider two variants of our algorithms, one using the degeneracy ordering and the other using unilateral ordering in our experiments. The details of the experimental setting will be shown in Section 5.

5 Experiments

5.1 Experimental setup

Datasets. We use 15 real datasets in our experiments, which can be obtained from http://konect.cc/. For each graph, we ignore the weights and multiple edges [4] at the very beginning. Given a graph G, in addition to the number of vertices and edges inside, we also collect the statistics and report the edge density, which is defined as $\rho = 2 \times |E|/(|L| + |R|)$, the maximum degree Δ, the number of all maximal bicliques β, and the value of γ introduced in Section 4, which are summarized in

[4]The multiple edges represent those that are duplicated themselves multiple times in a graph.

Table 1. Dataset Statistics. All datasets are reported with $|L| \leq |R|$ and are sorted based on $|E|$.

Graph	Category	$\|L\|$	$\|R\|$	$\|E\|$	ρ	β	Δ	γ
forum (FR)	Interaction	522	899	7,089	9.97	16,261	126	527
ukwiki (WK)	Authorship	779	16,699	32,615	3.73	23,772	9,254	9,734
escorts (ES)	Rating	6,625	10,107	39,044	4.66	49,538	305	1,263
youtube (YT)	Affiliation	30,087	94,238	293,360	4.71	1,826,587	7,591	14,947
github (GH)	Authorship	56,519	120,867	440,237	4.96	55,346,398	3,675	33,324
crossing (BC)	Interaction	105,278	340,523	1,149,739	5.15	54,458,953	13,601	67,516
overflow (SO)	Rating	96,678	545,195	1,301,942	4.05	3,320,824	6,119	36,412
actors (AC)	Affiliation	127,823	383,640	1,470,404	5.74	1,075,444	646	4,200
twitter (TW)	Interaction	175,214	530,418	1,890,661	5.35	5,536,526	19,805	143,233
bibsonomy (BS)	Feature	204,673	767,447	2,499,057	5.14	2,507,269	182,673	191,911
imdb (IM)	Affiliation	303,617	896,302	3,782,463	6.30	5,160,061	1,590	16,133
amazon (AZ)	Rating	1,230,915	2,146,057	5,743,258	3.40	7,551,358	12,180	115,066
dblp (DB)	Authorship	1,953,085	5,624,219	12,282,059	3.24	4,899,032	1,386	3,050
tvtropes (TV)	Feature	64,415	87,678	3,232,134	42.50	19,636,996,096	12,400	49,893
livejournal (LJ)	Affiliation	3,201,203	7,489,073	112,307,385	21.01	$> 10^7$B	1,053,676	3,695,926

Table 1. We use four representative datasets as default ones, shown in bold in Table 1, i.e., GH, BC, TW and DB, since they cover different graph scales.

Baselines. First, we introduce three variants of our algorithms. Specifically, we first consider the simple version IPS (i.e., the branch-and-bound algorithms without IE). In addition, for IPS with IE, recall that in Section 4, we consider using degeneracy ordering and unilateral ordering to branch the initial branch. Therefore, we denote IPS with degeneracy ordering and unilateral ordering by IPSD and IPSH, respectively. Second, we consider five existing algorithms, namely BCEAD [9], BCEAH [9], BCDelay [9], MBET [5] and MBETM [5]. The first two algorithms are the branch-and-bound algorithms adopting the basic pivot selection strategy, with variations in vertex ordering. The latter three baselines are the branch-and-bound algorithms that achieve polynomial delay. They differ primarily in their implementation details and the space organization.

Settings. When enumerating all maximal bicliques (i.e., $\tau_L = \tau_R = 1$), we compare our algorithms IPSD and IPSH with all baseline methods. For size-constrained maximal bicliques (i.e., $\tau_L > 1$ and/or $\tau_R > 1$), we exclude MBET and MBETM from consideration because the prefix tree structure proposed in [5] is incompatible with the pruning rules introduced in Section 3.3, i.e., size-constrained maximal bicliques can only be obtained after enumerating all maximal bicliques and filtering out the ineligible ones. All algorithms are implemented in C++, and experiments are conducted on a Linux machine with a 2.10GHz Intel CPU and 128GB memory. We set the time limit to 24 hours (i.e., 86,400 seconds). For any algorithm that exceeds this time limit, (1) the running time is recorded as "-" or "INF" and (2) the number of maximal bicliques that have already been enumerated within the 24-hour period is reported. For each algorithm, we report the average result over 5 runs. Implementation codes and datasets can be found via this link https://github.com/wangkaixin219/IPS.

5.2 Experimental results

(1) Comparison among variants (real datasets). In addition to the baselines introduced in Section 5.1, we further include several BB algorithms to verify the effectiveness of each proposed technique: (1) BCEA [9] and (2) BPS+, BPSD+ and BPSH+ correspond to the algorithms BCEA, BCEAD and BCEAH with the conventional pivoting strategy and our proposed new stopping criterion. Table 2 shows the results of the comparison among the different variants of our algorithms. We have the following observations. First, compare BPS+ with BCEA. BPS+ outperforms BCEA on all datasets,

Table 2. Comparison among variants of our algorithm (unit: seconds by default). The bold and underlined values indicate the best and the second best, respectively (similar notations are used for other tables).

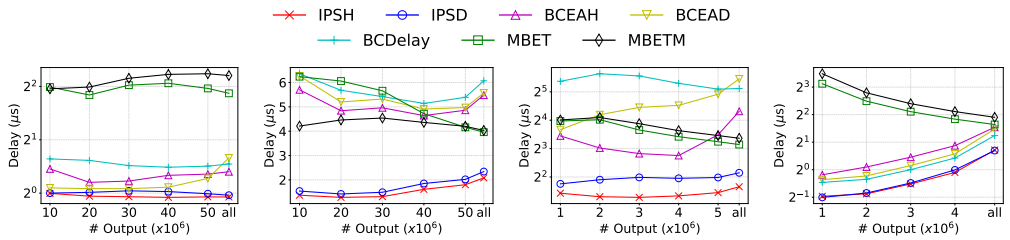
	BCEA	BPS+	IPS	BCEAD	BPSD+	IPSD	BCEAH	BPSH+	IPSH
FR	29ms	28ms	23ms	21ms	19ms	18ms	20ms	18ms	**16ms**
WK	248ms	192ms	153ms	121ms	73ms	43ms	127ms	53ms	**31ms**
ES	125ms	118ms	108ms	94ms	77ms	59ms	88ms	76ms	**53ms**
YT	17.75	6.61	5.77	7.65	2.59	**2.29**	5.92	2.59	2.31
GH	89.63	85.79	84.40	74.00	63.44	**52.34**	72.70	63.95	52.95
BC	324.19	184.52	166.71	308.08	172.06	129.52	303.33	161.61	**115.09**
SO	62.89	28.00	23.96	146.50	18.16	13.81	97.48	15.93	**12.59**
AC	8.71	8.46	8.01	4.82	3.50	2.72	4.94	3.52	**2.55**
TW	500.68	338.12	259.29	243.28	27.62	23.77	109.92	26.80	**18.47**
BS	85.79	80.88	43.62	17.60	16.12	15.09	17.89	14.61	**13.13**
IM	67.37	46.24	43.86	57.31	18.32	**17.23**	52.91	19.02	18.20
AZ	594.11	190.69	136.75	471.45	156.72	54.23	236.60	132.50	**35.08**
DB	16.00	15.72	12.38	13.62	11.37	**7.92**	14.92	11.85	8.03
TV	- (13B)	20.7h	18.2h	23.9h	9.7h	**9.2h**	23.6h	10.1h	9.7h
LJ	- (12B)	- (17B)	- (29B)	- (16B)	- (33B)	- (65B)	- (18B)	- (35B)	**- (75B)**

which shows the effectiveness of the new stopping criterion technique. The results are consistent if we compare BPSD+ and BCEAD (resp. BPSH+ and BCEAH). Second, compare IPS with BPS+. IPS runs faster than BPS+ on all datasets, which shows the effectiveness of the new pivoting strategy. We also collect the statistics of the produced branches. For example, on YT dataset, BCEA produces 8.28M branches. With the new stopping criterion technique, BPS+ reduces the number of branches to 5.17M branches. IPS further reduces the number of branches to 4.66M. This result further confirms that the new pivoting strategy is more compatible with the proposed stopping criterion compared to the conventional pivoting strategy. Together, these two techniques significantly contribute to the observed efficiency improvements. The results are also consistent if we compare IPSD and BPSD+ (resp. IPSH and BPSH+). Notably, IPSD can achieve up to an order of magnitude speedup over BCEAD on TW dataset. Third, compare IPS with IPSD (or IPSH). The latter ones alway runs faster than the former one, which verifies the effectiveness of the IE framework. Besides, IPSH and IPSD have comparable results on the majority of the datasets since they have the same worst-case time complexities, but differ with each other on some datasets such as BC and TW. The possible reasons are (1) generating degeneracy ordering and the unilateral ordering would incur different time costs, i.e., $O(m)$ for the former and $O(\Delta \cdot m)$ for the latter, respectively, and (2) the empirical time costs for enumeration mainly depends on the size of $|C|$ since the depth of the recursive tree is at most $|C|$. As indicated in [7], unilateral ordering would bring more benefits than degeneracy ordering, i.e., the maximum $|C_i|$ of a sub-branch B_i produced from the initial branch is smaller.

(2) Comparison among all baselines when enumerating all maximal bicliques (real datasets). Table 3 shows the results of the comparison among all baselines when enumerating all maximal bicliques. We observe that our proposed algorithm IPSH and IPSD can run faster than other algorithms on most datasets. Specifically, for all datasets except TV and LJ, IPSH can achieve an average 3.2x (up to 7.7x) speedup and 1.7x (up to 3.8x) speedup over BCEAH and MBET, respectively. This is consistent with our theoretical analysis that the worst-case running time of IPSH and IPSD is smaller than that of other algorithms. Moreover, on the hard datasets (i.e., TV and LJ), our algorithm can largely outperform the existing algorithms by enumerating much more

Table 3. Comparison among all baselines when enumerating all maximal bicliques (unit: seconds by default). "×" indicates that the algorithm is terminated due to the "Out Of Memory".

	IPSH	IPSD	BCEAH	BCEAD	BCDelay	MBET	MBETM
FR	**16ms**	18ms	20ms	21ms	26ms	19ms	19ms
WK	**31ms**	43	127ms	121ms	170ms	62ms	66ms
ES	**53ms**	59ms	88ms	94ms	98ms	72ms	79ms
YT	2.31	**2.29**	5.92	7.65	8.47	5.06	5.24
GH	52.95	**52.34**	72.70	74.00	81.29	202.49	255.01
BC	**115.09**	129.52	303.33	308.08	335.72	219.32	222.84
SO	**12.59**	13.81	97.48	146.50	239.18	12.74	16.90
AC	**2.55**	2.72	4.94	4.82	5.94	3.33	3.30
TW	**18.47**	23.77	109.92	243.28	192.37	48.86	57.11
BS	13.13	15.09	17.89	17.60	22.08	**12.21**	14.43
IM	18.20	**17.23**	52.91	57.31	60.75	22.97	25.20
AZ	**35.08**	54.23	236.60	471.45	892.14	48.34	59.04
DB	8.03	**7.92**	14.92	13.62	12.63	15.29	18.32
TV	9.7h	**9.2h**	23.6h	23.9h	- (17B)	×	- (6B)
LJ	**- (75B)**	- (65B)	- (18B)	- (16B)	- (15B)	×	- (2B)

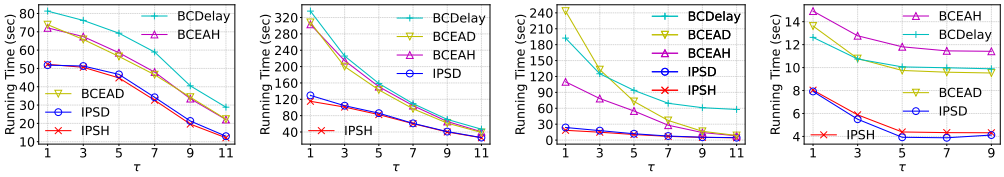


(a) GH (b) BC (c) TW (d) DB

Fig. 3. Results of delay testing.

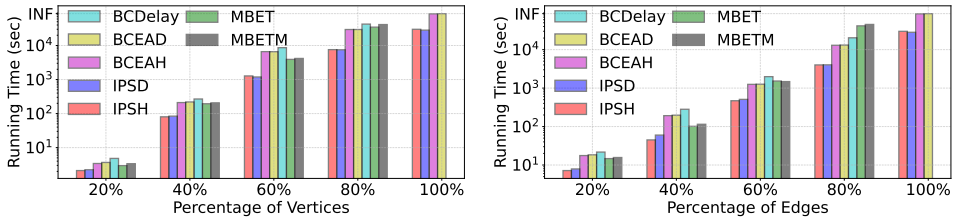
maximal bicliques than the other algorithms when all algorithms cannot finish the task within the time limit. On BS, our algorithms are not so efficient as MBET. The possible reason is that MBET uses the bitmap to compress the neighborhood information and compute the intersection by performing a bitwise AND operation. We do not utilized this data representation in our implementation, despite the potential to further enhance the acceleration of our algorithm.

(3) Delay test for enumerating all maximal bicliques (real datasets). Following [9], the delay is evaluated by dividing the running time for reporting the first k maximal bicliques by k. Figure 3 shows the results of the delay testing by varying k. IPSH and IPSD have the smallest delays across different datasets and their delays do not change a lot when we vary the number of the outputs. The possible reason is that IPSH and IPSD can output the maximal bicliques in a batch fashion when the condition for the new stopping criterion is satisfied, i.e., the delay for the maximal bicliques in the same batch is $O(\gamma)$, which is the optimal since we need at least $O(\gamma)$ to output a maximal biclique. This compensates for the fact that the delay between two consecutive batches could be exponential. Moreover, IPSH and IPSD outperform other algorithms when outputting a fixed number of results, which indicates that our algorithms are also suitable for the applications that only require to output a fixed number of results.



(a) GH (b) BC (c) TW (d) DB

Fig. 4. Comparison among baselines when enumerating size-constrained maximal bicliques (varying $\tau = \tau_L = \tau_R$).



(a) Varying % of vertices (TV) (b) Varying % of edges (TV)

Fig. 5. Results of scalability testing on TV dataset.

(4) Comparison among baselines when enumerating size-constrained maximal bicliques (real datasets). Figure 4 shows the result of varying $\tau = \tau_L = \tau_R$ for maximal biclique enumeration, respectively. As expected, our proposed IPSH and IPSD outperform other algorithms under most settings. Meanwhile, as the value of τ increases, the running time of all algorithms decreases. The reason is that the pruning rules become more powerful as τ increases and they can dramatically prune many unpromising branches so as to reduce the search space.

(5) Scalability test (real datasets). We conduct the scalability test on one hard dataset TV. Following existing studies [5, 9], we randomly sample 20%-80% vertices and edges from the dataset and build the induced subgraph. We show the results in Figure 5. We have the following observations. First, as the number of the sampled vertices/edges increases, the running time of all algorithms increases, among which IPSH and IPSD run the fastest. Besides, our algorithm can enumerate all these 19 billion maximal bicliques in this hard dataset within INF while other algorithms cannot.

(6) Space costs (real datasets). We report the space costs of different algorithms in Table 4. We have the following observations. First, our algorithms use slightly more memory than BCEAD and BCEAH to enumerate the maximal bicliques. The reason is that our algorithms need additional space for each branch $B = (S, C, X)$ to record the non-neighborhood information to check whether v should be skipped. Second, our algorithms use much less memory than MBET and MBETM. The reason is that MBET and MBETM store and update the whole prefix tree in an in-memory fashion, which increases dramatically as the algorithm proceeds.

6 Related Work

Maximal Biclique Enumeration. The maximal biclique enumeration problem has also been studied for decades. Earlier studies [16, 31, 43] enumerate all maximal bicliques in a breath-first search manner, where they first build the powerset of one side of the bipartite graph, e.g., L, and

Table 4. Space costs (unit: MB by default).

	IPSH	IPSD	BCEAH	BCEAD	BCDelay	MBET	MBETM
FR	6.53	6.53	6.40	**6.40**	6.40	7.82	7.56
WK	10.58	10.59	9.45	**9.21**	9.58	22.32	20.97
ES	9.91	9.89	8.87	**8.85**	8.86	16.59	15.68
YT	35.79	36.64	28.33	**28.25**	28.40	180.18	114.19
GH	47.57	48.16	38.10	**37.90**	38.83	2.04G	311.48
BC	106.86	105.10	90.27	90.60	**89.97**	4.29G	869.92
SO	152.56	160.99	**113.57**	113.72	113.74	425.65	380.91
AC	131.62	136.31	99.94	**99.86**	100.01	371.86	328.82
TW	179.06	203.48	140.37	**139.21**	141.22	639.70	442.07
BS	285.43	269.72	197.57	197.73	**197.14**	680.96	625.87
IM	317.99	329.87	239.94	**239.58**	240.13	839.36	803.77
AZ	691.27	679.07	**512.69**	513.07	514.91	1.90G	1.65G
DB	1.77G	1.72G	1.12G	**1.12G**	1.12G	3.56G	3.78G

then extract those that induce the maximal bicliques. Recent studies [2, 5, 7, 9, 11, 23, 44] usually enumerates the maximal bicliques in a depth-first search manner as they follow the Bron-Kerbosch branch-and-bound framework [4], which is originally designed for maximal clique enumeration problem. Specifically, [11, 23] reduce the problem to the maximal clique enumeration problem by transforming the bipartite graph into a general graph. [44] is the first study that adapts the Bron-Kerbosch branching strategy to enumerate maximal biclique directly. Motivated by the phenomenon that the pivoting strategy [30] can prune a significant number of branches, [2, 9] both employ the technique in biclique scenario but differ in which pivot is selected. [7] aims to optimize the order of vertices, introducing the unilateral vertex ordering to accelerate the enumeration. [5] further improves the efficiency and reduces the memory usage by using the prefix-tree to store the lists of elements compactly. However, all these branch-and-bound algorithms differ from IPS in that IPS involves a new pivoting strategy, which would theoretically reduce the number of created branches under the worst case, i.e., $O(\alpha^n)$ branches for IPS v.s. $O(\sqrt{2}^n)$ branches for all existing studies with $\alpha < \sqrt{2}$. This new pivot strategy also makes IPS and its variants have a strictly better time complexity than that of existing studies.

Other Biclique-related Problems. There are also several other interesting topics that involve the biclique inside. First, several biclique enumeration tasks impose the size constraints. [39] aims to find all (p, q)-bicliques in a graph, i.e., each returned biclique H satisfies $|L(H)| = p$ and $|R(H)| = q$, where p and q are user-specified parameters. It extends the ideas for k-clique listing to this setting with several optimizations such as preallocated arrays and vertex renumbering. [33, 34] target a special (p, q)-biclique with $p = q = 2$, a.k.a, butterfly, in the bipartite graph and design specialized algorithm for this task. Second, diversified top-k search is another topic that has been extensively studied, which aims to find top-k results that are not only most relevant to a query but also diversified. In the biclique enumeration setting, [20] studies the top-k diversified maximum biclique search, which aims to find k bicliques such that they cover the most number of edges. It extends the idea for maximum biclique search [19] by iteratively identifying and removing the maximum biclique in the remaining graph. [1] extends this problem with a size constraint t for one side, i.e., $|L(H)| = t$, while maximizing coverage on the other side. Third, the maximum biclique search problem has also gained significant research interest due to its wide range of applications in fields such as bioinformatics and social network analysis. There are three lines of research - (1) [10]

targets vertex-based maximum biclique search problem, which aims to find a biclique H^* such that $|V(H^*)|$ is maximized, (2) [19, 20, 27, 29] target edge-based maximum biclique search problem, which aims to find a biclique H^* such that $|E(H^*)|$ is maximized and (3) [6, 25, 37, 42, 46, 47] target the maximum balanced biclique search problem, which aims to find a biclique H^* such that $|L(H^*)| = |R(H^*)|$ and $|E(H^*)|$ is maximized. We note that the techniques used in the above studies are not suitable for our problem setting since (1) we do not impose an exact size constraint for the returned bicliques, (2) the bicliques returned can be overlapped for our problem and (3) we aim to find all maximal bicliques but not only the maximum one.

7 Conclusion

In this paper, we study the maximal biclique enumeration problem. We propose the algorithm IPS, which employs two techniques to improve performance. The first is a new termination condition, designed to find all maximal bicliques in a branch in nearly optimal time. The second is a new pivot selection strategy that aligns well with the early-termination technique. IPS achieves better worst-case time complexity than all existing algorithms for maximal biclique enumeration. Extensive experiments on several datasets demonstrate the efficiency of our algorithm and the effectiveness of the proposed techniques. An interesting research direction is to generalize the idea of the new pivoting strategy for clique-related problems, such as maximal clique enumeration, maximum clique search, and top-k diversified clique search.

Acknowledgments

We would like to thank the anonymous reviewers for providing constructive feedback and valuable suggestions. This work was supported by the National Natural Science Foundation of China (Grant No. 62506018). This work was also supported in part by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Award MOE-T2EP20224-0011 and Tier 1 Award (RG20/24)). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

References

[1] Aman Abidi, Lu Chen, Chengfei Liu, and Rui Zhou. 2022. On Maximising the Vertex Coverage for Top-k t-Bicliques in Bipartite Graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2346–2358.

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

[3] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. 1976. *Graph theory with applications*. Vol. 290. Macmillan London.

[4] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

[5] Jiujian Chen, Kai Wang, Rong-Hua Li, Hongchao Qin, Xuemin Lin, and Guoren Wang. 2023. Maximal Biclique Enumeration: A Prefix Tree Based Approach. (2023).

[6] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2021. Efficient exact algorithms for maximum balanced biclique search in bipartite graphs. In *Proceedings of the 2021 International Conference on Management of Data*. 248–260.

[7] 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.

[8] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. 2018. D2K: scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1272–1281.

[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] Michael R Garey and David S Johnson. 1979. *Computers and intractability*. Vol. 174. freeman San Francisco.

[11] Alain Gély, Lhouari Nourine, and Bachir Sadi. 2009. Enumeration aspects of maximal cliques and bicliques. *Discrete applied mathematics* 157, 7 (2009), 1447–1459.

[12] Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. 2020. Redundancy-free computation for graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 997–1005.

[13] Dieter Kratsch and FV Fomin. 2010. *Exact exponential algorithms*. Springer-Verlag Berlin Heidelberg.

[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] Michael Ley. 2002. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *International symposium on string processing and information retrieval*. Springer, 1–10.

[16] Jinyan Li, Guimei Liu, Haiquan Li, and Limsoon Wong. 2007. Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms. *IEEE Transactions on Knowledge and Data Engineering* 19, 12 (2007), 1625–1637.

[17] Xin Li and Hsinchun Chen. 2013. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems* 54, 2 (2013), 880–890.

[18] Guimei Liu, Kelvin Sim, and Jinyan Li. 2006. Efficient mining of large maximal bicliques. In *Data Warehousing and Knowledge Discovery: 8th International Conference, DaWaK 2006, Krakow, Poland, September 4-8, 2006. Proceedings 8*. Springer, 437–448.

[19] 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).

[20] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2022. Maximum and top-k diversified biclique search at scale. *The VLDB Journal* 31, 6 (2022), 1365–1389.

[21] Cristina Maier and Dan Simovici. 2021. On biclique connectivity in bipartite graphs and recommendation systems. In *Proceedings of the 2021 5th International Conference on Information System and Data Mining*. 151–156.

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

[23] Kazuhisa Makino and Takeaki Uno. 2004. New algorithms for enumerating all maximal cliques. In *Algorithm Theory-SWAT 2004: 9th Scandinavian Workshop on Algorithm Theory, Humlebæk, Denmark, July 8-10, 2004. Proceedings 9*. Springer, 260–272.

[24] David W Matula and Leland L Beck. 1983. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)* 30, 3 (1983), 417–427.

[25] Ciaran McCreesh and Patrick Prosser. 2014. An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem. In *Integration of AI and OR Techniques in Constraint Programming: 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings 11*. Springer, 226–234.

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

[27] Eran Shaham, Honghai Yu, and Xiao-Li 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*. SIAM, 315–323.

[28] Kelvin Sim, Jinyan Li, Vivekanand Gopalkrishnan, and Guimei Liu. 2009. Mining maximal quasi-bicliques: Novel algorithm and applications in the stock market and protein networks. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 2, 4 (2009), 255–273.

[29] Melih Sözdinler and Can Özturan. 2018. Finding maximum edge biclique in bipartite networks by integer programming. In *2018 IEEE International Conference on Computational Science and Engineering (CSE)*. IEEE, 132–137.

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

[31] Takeaki Uno, Masashi Kiyomi, Hiroki Arimura, et al. 2004. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Fimi*, Vol. 126.

[32] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. 501–508.

[33] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *PVLDB* (2019).

[34] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2023. Accelerated butterfly counting with vertex priority on bipartite graphs. *The VLDB Journal* 32, 2 (2023), 257–281.

[35] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2024. Efficient k-Clique Listing: An Edge-Oriented Branching Strategy. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–26.

[36] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2025. Maximal Clique Enumeration with Hybrid Branching and Early Termination. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 599–612.

[37] Yiyuan Wang, Shaowei Cai, and Minghao Yin. 2018. New heuristic approaches for maximum balanced biclique problem. *Information Sciences* 432 (2018), 362–375.
[38] Zhengren Wang, Yi Zhou, Mingyu Xiao, and Bakhadyr Khoussainov. 2022. Listing maximal k-plexes in large real-world graphs. In *Proceedings of the ACM Web Conference 2022*. 1517–1527.
[39] Jianye Yang, Yun Peng, and Wenjie Zhang. 2021. (p, q)-biclique counting and enumeration for large sparse bipartite graphs. *Proceedings of the VLDB Endowment* 15, 2 (2021), 141–153.
[40] Kaiqiang Yu and Cheng Long. 2023. Maximum k-biplex search on bipartite graphs: A symmetric-bk branching approach. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.
[41] Kaiqiang Yu, Cheng Long, Shengxin Liu, and Da Yan. 2022. Efficient algorithms for maximal k-biplex enumeration. In *Proceedings of the 2022 International Conference on Management of Data*. 860–873.
[42] Bo Yuan, Bin Li, Huanhuan Chen, and Xin Yao. 2014. A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem. *IEEE transactions on cybernetics* 45, 5 (2014), 1054–1067.
[43] Mohammed J Zaki and Ching-Jui Hsiao. 2002. CHARM: An efficient algorithm for closed itemset mining. In *Proceedings of the 2002 SIAM international conference on data mining*. SIAM, 457–473.
[44] 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 (2014), 1–18.
[45] Ding Zhou, Sergey A Orshanskiy, Hongyuan Zha, and C Lee Giles. 2007. Co-ranking authors and documents in a heterogeneous network. In *Seventh IEEE international conference on data mining (ICDM 2007)*. IEEE, 739–744.
[46] Yi Zhou and Jin-Kao Hao. 2017. Combining tabu search and graph reduction to solve the maximum balanced biclique problem. *arXiv preprint arXiv:1705.07339* (2017).
[47] Yi Zhou, André Rossi, and Jin-Kao Hao. 2018. Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs. *European Journal of Operational Research* 269, 3 (2018), 834–843.

Received October 2025; revised January 2026; accepted February 2026