Efficient Maximal Biplex Enumerations with Improved Worst-Case Time Guarantee

QIANGQIANG DAI, Beijing Institute of Technology, China
RONG-HUA LI, Beijing Institute of Technology, China
DONGHANG CUI, Beijing Institute of Technology, China
MEIHAO LIAO, Beijing Institute of Technology, China
YU-XUAN QIU, Shenzhen University, China
GUOREN WANG, Beijing Institute of Technology, China

A k-biplex is an induced subgraph of a bipartite graph which requires every vertex on the one side disconnecting at most k vertices on the other side. Enumerating all maximal k-biplexes in a bipartite graph is a fundamental operator in bipartite graph analysis and finds applications in various domains, including community detection, online recommendation, and fraud detection in finance networks. The state-of-the-art solutions for maximal k-biplex enumeration suffer from efficiency issues as k increases ($k \geq 2$), with the time complexity of $O(m2^n)$, where n (m) denotes the number of vertices (edges) in the bipartite graph. To address this issue, we propose two theoretically and practically efficient enumeration algorithms based on novel branching techniques. Specifically, we first devise a new branching rule as a fundamental component. Building upon this, we then develop a novel branch-and-bound enumeration algorithm to efficiently enumerate maximal k-biplexes. We prove that our algorithm achieves a worst-case time complexity of $O(m\alpha_k^n)$, where $\alpha_k < 2$, thus significantly improving the time complexity compared to previous algorithms. To enhance the performance, we further propose an improved enumeration algorithm based on a novel pivot-based branching rule. Theoretical analysis reveals that our improved algorithm has a time complexity of $O(m\beta_k^n)$, where β_k is strictly less than α_k. In addition, we also present several non-trivial optimization techniques, including graph reduction, upper-bounds based pruning, and ordering-based optimization, to further improve the efficiency of our algorithms. Finally, we conduct extensive experiments on 6 large real-world bipartite graphs to evaluate the efficiency and scalability of the proposed solutions. The results demonstrate that our improved algorithm achieves up to 5 orders of magnitude faster than the state-of-the-art solutions.

CCS Concepts: • **Theory of computation → Backtracking**.

Additional Key Words and Phrases: bipartite graph, cohesive subgraph, k-biplex, branch-and-bound

ACM Reference Format:
Qiangqiang Dai, Rong-Hua Li, Donghang Cui, Meihao Liao, Yu-Xuan Qiu, and Guoren Wang. 2024. Efficient Maximal Biplex Enumerations with Improved Worst-Case Time Guarantee. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 135 (June 2024), 26 pages. https://doi.org/10.1145/3654938

Authors' addresses: Qiangqiang Dai, qiangd66@gmail.com, Beijing Institute of Technology, Beijing, China; Rong-Hua Li, lironghuabit@126.com, Beijing Institute of Technology, Beijing, China; Donghang Cui, cuidonghang@bit.edu.cn, Beijing Institute of Technology, Beijing, China; Meihao Liao, mhliao@bit.edu.cn, Beijing Institute of Technology, Beijing, China; Yu-Xuan Qiu, qiuyx.cs@gmail.com, Shenzhen University, Shenzhen, China; Guoren Wang, wanggrbit@126.com, Beijing Institute of Technology, Beijing, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 2836-6573/2024/6-ART135
https://doi.org/10.1145/3654938

1 INTRODUCTION

Many real-world networks, such as author-publication networks [26], user-item networks [40], and biological networks [27], can be modeled as bipartite graphs, where the vertices are divided into two independent sets, representing two different types of entities, while the edges signify associations or connections between vertices belonging to the two sets. Mining cohesive subgraphs from a bipartite graph has emerged as a crucial technique to enable valuable insights and predictions regarding the relationships and structures within these networks. For example, in user-item networks, users sharing similar preferences tend to interact with comparable sets of items, thereby frequently forming dense subgraphs within the networks. The identification of these cohesive subgraphs holds significant potential for facilitating personalized recommendations [21, 35]. Similarly, in gene expression networks, each gene is correlated in a set of samples, and local patterns on gene expression data often give rise to dense subgraphs. The identification of cohesive subgraphs from these gene expression networks can contribute to discovering biological activities that are common to a set of genes [19, 36].

The classical biclique model [10, 17, 18, 28, 32] is a widely-used approach to model cohesive subgraphs in bipartite graphs, where each vertex of one side is connected to all vertices of the other side. This model has demonstrated successful applications in various bipartite graph analysis tasks, including community detection [7, 23, 25], biological network analysis [8, 39], and anomaly detection [2, 6]. However, the strict requirement of pairwise connections between two sides of vertices might be overly restrictive for cohesive subgraph mining, as disconnecting a few edges within a subgraph could still indicate its cohesion [22, 38]. To remedy this issue, several relaxed biclique models, such as k-biplex [38, 49] and quasi biclique [33, 42], have also been developed. In this paper, we primarily focus on the k-biplex model [38], which is a classic and valuable approximation of the biclique model and finds extensive applications across real-world scenarios, such as community detection [16, 38] and fraud detection [48].

Given a bipartite graph G, a subgraph H is referred to as a maximal k-biplex of G if (1) every vertex on the one side disconnects at most k vertices on the other side; and (2) there does not exist any other subgraph of G that contains H while still satisfying condition (1). To enumerate all maximal k-biplexes from a bipartite graph, several advanced approaches have been developed in recent years [16, 38, 48, 49]. Among them, the state-of-the-art is a pivot-based enumeration algorithm [16]. Such an approach, however, still suffers from the following issues. First, its practical performance is not sufficiently efficient, which often requires several hours to process medium-size bipartite graphs (as shown in [16]). Moreover, as k increases ($k \geq 2$), the performance of this approach can be quickly reduced even when handling medium-size graphs. Second, the worst-case time complexity of such an approach is $O(m2^n)$ [16], which is the same as that of the straightforward brute-force enumeration algorithm. Furthermore, as discussed in [16], it is difficult to achieve a tighter time complexity using their pivot-based enumeration technique. Therefore, developing both theoretical and practical efficient solutions for enumerating all maximal k-biplexes on bipartite graphs remains a very challenging problem.

Contributions. To address the aforementioned challenges, we propose two novel maximal k-biplex enumeration algorithms based on a carefully-designed branching strategy and a pivoting technique. The striking feature of our algorithms is that they not only offer much better practical performance but also have improved worst-case time complexity guarantees (compared to the state-of-the-art algorithm). In summary, the main contributions of our paper are as follows.

Novel enumeration algorithms. We propose two new algorithms for efficiently enumerating maximal k-biplexes on the bipartite graphs. Our first algorithm is based on a new branch-and-bound method with a carefully-designed branching rule. We prove that this algorithm can achieve an improved

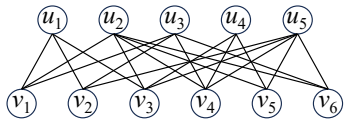


Fig. 1. Running example: a bipartite graph G.

worst-case time complexity of $O(m\alpha_k^n)$, where n and m are the numbers of vertices and edges of the bipartite graph, respectively, and α_k is a positive real number strictly smaller than 2. To enhance the efficiency, we also propose an improved enumeration algorithm that incorporates a novel pivoting technique along with an improved branching rule. Interestingly, we prove that this improved algorithm achieves an even tighter worst-case time complexity of $O(m\beta_k^n)$, where β_k is a positive real number strictly smaller than α_k. Specifically, when $k = 0, 1$, and 2, β_k takes on the values 1.414, 1.618, and 1.839, respectively, To the best of our knowledge, our algorithms are the first algorithms that can break the $O(2^n)$ complexity for enumerating maximal k-biplexes, and our improved algorithm represents the best solution to date in terms of the worst-case time complexity.

New optimization techniques. To further improve the efficiency of our algorithms, we also develop several non-trivial optimization techniques, including graph reduction, upper-bounds based pruning, and ordering-based optimization. Specifically, we prove that each edge in a maximal k-biplex with the size of each side no less than q is associated with a minimum of $(q-k-1)(q-2k-1)$ butterflies, where the butterfly refers to a 2×2 biclique. Based on this, we present a novel butterfly-based reduction technique to significantly reduce the bipartite graphs without losing any k-biplex (with the size of each side no less than q). Furthermore, we develop a new upper-bound technique as well as a near-linear time upper-bound computation algorithm, enabling efficient pruning the unnecessary branches in the enumeration process. In addition, we present an ordering-based optimization technique to further reduce the search space of our enumeration algorithms.

Extensive experimental evaluations. We construct extensive experiments to evaluate the efficiency and scalability of the proposed algorithms on 6 large real-world bipartite graphs. The experimental results demonstrate that our algorithms substantially outperform the state-of-the-art algorithm for maximal k-biplex enumeration by up to 5 orders of magnitude across various parameter settings. For example, On the Amazon dataset (with 3.7 million edges), our best algorithm takes less than 5 seconds to enumerate all maximal 3-biplexes with sizes no less than 15. In contrast, the state-of-the-art algorithm failed to terminate within 24 hours under identical conditions. To ensure reproducibility, we make the source code of this work available at https://github.com/dawhc/MaximalBiPlex.

2 PROBLEM DEFINITION

Let $G = (L, R, E)$ be an undirected and unweighted bipartite graph with two disjoint vertices sets L and R and an edge set $E \subseteq L \times R$. We denote by $n = |L| + |R|$ and $m = |E|$ the number of vertices and edges of G, respectively. For a vertex $v \in L$ (resp. $u \in R$), the set of neighbors of v (resp. u) in G is denoted by $N_v(G) = \{w \in R | (v, w) \in E\}$ (resp. $N_u(G) = \{w \in L | (u, w) \in E\}$). Then, the degree of $v \in L$ (resp. $u \in R$) in G is defined as $d_v(G) = |N_v(G)|$ (resp. $d_u(G) = |N_u(G)|$). Given a pair of vertex sets (A, B) with $A \subseteq L$ and $B \subseteq R$, we define $G(A, B) = (A, B, E_{A,B})$ as a subgraph of G induced by the vertex sets A and B, where $E_{A,B} = \{(v, u) \in E | v \in A, u \in B\}$. Below, we give the definition of the k-biplex [38].

Definition 1 (k-biplex). Given a bipartite graph $G = (L, R, E)$, a positive integer k, and two vertex sets $S_L \subseteq L$ and $S_R \subseteq R$, the subgraph $G(S_L, S_R)$ is a k-biplex if every vertex $v \in S_L$ has a degree no less than $|S_R| - k$ and every vertex $u \in S_R$ has a degree no less than $|S_L| - k$.

A k-biplex $G(S_L, S_R)$ is said to be maximal in G if no other k-biplex $G(S'_L, S'_R)$ satisfies $S_L \subseteq S'_L$ and $S_R \subseteq S'_R$. To simplify the presentation, we will refer to the sets of (S_L, S_R) as a k-biplex if $G(S_L, S_R)$ is a k-biplex in G. Below, we give a notable property of the k-biplex.

Property 1 (Hereditary property [49]). The k-biplex meets the hereditary property for any positive integer k, i.e., given a k-biplex (A, B) of G, (A', B') is also a k-biplex of G for every $A' \subseteq A$ and $B' \subseteq B$.

This nice property suggests that the communities in bipartite graphs detected by the k-biplex are often robust, since the community structure cannot be destroyed even if some individuals are removed. Furthermore, by setting $k = 0$, the k-biplex model degenerates to the traditional biclique model. As a result, k-biplex can also be applied to various real-world applications, similar to the biclique. However, many maximal k-biplexes with small sizes may not have practical application value since they can be too sparse and unsuitable for modeling real communities. For instance, consider two edges (u_1, v_1) and (u_2, v_2) of a bipartite graph G. If the distance between u_1 (resp. v_1) and u_2 (resp. v_2) is greater than 2, then the sets $\{u_1, u_2\}$ and $\{v_1, v_2\}$ form a maximal 1-biplex. It is easy to verify that this maximal 1-biplex is not a cohesive subgraph of G and has limited practical utility. Thus, it is more meaningful to mine relatively-large maximal k-biplexes. In addition, relative-large k-biplexes are often very cohesive as indicated by the following lemma.

Lemma 1 ([16]). Given a k-biplex $G(A, B)$ of G, the diameter of $G(A, B)$ is at most 3 if $|A| \geq 2k + 1$ and $|B| \geq 2k + 1$.

As shown in Lemma 1, any k-biplex (A, B) of G with $|A| \geq 2k + 1$ and $|B| \geq 2k + 1$ must be densely-connected, making it a more probable representation of communities in real-world bipartite graphs. Therefore, this paper focuses mainly on enumerating relatively-large k-biplexes. In the rest of this paper, if the context is clear, enumerating all k-biplexes means enumerating all relatively-large k-biplexes. Below, we formally define our problems.

Problem definition. Given a bipartite graph G and a threshold $q \geq 2k + 1$, the goal of this paper is to enumerate all maximal k-biplexes in G with the sizes of both sides no less than q.

2.1 Existing Solutions

The problem of enumerating all maximal k-biplexes of G has been proven to be NP-hard [16, 46], and several approaches have been proposed to address this problem [16, 38, 48, 49]. Below, we provide a brief overview of these existing solutions.

Reverse search based algorithms. The algorithm, which was first introduced by [48], leverages a reverse search framework [3, 12] that was originally proposed for enumerating all maximal subgraphs with the hereditary property. In general, the algorithm operates by following three primary steps. Firstly, the algorithm identifies a (random) single solution as its initial input. Secondly, the algorithm makes use of a local search to identify the new local solutions (which may not be maximal) from each almost-satisfying subgraph generated by the current solution. Here, a graph H is considered to be an almost-satisfying subgraph if it is not a k-biplex but would become one upon the removal of a single vertex. Lastly, the algorithm maximizes each local solution and iteratively employs the newly-discovered solution to continue the local search operations.

A nice feature of this algorithm is its ability to achieve polynomial delay time complexity [48]. However, this algorithm may involve many unnecessary local search calculations that may produce duplicate solutions and require the storage of every solution to avoid redundancy. In addition, when using this algorithm to enumerate relatively-large maximal k-biplexes, it may still be necessary to enumerate all maximal k-biplexes of an input graph since some relatively-large maximal k-biplexes may only be identified in the almost-satisfying subgraph whose size is less than q on both sides. Consequently, this algorithm can be extremely inefficient when processing large real-world graphs.

Set enumeration based algorithms. The fundamental algorithms, as introduced in [38, 49], employ the set enumeration technique that iteratively examines each subset of the given candidate sets to obtain all maximal k-biplexes. This enumeration process generates a prefix tree structure, leading to these algorithms being commonly referred to as prefix tree-based algorithms.

Since the prefix tree based algorithms detect a large number of non-maximal k-biplexes, it is also rather inefficient. To address this issue, Dai et al. [16] recently proposed a pivot-based set enumeration algorithm. Specifically, given the candidate sets $C_L \subseteq L$ and $C_R \subseteq R$ to expand the current k-biplex (A, B), the authors found that the remaining maximal k-biplexes to be detected can be identified by expanding only a portion of vertices in C_L and C_R. This is achieved by selecting a pivot vertex $v \in C_L$ (resp. $v \in C_R$) to expand the current (A, B), and then expanding the k-biplex only using the vertices in $\{w \in C_L | (B \setminus N_v(G)) \subsetneq N_w(G)\}$ and $C_R \setminus N_v(G)$ (resp. $C_L \setminus N_v(G)$ and $\{w \in C_R | (A \setminus N_v(G)) \subsetneq N_w(G)\}$). As a result, the pivoting technique can prune many redundant branches, making it efficient in processing real-world graphs. To the best of our knowledge, this pivot-based algorithm is the state-of-the-art approach to this problem.

Unfortunately, the worst-case time complexity of the pivot-based algorithm is $O(m2^n)$, which is trivially equal to the brute-force enumeration algorithm. Moreover, when using this pivot-based algorithm to enumerate relatively-large maximal k-biplexes (whose sizes of both sides are no less than a threshold q), it can be very costly in processing large real-world graphs, since this pivoting technique is ineffective in reducing the enumeration of unnecessary maximal k-biplexes of G. For instance, when setting $k = 3$, the algorithm cannot complete computations within 24 hours even when $q = 20$ on most datasets (see Sec. 5). Therefore, there is an urgent need for an algorithm that can address our problem both theoretically and practically. In the following sections, we will develop novel algorithms with a better worst-case time complexity guarantee, followed by some key optimization techniques to further improve the efficiency of our algorithms.

3 THE PROPOSED ALGORITHMS

In this section, we develop two new algorithms for efficiently enumerating maximal k-biplexes of G. Notably, all proposed algorithms in this section have time complexities that can break the barrier of $O(2^n)$. To our knowledge, the time complexity of our best algorithm is significantly lower than that of all existing algorithms.

3.1 The Proposed Basic Algorithm

Our basic algorithm is inspired by the traditional branch-and-bound technique [11, 24], which selects a specific vertex v of G to divide the current problem into two subproblems. The first subproblem involves enumerating all maximal k-biplexes that contain v, while the second subproblem involves enumerating all maximal k-biplexes that exclude v. For convenience, we refer to such a vertex v as the *branching vertex*. However, it is worth noting that the random selection of each vertex from G as a branching vertex can lead to significant computational overhead. Especially in the worst-case scenario, this method carries the risk of exhaustively enumerating all subsets of G, potentially generating a total of up to $O(2^n)$ subbranches. To tackle this problem, we propose a new branching rule for selecting branching vertices, whose key idea is as follows.

Key idea. Consider (S_L, S_R) as a k-biplex, and let (C_L, C_R) be candidate sets containing all possible vertices for expanding (S_L, S_R). Specifically, for each v in (C_L, C_R), adding v to (S_L, S_R) will generate a larger k-biplex. It is evident that the smaller size of the candidate sets (C_L, C_R) generally leads to a lower computational cost for enumerating the maximal k-biplexes containing (S_L, S_R). Based on this observation, we note that selecting non-neighboring vertices of the vertex u that is included in S_L (or S_R) as branching vertices is remarkably helpful in reducing the size of (C_L, C_R). The rationale behind this approach is that once the current k-biplex (S_L, S_R) already includes k non-neighbors

Algorithm 1: The basic branch-and-bound algorithm

Input: Bipartite graph $G = (L, R, E)$, two parameters k and q
Output: All relatively-large maximal k-biplexes of G

$Branch(\emptyset, \emptyset, L, R, \emptyset, \emptyset)$;
Function: $Branch(S_L, S_R, C_L, C_R, X_L, X_R)$
  if $|S_L \cup C_L| < q$ *or* $|S_R \cup C_R| < q$ **then return**;
  if $C_L = \emptyset$ *and* $C_R = \emptyset$ **then**
    if $X_L = \emptyset$ *and* $X_R = \emptyset$ **then**
      if $|S_L| \geq q$ *and* $|S_R| \geq q$ **then** Output (S_L, S_R);
    return;
  $v \leftarrow \arg\max_{v \in S_L} \overline{d}_v(S_R \cup C_R)$; $\overline{d}_L \leftarrow \overline{d}_v(S_R \cup C_R)$;
  $u \leftarrow \arg\max_{u \in S_R} \overline{d}_u(S_L \cup C_L)$; $\overline{d}_R \leftarrow \overline{d}_u(S_L \cup C_L)$;
  $w \leftarrow -1$;
  if $\overline{d}_L > k$ **then** $w \leftarrow$ a vertex in $\overline{N}_v(C_R)$;
  else if $\overline{d}_R > k$ **then** $w \leftarrow$ a vertex in $\overline{N}_u(C_L)$;
  else
    if $(S_L \cup C_L, S_R \cup C_R)$ *is a* k*-biplex* **then**
      Output $(S_L \cup C_L, S_R \cup C_R)$ if it is the maximal;
      return;
    $w \leftarrow$ the vertex in C_L with smallest degree in $S_R \cup C_R$;
  if $\overline{d}_L > k$ **then** $BranchB(S_R, w, S_L, C_R, C_L, X_R, X_L)$;
  else $BranchB(S_L, w, S_R, C_L, C_R, X_L, X_R)$;
Function: $BranchB(S_L, v, S_R, C_L, C_R, X_L, X_R)$
  $(C'_L, C'_R) \leftarrow Updates(S_L, v, S_R, C_L, C_R)$;
  $(X'_L, X'_R) \leftarrow Updates(S_L, v, S_R, X_L, X_R)$;
  $Branch(S_L \cup \{v\}, S_R, C'_L, C'_R, X'_L, X'_R)$;
  $Branch(S_L, S_R, C_L \setminus \{v\}, C_R, X_L \cup \{v\}, X_R)$;

of u, all the remaining non-neighbors of u can be directly excluded from (C_L, C_R). Moreover, it is easy to find that if the vertex $u \in S_L$ (or $u \in S_R$) has a smaller degree in (C_L, C_R), it will be more beneficial to reduce the size of (C_L, C_R) by preferentially selecting non-neighbors of such vertex u as the branching vertices. Motivated by this idea, the proposed branching rule is outlined as follows.

Proposed branching rule. Given a vertex $v \in L$ and a set $B \subseteq R$, we denote by $N_v(B)$ ($\overline{N}_v(B) = B \setminus N_v(B)$) the set of neighbors (non-neighbors) of v in B and define $d_v(B)$ ($\overline{d}_v(B)$) as $|N_v(B)|$ ($|\overline{N}_v(B)|$). Consider a k-biplex (S_L, S_R) and the candidate sets (C_L, C_R) in a recursive call. We can use the following rule for selecting the branching vertex. Note that we only consider the case where the branching vertex is selected from C_R, as the branching vertex from C_L can be obtained by a similar method.

- Finding a vertex $v \in S_L$ with the smallest degree in $S_R \cup C_R$, i.e., $\overline{d}_v(S_R \cup C_R) \leq \overline{d}_{v'}(S_R \cup C_R)$ for each $v' \in S_L$.
- If $\overline{d}_v(S_R \cup C_R) > k$, we select a vertex u in $\overline{N}_v(C_R)$ as the branching vertex.
- If $\overline{d}_v(S_R \cup C_R) \leq k$, we select a vertex u in C_R whose degree in $S_L \cup C_R$ is smallest as the branching vertex, i.e., the vertex $u \in C_R$ satisfies $\overline{d}_u(S_L \cup C_L) \leq \overline{d}_{u'}(S_L \cup C_L)$ for each $u' \in C_R$.

Algorithm 2: $Updates(S_L, v, S_R, C_L, C_R)$

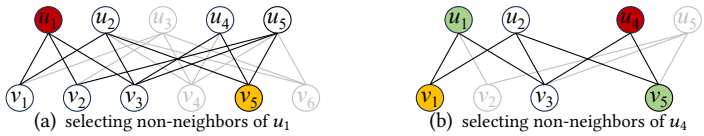
```
1 C'_L ← C_L \ {v}; C'_R ← C_R ∩ N_v(G);
2 foreach u ∈ C_R \ N_v(G) do
3     if d̄_u(S_L ∪ {v}) ≤ k ∧ d̄_v(S_R ∪ {u}) ≤ k then
4         C'_R ← C'_R ∪ {u};
5 foreach u ∈ S_R \ N_v(G) do
6     if d̄_u(S_L ∪ {v}) = k then C'_L ← C'_L ∩ N_u(G);
7 return (C'_L, C'_R);
```

Implementation details. Armed with the above branching rule, we develop our basic algorithm, as shown in Algorithm 1.

Algorithm 1 begins by invoking the *Branch* procedure, which utilizes the proposed branching rule for the branch-and-bound approach, to enumerate the maximal k-biplexes on G (line 1). This *Branch* procedure takes six parameters: S_L, S_R, C_L, C_R, X_L, and X_R. Here, (S_L, S_R) refers to a k-biplex, (C_L, C_R) are the candidate sets used to expand (S_L, S_R), and (X_L, X_R) are exclusion sets containing all vertices in (C_L, C_R) that have been used to expand (S_L, S_R). Before the branch-and-bound procedure, the algorithm needs to select a vertex w from (C_L, C_R) as the branching vertex following the proposed branching rule (lines 8-17). Specifically, it first obtains a vertex v in S_L (resp. u in S_R) that has the smallest in $S_R \cup C_R$ (resp. $S_L \cup C_L$). If the number of non-neighbors of v in $S_R \cup C_R$ (or u in $S_L \cup C_L$) is greater than k, the procedure selects the branching vertex w from $\overline{N}_v(C_R)$ (or $\overline{N}_u(C_R)$) (lines 8-12). Otherwise, the procedure selects $w \in C_R$ that has the smallest degree in $S_L \cup C_L$ as the branching vertex (line 17). It is noteworthy that if the current search space corresponds to a maximal result, the ongoing recursive call will terminate and produce $(S_L \cup C_L, S_R \cup C_R)$ as the maximal solution (lines 14-16). Subsequently, the algorithm continues the branch-and-bound procedure with vertex w (lines 18-19). When both C_L and C_R are the empty sets (line 4), the recursive call terminates and outputs (S_L, S_R) as a result, provided that X_L and X_R are also the empty sets (lines 5-7).

When the branching vertex w is added into (S_L, S_R), the corresponding candidate sets (resp. exclusion sets) need to be updated to ensure that all remaining vertices can also be used to expand the new k-biplex $(S_L, S_R) \cup \{w\}$. To achieve this, we present a procedure shown in Algorithm 2. Specifically, if the vertex $v \in C_L$ is added to S_L, every vertex $u \in C_R$ (resp. $u \in X_R$) must satisfy that either u is the neighbor of v, or $\overline{d}_v(S_R) < k$ and u has at most $k-1$ non-neighbors in S_L (lines 1-4). For each $u \in C_L$ (resp. $u \in X_L$), it must be the common neighbor of the vertices in $\overline{N}_v(S_R)$ that have exactly k non-neighbors in $S_L \cup \{v\}$ (lines 5-6). By following these constraints, the remaining vertices in candidate sets (resp. exclusion sets) can be used to expand the newly-created k-biplex. The following example further illustrates the idea of the proposed Algorithm 1.

Example 1. Consider a graph $G = (L, R, E)$ depicted in Fig. 1, with $k = 1$. We initialize the current k-biplex as $(\emptyset, \emptyset)$ and its candidate sets as $(\{u_1, ..., u_5\}, \{v_1, ..., v_6\})$. Following the proposed branching rule, we select u_1 to expand $(\emptyset, \emptyset)$ due to its maximum number of non-neighbors in (C_L, C_R). Subsequently, each non-neighbor of u_1 becomes the branching vertex to further expand $(\{u_1\}, \emptyset)$. Assuming v_5 is selected, we can exclude the vertices $(\{u_3\}, \{v_4, v_6\})$ from the candidate sets to expand $(\{u_1\}, \{v_5\})$, obtaining the result shown in Fig. 2(a). Similarly, we iteratively utilize the vertices u_4 and v_1 to expand $(\{u_1\}, \{v_5\})$. Notably, upon adding v_1 to $(\{u_1, u_4\}, \{v_5\})$, we can further eliminate the vertices u_5 and v_2 from the current candidate sets, as depicted in Fig. 2(b). By continuing our branching rule, we can obtain a result $(\{u_1, u_2, u_4\}, \{v_1, v_3, v_5\})$.



(a) selecting non-neighbors of u_1 (b) selecting non-neighbors of u_4

Fig. 2. An illustrative example for the proposed branching rule when $k = 1$, where red vertices and green vertices represent current k-biplexes, the yellow vertices are the branching vertices, and the gray vertices are removed from (C_L, C_R).

We next analyze the time and space complexity of Algorithm 1, which are presented in Theorem 3.1 and Theorem 3.2.

Theorem 3.1. *The worst-case time complexity of Algorithm 1 is $O(m\alpha_k^n)$, where $\alpha_k < 2$ is the maximum real-root of equation $x^{k+3} - 2x^{k+2} + 1 = 0$. When $k = 0$, 1, and 2, we have $\alpha_0 = 1.618$, $\alpha_1 = 1.839$, and $\alpha_2 = 1.928$, respectively.*

Proof. Denote by $T(n)$ the total number of leaves of the enumeration tree generated by *Branch*, where $n = |L| + |R|$. Then, the time complexity of Algorithm 1 is $O(mT(n))$, since each recursive call of *Branch* takes at most $O(m)$ time to find the vertex that has the smallest degree and $O(n + kd_{max})$ time to update the corresponding candidate sets and exclusion sets, where d_{max} is the larges degree of vertices in G. Based on the branch-and-bound technique, we can derive a base recurrence:

$$T(n) \leq T(n-1) + T(n-1). \tag{1}$$

Let v be the vertex in S_L (or S_R) with the smallest $d_v(S_R \cup C_R)$ (or $d_v(S_L \cup C_L)$). Then, $T(n)$ can be tightened by the following analysis.

(1) If $\overline{d}_v(S_R \cup C_R) > k$, a vertex $w \in \overline{N}_v(C_R)$ is selected as the branching vertex. When w is added to S_R, we note that v still has the smallest degree in $S_R \cup C_R$ among all vertices in S_L if there is no vertex removed from $(C_L, C_R \setminus \{w\})$. Then, another vertex in $\overline{N}_v(C_R \setminus \{w\})$ can be selected as the branching vertex in the recursive call that enumerates maximal k-biplexes containing $(S_L, S_R \cup \{w\})$. Thus, the recurrence of $T(n)$ can also be:

$$T(n) \leq T(n-1) + T(n-2) + T(n-2). \tag{2}$$

This process can continue until k vertices in $\overline{N}_v(C_R)$ are added to S_R. Assume that the vertices in $P \subseteq \overline{N}_v(S_R)$ with $|P| = k$ are added into S_R. The following recurrence can be derived:

$$T(n) \leq \sum_{i=1}^{k} T(n-i) + T(n-|P|). \tag{3}$$

Based on the definition of k-biplex, when P with $|P| = k$ is added into S_R, all vertices in the candidate sets of C_R side must be the neighbors of v. This means that the size of the candidate sets of the recursive call that enumerates maximal k-biplexes containing $(S_L, S_R \cup P)$ is at most $n - \overline{d}_v(S_R \cup C_R)$. As a result, the branch $T(n - |P|)$ can be replaced with $T(n - \overline{d}_v(S_R \cup C_R))$. Since $\overline{d}_v(S_R \cup C_R) > k$, we then have:

$$T(n) \leq \sum_{i=1}^{k} T(n-i) + T(n-k-1). \tag{4}$$

(2) If $\overline{d}_v(S_R \cup C_R) \leq k$, a vertex $w \in C_L$ with the smallest degree in $S_R \cup C_R$ is selected as the branching vertex. We note that in the subbranch that enumerates maximal k-biplexes containing $(S_L \cup \{w\}, S_R)$, the vertex w has the smallest degree in $S_R \cup C_R$. If $\overline{d}_w(S_R \cup C_R) > k$, the recurrence

of this subbranch is equivalent to Eq. (4). Then, we have the following recurrence for the case $\overline{d}_w(S_R \cup C_R) > k$:

$$T(n) \leq \sum_{i=1}^{k+1} T(n-i) + T(n-k-2). \tag{5}$$

If $\overline{d}_w(S_R \cup C_R) \leq k$, every vertex in (C_L, C_R) has at most k non-neighbors in $(S_L \cup C_L, S_R \cup C_R)$. Thus, $(S_L \cup C_L, S_R \cup C_R)$ is a k-biplex and such a recursive call can be terminated.

In summary, the worst-case recurrence is $T(n) \leq \sum_{i=1}^{k+2} T(n-i)$. Based on a theoretical result [20] that for a given linear recurrence $Z(n) = \sum_{i=1}^{j} Z(n-a_i)$, the size of $Z(n)$ is bounded by $O(\gamma^n)$, where γ is the maximum real-root of function $x^n - \sum_{i=1}^{j} x^{n-a_i} = 0$. We then can derive that the maximum size of $T(n)$ is bounded by $O(\alpha_k^n)$, where α_k is the maximum real-root of function $x^{k+3} - 2x^{k+2} + 1 = 0$. Thus, this completes the proof. □

Theorem 3.2. *The worst-case space complexity of Algorithm 1 is $O(\delta n + m)$, where δ is the size of maximum k-biplex of G.*

Proof. Initially, Algorithm 1 requires $O(m)$ space to store the graph G. Then, within the *Branch* procedure, we note that only the subbranch that enumerates maximal k-biplexes containing a vertex v requires an additional $O(n)$ space to store new candidate sets and exclusion sets. Thus, *Branch* takes at most $O(\delta n + m)$ space based on a depth-first search strategy and a restriction that at most δ vertices can be moved to (S_L, S_R). □

3.2 The Proposed Pivot-Based Algorithm

Although Algorithm 1 achieves lower time complexity compared to the state-of-the-art algorithm [16], we observe that this algorithm still involves many unnecessary computations. The reason behind it is that when a vertex v is selected as a branching vertex, Algorithm 1 proceeds to select every remaining vertex in the candidate sets as the branching vertex in the subbranches that enumerate maximal k-biplexes excluding v. Consequently, Algorithm 1 enumerates numerous non-maximal k-biplexes, leading to inefficiency. For example, consider the bipartite graph G shown in Fig. 1. When $k = 1$ and selecting u_1 as the branching vertex, a maximal 1-biplex $(\{u_1, u_2, u_3, u_4\}, \{v_1, v_3, v_4\})$ can be identified. However, in the subsequent subbranch that enumerates maximal k-biplexes excluding u_1, we note that the non-maximal 1-biplex $(\{u_2, u_3, u_4\}, \{v_1, v_3, v_4\})$ can also be enumerated by selecting v_1 as the branching vertex, thereby resulting in redundant enumerations.

To address the aforementioned issue, our focus lies in reducing the number of vertices selected as the branching vertex in the candidate sets during each recursive call. To achieve our goal, we develop a new pivoting technique that determines which vertices in the candidate sets require to be selected as the branching vertex. The detailed rule is presented in the following theorem.

Theorem 3.3 (Pivoting rule in C). *Given that $\nexists u \in S_R$ with $\overline{d}_u(S_L \cup C_L) > k$ in a recursive call, and let a vertex $v \in C_L$ be the pivot vertex. Then, all vertices in $(C_L \setminus \{v\}, C_R \cap N_v(G))$ are unnecessary for selecting as the branching vertices to enumerate maximal k-biplexes containing (S_L, S_R).*

Proof. Denote by $\{u_1, u_2, ..., u_h\} = C_R \setminus N_v(G)$, where $h = |C_R \setminus N_v(G)|$. When selecting vertices in $(\{v\}, \{u_1, u_2, ..., u_h\})$ as the branching vertices, we can derive that the original problem can be divided into the following subproblems:

(1) the subproblem of enumerating all maximal k-biplexes that contain v.
(2) the subproblems of enumerating all maximal k-biplexes that contain u_i but exclude the vertices in $\{v\}$ and $\{u_1, ..., u_{i-1}\}$, where $i \in [1, h]$.

(a) Pivoting rule in C (b) Pivoting rule in X

Fig. 3. An illustrative example for the pivoting techniques when $k = 2$, where green vertices represent current k-biplexes, the vertices with gray contours are included in exclusion sets, the red vertices denote the pivot vertices, and the blue regions indicate the vertices that are unnecessary to be selected as the branching vertices.

It is easy to verify that this pivoting technique ignores the problem of enumerating all maximal k-biplexes containing (S_L, S_R) in the bipartite subgraph induced by $S_L \cup C_L \setminus \{v\}$ and $S_R \cup C_R \setminus N_v(G)$. Interestingly, we note that there are no maximal k-biplexes containing (S_L, S_R) in $H = G(S_L \cup C_L \setminus \{v\}, S_R \cup C_R \setminus N_v(G))$, since any maximal k-biplex in H can be expanded by the pivot vertex v. The reasons are as follows. Suppose that (A, B) is a maximal k-biplex containing (S_L, S_R) in H. When we add v into (A, B), only vertices in $B \setminus N_v(G)$ can potentially break the condition of having at most k non-neighbors in $A \cup \{v\}$. However, based on the conditions that $(B \setminus N_v(G)) \subseteq S_R$ and there is no vertex u in S_R with $\overline{d}_u(A \cup \{v\}) \geq k$, we can derive that $(A \cup \{v\}, B)$ is also a k-biplex in G. This implies that there is no need to enumerate all maximal containing (S_L, S_R) in H. Thus, the theorem is established. □

We note that the pivot vertex can also be selected from the exclusion sets (X_L, X_R). Nevertheless, the technique for this case is somewhat more intricate compared to the result described in Theorem 3.3. Hence, we present the detailed theoretical result below.

Theorem 3.4 (Pivoting rule in X). *Let $v \in X_L$ be a pivot vertex. If $\nexists u \in S_R$ with $\overline{d}_u(S_L \cup C_L) > k$, we have the following two cases:*

(1) If $\nexists u \in S_R$ with $\overline{d}_u(S_L \cup C_L \cup \{v\}) > k$, it follows that all vertices in $(C_L, C_R \cap N_v(G))$ are unnecessary to be selected as the branching vertices.
(2) Otherwise, all vertices in $(C_L \setminus Q, C_R \cap N_v(G))$ are unnecessary to be selected as the branching vertices, where the set Q satisfies $\overline{d}_u(S_L \cup C_L \setminus Q) < k$ for each vertex u in $\overline{N}_v(S_R)$.

Proof sketch. By Theorem 3.3, we observe that the algorithm only ignores the enumeration of maximal k-biplexes that contain (S_L, S_R) in subgraph $H_1 = G(S_L \cup C_L, S_R \cup C_R \cap N_v(G))$ for case (1) (or $H_2 = G(S_L \cup C_L \setminus Q, S_R \cup C_R \cap N_v(G))$ for case (2)). When using a similar method as presented in Theorem 3.3, we also derive that each maximal k-biplex (A, B) containing (S_L, S_R) in H_1 for case (1) (or H_2 for case (2)) can also be expanded by the pivot vertex $v \in X_L$. Thus, the vertices in H_1 (or H_2) are unnecessary to be selected as the branching vertices.

The following example further illustrates the concepts presented in Theorem 3.3 and Theorem 3.4.

Example 2. Given a bipartite graph G shown in Fig. 3(a), we assume that $k = 2$. Let the current k-biplex (S_L, S_R) be $(\{u_1\}, \emptyset)$, and the sets $(C_L, C_R) = (\{u_2, u_3, u_4, u_5\}, \{v_1, v_2, v_3, v_4, v_5\})$ be the candidate sets of (S_L, S_R). Suppose that we select v_4 as the pivot vertex for the recursive call that enumerates maximal k-biplexes containing (S_L, S_R). By Theorem 3.3, all vertices in $(\{u_2, u_3, u_4, u_5\}, \{v_1, v_2, v_3, v_5\})$ are unnecessary selected as the branching vertices. Consequently, it is sufficient to expand $(\{u_1\}, \emptyset)$ with v_4 in this recursive call. Similarly, in Fig. 3(b), let us consider the current k-biplex (S_L, S_R) as $(\{u_3\}, \{v_4\})$ and exclusion sets (X_L, X_R) as $(\emptyset, \{v_3\})$. Based on Theorem 3.4, we can deduce that all vertices in the candidate sets are unnecessary as the branching vertices when $u_3 \in X_L$ is selected as the pivot vertex. As a result, such a recursive call can be directly terminated.

Pivot-based branching rule. Equipping with the results of Theorem 3.3 and Theorem 3.4, we can derive a pivot-based branching rule for enumerating maximal k-biplexes on bipartite graphs. Denote by (S_L, S_R) the current k-biplex, (C_L, C_R) the candidate sets used to expand (S_L, S_R), and (X_L, X_R) the exclusion sets containing all vertices in (C_L, C_R) that have been used to expand (S_L, S_R). Then, this branching rule (we only consider the case where the branching vertex (or pivot vertex) is selected from C_R (or $C_R \cup X_R$), as the branching vertex (or pivot vertex) from C_L (or $C_L \cup X_L$) can be obtained by a similar method) is shown as follows.

- Finding a vertex $v \in S_L$ with the smallest degree in $S_R \cup C_R$.
- If $\overline{d}_v(S_R \cup C_R) > k$, we select a vertex u in $\overline{N}_v(C_R)$ as the branching vertex.
- If $\overline{d}_v(S_R \cup C_R) \leq k$, we select a pivot vertex from $C_R \cup X_R$ to construct the branching rules, which involves two cases:
  (1) If the pivot vertex v is selected from C_R, then each vertex in $(\{v\}, C_R \setminus N_v(G))$ is selected as the branching vertex (based on Theorem 3.3).
  (2) If the pivot vertex v is selected from X_R and satisfying $\nexists u \in \overline{N}_v(S_R)$ with $\overline{d}_u(S_L \cup C_L) \geq k$, then each vertex in $C_R \setminus N_v(G)$ is selected as the branching vertex (based on Theorem 3.4).

Note that this branching rule does not take into account the vertices that belong to case (2) of Theorem 3.4 as the pivot vertices. This is because we empirically find that case (2) leads to more computation costs without significantly improving practical performance compared to case (1). Therefore, for the sake of simplicity, we only select a pivot vertex from the candidate sets (C_L, C_R) or from the vertices in (X_L, X_R) belonging to the case (1) of Theorem 3.4 to perform the pivot-based enumeration.

Implementation details. Armed with this pivot-based branching rule, we then develop a new branch-and-bound procedure for enumerating maximal k-biplexes, which is shown in Algorithm 3.

Similar to the *Branch* procedure outlined in Algorithm 1, Algorithm 3 also first selects a vertex from (S_L, S_R) whose degree is minimum in (C_L, C_R) (lines 6-7). If there exists a vertex $v \in S_L$ (or $u \in S_R$) such that the number of non-neighbors in $\overline{d}_v(S_R \cup C_R)$ (or $\overline{d}_u(S_L \cup C_L)$) exceeds k, the basic branching rule for enumerating maximal k-biplexes is employed (lines 8-13). Otherwise, the pivoting technique is used to construct the branching rules. Specifically, the procedure first selects a vertex v (resp. u) from $C_L \cup X_L$ (resp. $C_R \cup X_R$) that has the fewest non-neighbors in C_R (resp. C_L) as the candidate pivot vertex (lines 15-19). If a candidate vertex v (or u) is chosen from X_L (or X_R), every vertex $u' \in \overline{N}_v(S_R)$ (or $v' \in \overline{N}_u(S_L)$) must have at most $k-1$ non-neighbors in $S_L \cup C_L$ (or $S_R \cup C_R$) (lines 17 and 19) based on Theorem 3.4. Then, the procedure selects a better vertex as the pivot vertex to continue enumerations (lines 20-22). Assume that the pivot vertex v is selected from C_L. The procedure subsequently invokes the *BranchP* to construct the pivot-based enumeration process, and then only selects each vertex u in $(\{v\}, C_R \setminus N_v(G))$ as the branching vertex to enumerate the maximal k-biplexes containing (S_L, S_R) and u. Finally, this procedure terminates if both C_L and C_R are empty sets (lines 1-5), and outputs the current k-biplex (S_L, S_R) as the maximal results when X_L and X_R are also the empty sets (lines 3-4).

We prove that such a pivot-based enumeration algorithm (Algorithm 3) has a better time complexity result in enumerating maximal k-biplexes compared to that of Algorithm 1.

Theorem 3.5. *The worst-case time complexity of Algorithm 3 is $O(m\beta_k^n)$ in enumerating maximal k-biplexes, where β_k is the maximum real-root of equation $x^{k+2} - 2x^{k+1} + 1 = 0$ if $k \geq 1$, i.e., when $k = 1, 2,$ and 3, we have $\beta_k = 1.618, 1.839,$ and 1.928, respectively.*

Proof sketch. We note that the algorithm either adopts the basic branching rule or the pivot-based branching rule to enumerate maximal k-biplexes. (1) When using the basic branching rule, we

Algorithm 3: $ImpBranch(S_L, S_R, C_L, C_R, X_L, X_R)$

if $|S_L \cup C_L| < q$ *or* $|S_R \cup C_R| < q$ **then return**;
if $C_L = \emptyset$ *and* $C_R = \emptyset$ **then**
  if $X_L = \emptyset$ *and* $X_R = \emptyset$ **then**
    if $|S_L| \geq lb$ *and* $|S_R| \geq lb$ **then** Output (S_L, S_R);
  return;
$v \leftarrow \arg\max_{v \in S_L} \overline{d}_v(S_R \cup C_R)$; $\overline{d}_L \leftarrow \overline{d}_v(S_R \cup C_R)$;
$u \leftarrow \arg\max_{u \in S_R} \overline{d}_u(S_L \cup C_L)$; $\overline{d}_R \leftarrow \overline{d}_u(S_L \cup C_L)$;
if $\overline{d}_L > k$ **then**
  $w \leftarrow$ a vertex in $\overline{N}_v(C_R)$;
  $BranchB(S_R, w, S_L, C_R, C_L, X_R, X_L)$;
else if $\overline{d}_R > k$ **then**
  $w \leftarrow$ a vertex in $\overline{N}_u(C_L)$;
  $BranchB(S_L, w, S_R, C_L, C_R, X_L, X_R)$;
else
  $v \leftarrow \arg\min_{v \in C_L} \overline{d}_v(C_R)$; $u \leftarrow \arg\min_{u \in C_R} \overline{d}_u(C_L)$;
  foreach $v' \in X_L$ *s.t.* $\overline{d}_{v'}(C_R) < \overline{d}_v(C_R)$ **do**
    if $\nexists u' \in \overline{N}_{v'}(S_R)$ *with* $\overline{d}_{u'}(S_L \cup C_L) \geq k$ **then** $v \leftarrow v'$;
  foreach $u' \in X_R$ *s.t.* $\overline{d}_{u'}(C_L) < \overline{d}_u(C_L)$ **do**
    if $\nexists v' \in \overline{N}_{u'}(S_L)$ *with* $\overline{d}_{v'}(S_R \cup C_R) \geq k$ **then** $u \leftarrow u'$;
  if $\overline{d}_v(C_R) \leq \overline{d}_u(C_L)$ **then**
    $BranchP(S_L, v, S_R, C_L, C_R, X_L, X_R)$;
  else $BranchP(S_R, u, S_L, C_R, C_L, X_R, X_L)$;
Function: $BranchP(S_L, v, S_R, C_L, C_R, X_L, X_R)$
  foreach $u \in C_R \setminus N_v(G)$ **do**
    $(C'_L, C'_R) \leftarrow Updates(S_R, u, S_L, C_R, C_L)$;
    $(X'_L, X'_R) \leftarrow Updates(S_R, u, S_L, X_R, X_L)$;
    $ImpBranch(S_L, S_R \cup \{u\}, C'_L, C'_R, X'_L, X'_R)$;
    $C_R \leftarrow C_R \setminus \{u\}$; $X_R \leftarrow X_R \cup \{u\}$;
  if $v \notin X_L$ **then**
    $(C'_L, C'_R) \leftarrow Updates(S_L, v, S_R, C_L, C_R)$;
    $(X'_L, X'_R) \leftarrow Updates(S_L, v, S_R, X_L, X_R)$;
    $ImpBranch(S_L \cup \{v\}, S_R, C'_L, C'_R, X'_L, X'_R)$;

can obtain that $T(n) \leq \sum_{i=1}^{k+1} T(n-i)$ as proved in Theorem 3.1. (2) When using the pivot-based branching rule, we can derive that $T(n) \leq \sum_{i=1}^{\overline{d}+1} T(n-i)$, where $\overline{d}$ is the number of non-neighbors of the pivot vertex v in the search space. We note that the subbranch for enumerating maximal k-biplexes containing $(S_L \cup \{v\}, S_R)$ would make use of the basic branching rule to continue enumerations if $\overline{d} > k$. Based on this, we can obtain that in the worst case, the recurrence of case (2) is bounded by $T(n) \leq \sum_{i=1}^{k+1} T(n-i)$. Thus, this theorem holds when adopting the theoretical result in Theorem 3.1.

Theorem 3.6. *For the special case of $k = 0$, the time complexity of Algorithm 3 is $O(m1.414^n)$.*

Proof sketch. In this case, we note that no basic branching rule is used in Algorithm 3 (lines 6-13), Then, a recurrence of $T(n) \leq T(n-\overline{d}-1)+\sum_{i=1}^{\overline{d}} T(n-\overline{d}-i)$ can be obtained, where $\overline{d}$ is the number of non-neighbors of the pivot vertex v in the search space. Based on the theoretical results presented in [16], when $k = 1$, $T(n)$ has the maximum size. Thus, we have $T(n) \leq T(n-2) + T(n-2) = 2T(n-2) = 2^{n/2}T(0) \approx 1.414^n T(0)$.

Remark. Although Algorithm 1 is found to underperform both theoretically and practically compared to Algorithm 3, it plays a crucial role in the development of our improved algorithm. Specifically, the developed pivot-based branching rule in Algorithm 3 builds on the branching rule established in Algorithm 1. Furthermore, the non-trivial worst-case time complexity achieved by Algorithm 1 serves as the foundation for demonstrating that Algorithm 3 can achieve an even better time complexity. These close relationships highlight the necessity of the proposed Algorithm 1 in this paper.

Discussions. We observe that several recent works in [16, 47, 51] are also applicable for solving our problem. However, to our knowledge, the techniques developed in Algorithm 3 are very different from all these existing techniques. Firstly, a BPEA algorithm developed in [16] also presents a pivoting technique for enumerating maximal k-biplexes. As analyzed in [16], BPEA achieves a time complexity of $(m1.414^n)$ only when $k = 0$, which sames as that of our algorithm, since both BPEA and Algorithm 3 correspond to enumerate maximal bicliques for the case where $k = 0$. However, when $k \geq 1$, the time complexity of this approach is still $O(m2^n)$, which is significantly higher than that of our algorithm. The reasons for this difference are as follows. On the one hand, when using the technique in [16] to expand (A, B) with a vertex u from (C_L, C_R), it is unclear whether u possesses k non-neighbors in B (and A). Consequently, the search space (size of (C_L, C_R)) will decrease by at least 1 when u is added to (A, B). On the other hand, the pivoting technique in [16] requires considering the vertices in $(P \cup \{u\}, C_R \setminus N_u(G))$ to expand the current k-biplex (A, B), where P is the subset of C_L, which can obtain a recurrence of $Z(n) \leq \sum_{i=1}^{\overline{d}+1+|P|} Z(n-i)$. Since the size of P is also unclear, we can only obtain that $\overline{d} + |P| < n$, indicating that the size of $Z(n)$ is bounded by $O(2^n)$. Thus, BPEA achieves a higher time complexity over Algorithm 3 when $k \geq 1$.

Secondly, we note that an approach for enumerating maximal k-plexes proposed in [51] can be adapted to address our problem. In this approach, a vertex v from (A, B) (or (C_L, C_R)) with the smallest degree is selected, and subsequently $k+1$ (or $k+2$) subbranches is constructed to enumerate maximal k-biplexes based on the vertex v. It can be observed that this approach also achieves the same time complexity as Algorithm 1 [51]. However, the branching rule employed in Algorithm 1 differs significantly from the approach in [51], since Algorithm 1 only involves at most 2 subbranches in each recursive call. Furthermore, the time complexity analysis of Algorithm 1 also varies from that in [51]. In Theorem 3.1, we analyze that the recurrence inequation of Algorithm 1 is implicit and requires detecting recursive calls up to a depth of k, whereas the recurrence inequation in [51] is more straightforward, with a maximum of $k+2$ subbranches for each recursive call. Nevertheless, we further improve the branching rule in Algorithm 1 and develop an Algorithm 3 to achieves better efficiency in both theoretical results and practical performance, making it challenging to achieve the same level of efficiency as Algorithm 3 for the approach in [51].

Lastly, the approach proposed in [47] for finding the maximum k-biplex can also be slightly modified to enumerate maximal k-biplexes, while maintaining a time complexity less than $O(m2^n)$. It is important to note that the enumeration of maximal k-biplexes poses greater challenges compared to finding a maximum k-biplex. However, we emphasize that Algorithm 3 surpasses the aforementioned approach in terms of time complexity. Thus, even if this approach is adapted

to enumerate maximal k-biplexes, its theoretical efficiency clearly falls short of Algorithm 3. Furthermore, as discussed in [16], this approach achieves the similar time complexity to Algorithm 1, as their branching rules exhibit similarities with [51]. In fact, the literature [16] leverages an adaptation from [47] to compare its performance against BPEA. The experiments demonstrate that the approach developed in [47] exhibits significantly lower efficiency compared to BPEA, further validating the superiority of proposed Algorithm 3.

4 OPTIMIZATION TECHNIQUES

In this section, we develop several techniques, including graph reduction, novel upper-bound based pruning, and ordering-based optimization, to further improve the efficiency of the proposed algorithms. Below, we first introduce the graph reduction techniques.

4.1 Graph Reduction Techniques

Here we introduce several key observations to reduce unnecessary vertices of the input graph, i.e., removing the vertices that are definitely not in any maximal k-biplexes with the sizes of both sides no less than q.

Core-based reduction. A simple and efficient technique to reduce the input graph is based on a (l, r)-core concept [9], which is formally defined as follows.

Definition 2 ((l, r)-core). Given a bipartite graph G, a subgraph $G(A, B)$ of G is an (l, r)-core if $d_v(B) \geq r$ for every $v \in A$ and $d_u(A) \geq l$ for every $u \in B$.

In this paper, we call an (l, r)-core $G(A, B)$ maximal in G if there is no other (l, r)-core $G(A', B')$ in G with $A \subseteq A'$ and $B \subseteq B'$. The following lemma then widely used to improve the efficiency for the enumeration of relative-large maximal k-biplexes [16, 49].

Lemma 2 ([49]). Any maximal k-biplex (A, B) of G must be contained in the maximal $(q-k, q-k)$-core of G if $|A| \geq q$ and $|B| \geq q$.

Based on Lemma 2, every vertex excluded in the $(q-k, q-k)$-core of G can be directly removed from the original bipartite graph, which can significantly reduce the size of the input graph and thus improve the efficiency of enumeration algorithms. To obtain the $(q-k, q-k)$-core of bipartite graph G, the algorithms proposed in [4, 34] can be applied, which take only $O(n+m)$ time.

Butterfly-based reduction. To further reduce the graph size, we propose a new butterfly-based graph reduction technique. Here a butterfly is defined as a 2×2 biclique, which consists of four vertices $\{v_1, v_2, u_1, u_2\}$ such that $v_1, v_2 \in L$, $u_1, u_2 \in R$, and the edges (v_1, u_1), (v_1, u_2), (v_2, u_1), and (v_2, u_2) all exist in E. Below, we first give a useful lemma.

Lemma 3. Given a k-biplex (S_L, S_R), the number of common neighbors for each $v_i, v_j \in S_L$ (resp. $u_i, u_j \in S_R$) is no less than $|S_R| - 2k$ (resp. $|S_L| - 2k$).

Proof. Let $\mathsf{cn}_{S_R}(v_i, v_j)$ be the number of common neighbors of v_i and v_j in S_R. We can derive that $\mathsf{cn}_{S_R}(v_i, v_j) \geq |S_R| - \overline{d}_{v_i}(S_R) - \overline{d}_{v_j}(S_R)$. For a k-biplex (S_L, S_R), we have $\overline{d}_{v_i}(S_R) \leq k$ and $\overline{d}_{v_j}(S_R) \leq k$, where $v_i, v_j \in S_L$, implying $\mathsf{cn}_{S_R}(v_i, v_j) \geq |S_R| - 2k$. This inequality also holds when $u_i, u_j \in S_R$. Thus, we establish this lemma. □

Denote by $\mathsf{bfy}_G(e)$ the number of butterflies in G that contains the edge e. Then, we can derive the lower bound of $\mathsf{bfy}_H(e)$ for each e in a k-biplex $H = G(A, B, E_H)$, which is presented in the following lemma.

Lemma 4. Given a k-biplex $H = G(A, B, E_H)$ with $\min\{|A|, |B|\} \geq q$, for each edge $e \in E_H$, we have $\mathsf{bfy}_H(e) \geq (q-k-1)(q-2k-1)$.

Algorithm 4: Butterfly-based reduction

Input: Bipartite graph $G = (L, R, E)$ and the size threshold q
Output: A pruned bipartite graph

1 **for** $e = (u, v) \in E$ **do**
2 $\text{bfy}_G(e) \leftarrow 0$;
3 **for** $u' \in N_v(G)$ *s.t.* $u' \neq u$ **do**
4 $\text{bfy}_G(e) \leftarrow \text{bfy}_G(e) + |N_u(G) \cap N_{u'}(G)| - 1$;
5 **while** $\exists e = (u, v) \in E$ *with* $\text{bfy}_G(e) < (q - k - 1)(q - 2k - 1)$ **do**
6 Remove $e = (u, v)$ from G;
7 **for** $u' \in N_v(G)$ *s.t.* $u' \neq u$ **do**
8 $e' = (v, u')$;
9 $\text{bfy}_G(e') \leftarrow \text{bfy}_G(e') - |N_u(G) \cap N_{u'}(G)| + 1$;
10 **for** $v' \in N_u(G) \cap N_{u'}(G)$ **do**
11 $e_1 = (u, v')$; $e_2 = (u', v')$;
12 $\text{bfy}_G(e_1) \leftarrow \text{bfy}_G(e_1) - 1$; $\text{bfy}_G(e_2) \leftarrow \text{bfy}_G(e_2) - 1$;
13 **if** $d_v(G) < q - k$ **then**
14 **for** $u' \in N_v(G)$ *s.t.* $u' \neq u$ **do** $\text{bfy}_G(e' = (v, u')) = 0$;
15 **if** $d_u(G) < q - k$ **then**
16 **for** $v' \in N_u(G)$ *s.t.* $v' \neq v$ **do** $\text{bfy}_G(e' = (v', u)) = 0$;
17 **return** G;

Proof. Given an edge $e = (u, v) \in H_E$ with $u \in A$ and $v \in B$, if $y \in N_u(H)$ and $x \in N_v(H) \cap N_y(H)$, we obtain that $\{u, x, v, y\}$ must be a butterfly in H. With this observation, we can compute $\text{bfy}_H(e)$ as follows: $\text{bfy}_H(e) = \sum_{y \in N_u(H), y \neq v}(|N_v(H) \cap N_y(H)| - 1)$. From Lemma 3, it can be deduced that $|N_v(H) \cap N_y(H)| \geq q - 2k$ for every $y \in B$ if $\min\{|A|, |B|\} \geq q$. Moreover, based on the fact that $|N_u(H)| \geq q - k$, we can conclude that $\text{bfy}_H(e) \geq (q - k - 1)(q - 2k - 1)$. Thus, the lemma is established. □

With Lemma 4, we can safely remove all unnecessary edges of the bipartite graph G when enumerating all maximal k-biplexes with sizes of both sides no less than q. To achieve this, we first compute $\text{bfy}_G(e)$ for each e in G. Then, we iteratively remove all edges with the number of butterflies less than $(q - k - 1)(q - 2k - 1)$. The detailed pseudo-code is presented in Algorithm 4.

In Algorithm 4, it first computes the number of butterflies containing each edge e in G (lines 1-4). Specifically, given an edge $e = (u, v) \in E$, the value of $\text{bfy}_G(e)$ in G can be computed using the formula $\text{bfy}_G(e) = \sum_{y \in N_u(G), y \neq v}(|N_v(G) \cap N_y(G)| - 1)$, since $\{u, x, v, y\}$ must be a butterfly in G if $y \in N_u(G)$ and $x \in N_v(G) \cap N_y(G)$. Then, the algorithm iteratively removes each edge e from E if $\text{bfy}_G(e) < (q - k - 1)(q - 2k - 1)$ (lines 5-12). Note that when an edge $e = (u, v)$ is removed from E, the butterflies containing the edge e will be invalidated. Consequently, the $\text{bfy}_G(e)$ values of the remaining edges on these butterflies must be updated as well (lines 7-12). Finally, the algorithm terminates when the value of $\text{bfy}_G(e)$ for each remaining edge e is no less than $(q-k-1)(q-2k-1)$, at which point the subgraph induced by these remaining edges is returned. The following shows the time complexity of Algorithm 4.

Theorem 4.1. *The time complexity of Algorithm 4 is $O(md_{max}^2)$, where d_{max} is the largest degree of vertices in G.*

Proof. For each edge e, it takes at most $O(d_{max}^2)$ time to compute $\text{bfy}_G(e)$, so the total time for the initial phase of the algorithm is bounded by $O(md_{max}^2)$ (lines 1-4). Moreover, when removing an edge e, the value $\text{bfy}_G(e)$ of the remaining edges on the butterflies that contains e must be updated (lines 7-12), which also takes $O(d_{max}^2)$ time. Thus, the overall time complexity of Algorithm 4 is bounded by $O(md_{max}^2)$. □

Although the worst-case time complexity of Algorithm 4 is $O(md_{\max}^2)$, the practical performance of Algorithm 4 is very efficient as confirmed in our experiments. The reasons are as follows. First, the complexity for most edges often cannot reach $O(d_{\max}^2)$, so the practical cost of Algorithm 4 is typically much lower than the worst-case time complexity. Second, we can first use the core-based technique to reduce the input graph, and then apply the butterfly-based reduction technique to the subgraph obtained by the core-based reduction. This approach takes advantage of both techniques to improve the practical performance of Algorithm 4.

Discussion. It is worth noting that the literature [13] employed coreness and cliqueness pruning techniques for enumerating large maximal k-plexes. We observe that the coreness pruning proposed in [13] can be directly extended for enumerating large k-biplexes, aligning with the core-based graph reduction presented in Lemma 2. However, our proposed butterfly-based pruning technique significantly differs from the cliqueness pruning technique. Specifically, the idea of our proposed butterfly-based technique is derived from a result of Lemma 3. By leveraging this result, we establish that each edge in a bipartite graph must be associated with at least $(q-k-1)(q-2k-1)$ butterflies if it is included in a k-biplex with both sides having a size no less than q. In contrast, the cliqueness pruning in [13] relies on the existence of a clique of size $\lceil q/k \rceil$ in a k-plex of size q, which poses challenges to extend it to enumerate maximal k-biplexes. Additionally, computing cliques of size $\lceil q/k \rceil$ is much more difficult than computing butterflies, which makes the cliqueness pruning technique in [13] inefficient in removing unnecessary vertices from bipartite graphs.

4.2 Upper-Bound Techniques

If we can determine an upper bound on the size of one side of the maximal k-biplexes containing (A, B) smaller than a threshold q, it is obvious to obtain that all maximal k-biplexes that contain (A, B) are unnecessary to be enumerated. With this idea, we present a new upper-bound technique.

Let $\text{ub}_G^k(A, B)$ be the upper bound on the size of the maximal k-biplex in G that contains (A, B). In other words, $\text{ub}_G^k(A, B)$ satisfies the inequality $\min\{|X|, |Y|\} \leq \text{ub}_G^k(A, B)$, where (X, Y) is a random maximal k-biplex of G that contains (A, B). Then, we can easily derive the following lemma.

Lemma 5. Let (A, B) be a k-biplex of G, then we have $\text{ub}_G^k(A, B) \leq \min_{u \in A}\{d_u(G) + k - \overline{d}_u(B)\}$.

Such a basic upper-bound technique is not very tight, which may result in poor pruning performance. To mitigate this scenario, we develop a tighter upper bound. Given a k-biplex (A, B), let $T_B^r(A) \subseteq B$ be r vertices in B that have most number of non-neighbors in A, i.e., each vertex v_i in $T_B^r(A)$ satisfies $\overline{d}_{v_i}(A) \geq \overline{d}_{v_j}(A)$, where $v_j \in B \setminus T_B^r(A)$ and $|T_B^r(A)| = r$. Denote by $nb_A(B) = \sum_{v \in A} \overline{d}_v(B)$ the sum of non-neighbor counts in B for each vertex in A. Suppose that the vertex $u \in L \setminus A$ is used to add to (A, B), and $D = \{v \in N_u(G) \setminus B | (A \cup \{u\}, B \cup \{v\})$ is a k-biplex$\}$ is the set of neighbors of u that can be used to expand $(A \cup \{u\}, B)$. Then, the next lemma shows a tighter upper bound.

Lemma 6. Denote by $I \subseteq A$ the set of vertices that have more than k non-neighbors in $B \cup D$ i.e., $I = \{v \in A | \overline{d}_v(B \cup D) > k\}$. Let r be the minimum non-negative integer satisfying $\sum_{v \in T_D^r(A)} \overline{d}_v(A) \geq nb_I(B \cup D) - |I| \times k$, then we have $\text{ub}_G^k(A \cup \{u\}, B) \leq |B| + |D| + k - \overline{d}_u(B) - r$.

Algorithm 5: $UpperBound(A, B, u)$

```
1  s ← 0;
2  D ← {v ∈ N_u(G) \ B | (A ∪ {u}, B ∪ {v}) is a k-biplex};
3  for w ∈ A do
4      if d̄_w(B ∪ D) > k then s ← s + d̄_w(B ∪ D) − k;
5  ub ← |B| + |D| + k − d̄_u(B);
6  while s > 0 do
7      v ← arg max_{w∈D} d̄_w(A);
8      D ← D \ {v};
9      s ← s − d̄_v(A);
10     ub ← ub − 1;
11 return ub;
```

Proof. It is easy to see that $\mathrm{ub}_G^k(A \cup \{u\}, B) \leq |B| + |D| + k - \overline{d}_u(B)$, as there are at most $|D| + k - \overline{d}_u(B)$ vertices that can be added to $(A \cup \{u\}, B)$. Then, it is important to note that all vertices in $I \subseteq A$ violate the definition of k-biplex, and it is necessary to remove some vertices from D. In the worst-case, we assume that each vertex in I is the non-neighbor of $v \in T_D^r(A)$. Consequently, r vertices must be removed from D to ensure that every vertex in I satisfies the definition of k-biplex. □

The following example illustrates the idea of Lemma 6.

Example 3. Consider a bipartite graph G shown in Fig. 1, and suppose that $(A, B) = (\{u_1\}, \{v_2\})$ forms a k-biplex with vertex $v_6 \in R$ used to add to (A, B). When $k = 1$, we have $D = \{u_2, u_3, u_5\} \subseteq N_{v_6}(G)$ and $I = \{v_2\} \subseteq B$. By Lemma 6, we can compute that $\sum_{v \in T_D^1(B)} \overline{d}_v(B) = 1 \geq nb_I(A \cup D) - |I| \times k = 2 - 1 = 1$, which implies $r \geq 1$. Hence, $\mathrm{ub}_G^k(A, B \cup \{v_6\}) \leq |A| + |D| + k - \overline{d}_{v_6}(A) - r = 3$. However, using Lemma 5 yields a looser bound of $\mathrm{ub}_G^k(A, B \cup \{v_6\}) \leq 4$. This example further illustrates that Lemma 6 provides a tighter bound than Lemma 5.

Based on Lemma 6, we propose an algorithm to compute $\mathrm{ub}_G^k(A \cup \{u\}, B)$, as shown in Algorithm 5.

In Algorithm 5, it admits three parameters: A, B, and u, where (A, B) denotes a current k-biplex, and $u \in L \setminus A$ is the vertex used to add to (A, B). The algorithm begins by obtaining the set $D \subseteq N_u(G)$ of vertices that can be added to $(A \cup \{u\}, B)$ to form a larger k-biplex (line 2). Then, the algorithm finds the vertices in A that violate the definition of the k-biplex and computes the sum of non-neighbor counts s in $B \cup D$ for these violating vertices (lines 2-3). Subsequently, the algorithm iteratively removes each vertex v in D that has the highest non-neighbor counts in A, and uses the value of $\overline{d}_v(A)$ to decrease the total non-neighbor counts s (lines 6-10), as it is possible that all vertices in $\overline{N}_v(A)$ violate the definition of the k-biplex in $G(A \cup \{u\}, B \cup D)$. Finally, the algorithm terminates when $s \leq 0$, and returns the number of remaining vertices in D as the upper bound (line 5 and line 10).

Theorem 4.2. *The time complexity of Algorithm 5 is $O(kd_{max})$ if the set $D \subseteq N_v(G)$ is given, where d_{max} is the maximum degree of the bipartite graph.*

Proof. If we use an array to store the non-neighbors in A of each vertex in D, it is straightforward to see that lines 3-4 take at most $O(kd_{max})$ time and line 7 takes at most $O(k)$ time. Thus, the time complexity of Algorithm 5 is bounded by $O(kd_{max})$. □

Algorithm 6: The improved branch-and-bound algorithm

Input: Bipartite graph $G = (L, R, E)$, two parameters k and q
Output: All relatively-large maximal k-biplexes of G

1 Reduce the bipartite graph G with core-based reduction;
2 Further reduce G with Algorithm 4, and let $H = (L_H, R_H, E_H)$ be the subgraph of G induced by remaining vertices and edges;
3 $V \leftarrow \emptyset$;
4 $d_{Lmax} \leftarrow \max_{v \in L_H} d_v(H)$; $d_{Rmax} \leftarrow \max_{u \in R_H} d_u(H)$;
5 **if** $d_{Lmax} \geq d_{Rmax}$ **then** $V \leftarrow L_H$;
6 **else** $V \leftarrow R_H$;
7 $O \leftarrow$ the ordering of vertices in H;
8 **foreach** $v_i \in O$ *s.t.* $v_i \in V$ **do**
9 Let $H_{v_i} = (L_{H_{v_i}}, R_{H_{v_i}}, E_{H_{v_i}})$ be the subgraph $G_{v_i}^{>}$;
10 $C_L \leftarrow L_{H_{v_i}} \setminus \{v_i\}$; $C_R \leftarrow R_{H_{v_i}}$;
11 $X_L \leftarrow \{v_j \in O | v_i \succ v_j, N_{v_i} \cap N_{v_j} \neq \emptyset\}$; $X_R \leftarrow \emptyset$;
12 Reduce the size of C_L, C_R, X_L, X_R with Lemma 3;
13 $ImpBranch(\{v_i\}, \emptyset, C_L, C_R, X_L, X_R)$; `/* Employed the upper-bound techniques presented in Sec. 4.2 */`

4.3 Ordering-Based Optimization

Inspired by existing solutions [16], an ordering technique can be employed to improve the efficiency of enumerating maximal k-biplexes. The detailed technique is introduced below.

Consider an ordering $O = \{v_1, v_2, \ldots, v_n\}$ of vertices in $L \cup R$. We define $v_i \succ v_j$ to indicate that the vertex v_i is ranked after v_j in O (i.e., $v_i \in \{v_j, v_{j+1}, \ldots, ..., v_n\}$). Denote by $N_v^{>2}(G)$ the set of vertices in G that are at a distance of 2 from v, and ranked after v in O, i.e., $N_v^{>2}(G) = \{v' \in O | v' \succ v, N_{v'}(G) \cap N_v(G) \neq \emptyset\}$. Denote by $\Gamma_v^{>2}(G) = N_v^{>2}(G) \cup \{v\}$. We define the subgraph $G_v^{>}$ of G as the bipartite graph induced by the vertices in $\Gamma_v^{>2}(G)$ and their neighbors ($\cup_{u \in \Gamma_v^{>2}(G)} N_u(G)$). Given a maximal k-biplex (A, B) of G with the sizes of both sides no less than $q \geq 2k + 1$, let v_i be the rank smallest vertex in A according to the ordering O. We observe that (A, B) must be contained in the subgraph $G_{v_i}^{>}$, where $v_i \in L$ with the condition of $A \subseteq L$. Therefore, we can devise an ordering-based enumeration algorithm to identify the maximal k-biplexes of G, i.e., enumerating all maximal k-biplexes in $G_{v_i}^{>}$ for each $v_i \in O \cap L$.

In this paper, we adopt the widely used degeneracy ordering [29, 30] as the vertex ordering scheme, which is obtained by treating the bipartite graphs as the traditional graph and then applying the peeling technique [4, 29]. Moreover, we note that selecting vertices from one side of the bipartite graph to construct the subgraph $G_{v_i}^{>}$ is sufficient for enumerating all maximal k-biplexes. Based on our empirical studies, we find that in most real-world graphs, choosing the side with the larger maximum degree yields better performance. This observation may attributed to the fact that such a selection method leads to a smaller subgraph $G_{v_i}^{>}$, thus improving the efficiency of enumeration process.

The implementation details. Armed with the optimization techniques presented in this section, we develop an improved algorithm for enumerating maximal k-biplexes, which is presented in Algorithm 6. Initially, this algorithm makes use of the core-based reduction and the butterfly-based reduction (lines 1-2). Then, the algorithm selects one side of the vertices, where the side has a larger maximum degree compared to the other side, to perform the ordering-based enumerations in the subgraph induced by the remaining edges. When it comes to enumerating the maximal k-biplexes

Table 1. Real-world bipartite graph datasets.

Datasets	$\|L\|$	$\|R\|$	$\|E\|$	$d_{1\max}$	$d_{2\max}$
YouTube	94,238	30,087	293,360	1,035	7,591
IMDB	303,617	896,302	3,782,463	1,334	1,590
Amazon	2,146,057	1,230,915	5,743,258	12,217	3,146
Wikisource	18,038	2,192,849	6,561,379	916,505	30,606
Twitter	244,537	9,129,669	12,656,613	855	24,106
Google	5,998,790	4,443,631	20,592,962	423	95,165

of G, the algorithm simply makes use of the recursive procedure *ImpBranch* to enumerate maximal k-biplexes containing v_i in subgraph $G^{>}_{v_i}$ for each $v_i \in V$ according to the ordering O (lines 8-13). Note that in each recursive call of *ImpBranch*, the upper-bound techniques presented in Sec. 4.2 are also used for improving efficiency. After each vertex v_i in V has been processed by the recursive calls, this algorithm finished.

5 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the efficiency of our algorithms. Below, we first introduce the experimental setup and then present the results.

5.1 Experimental Setup

Algorithms. We implement two algorithms, namely BPBnB and BPPivot, for enumerating maximal k-biplexes on bipartite graphs, where BPBnB is the Algorithm 1 and BPPivot is the Algorithm 3 equipped with pivot-based branching rules. To ensure fairness in the experiments, both algorithms BPBnB and BPPivot are also augmented with all optimization techniques presented in Sec. 4, including the graph reductions, upper-bound techniques, and ordering-based optimization (Algorithm 6).

To evaluate the performance of our proposed algorithms, we also conduct comparative evaluations against the state-of-the-art algorithm called BPEA, as introduced in [16]. As highlighted in [16], all other existing solutions, including the maximal k-biplex enumeration algorithms developed in [38, 48, 49] and an adaptation in [47], exhibit significantly lower performance compared to BPEA. Therefore, in this paper, we exclusively make use of BPEA as the baseline algorithm for evaluating the efficiency of our proposed algorithms. All the algorithms are implemented in C++, and tested on a PC with one 2.2 GHz CPU and 64GB memory running CentOS operating system. Note that the runtime of all algorithms (including our algorithms and the state-of-the-art BPEA) displayed in our experiments includes both time spend on graph reductions and the enumeration process.

Datesets. We use 6 real-world bipartite graphs to evaluate the efficiency of different algorithms. Most of these bipartite graphs are widely used as benchmark datasets to evaluate the performance of various k-biplex enumeration algorithms [16, 48, 49]. The detailed statistics of each dataset are shown in Table 1, where the columns d_{1max} and d_{2max} denote the maximum degree of vertices in L and R, respectively. All datasets can be downloaded from http://konect.cc.

Parameters. As the k-biplexes with small sizes are often sparse and not of significant practical value [16, 49], we focus only on enumerating the maximal k-biplexes with sizes no less than $q \geq 2k + 1$ in our experiments. We vary the value of k from 1 to 6 with a default setting of 2. Due to the extensive number of small-sized k-biplexes in each dataset, it becomes impractical for all algorithms to complete the processing within the 24 hour time constraint. Consequently, we select q values ranging between 10 and 20 for all tested datasets if $k \leq 2$. For $k \geq 3$, we employ an adaptive approach to select the values of q for each dataset.

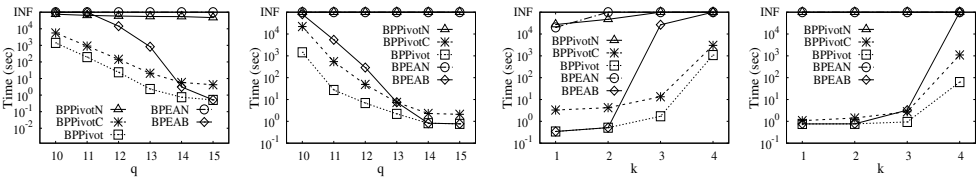
Table 2. Runtime of different algorithms with varying k and q on real-world bipartite graphs (in seconds).

Datasets	k	q	#Nums	BPPivot	BPBnB	BPEA	k	q	#Nums	BPPivot	BPBnB	BPEA	k	q	#Nums	BPPivot	BPBnB	BPEA
YouTube	1	10	1.60×10^7	**58.59**	—	136.65	3	15	3.15×10^9	**38381**	—	—	5	20	2.54×10^8	**11419**	20259	—
		14	5.92×10^2	**0.14**	0.61	1.76		17	2.17×10^6	**56.02**	202.64	43221		22	2.12×10^2	1.12	**0.43**	—
	2	12	1.27×10^9	**7847.2**	—	40265	4	18	3.30×10^8	**8020.6**	31323	—	6	22	6.40×10^7	**6698.1**	8171.3	—
		16	8.95×10^3	**0.65**	1.03	160.71		20	1.80×10^3	2.22	**1.29**	50024		24	0.17×10^2	**0.24**	0.91	—
IMDB	1	15	2.03×10^6	**6.42**	63362	30.05	3	20	1.17×10^8	**1292.5**	70292	61698	5	24	4.02×10^8	**8814.25**	—	—
		19	24	**0.28**	0.28	0.41		22	3.38×10^4	**1.31**	6.27	768.44		26	3.8×10^3	**0.44**	0.79	1659.9
	2	15	2.06×10^9	**9363.1**	—	—	4	22	3.69×10^8	**5955.2**	—	—	6	26	1.59×10^8	**4336.8**	17483	—
		20	1.66×10^4	**0.53**	1.65	79.80		24	1.67×10^4	**1.11**	2.49	2072.2		28	205	**0.21**	0.28	37.23
Wikisource	1	15	3.08×10^6	**16.04**	78023	37.50	3	19	6.32×10^7	**3441.7**	67208	—	5	23	2.02×10^3	**5864.1**	—	—
		20	0	**0.09**	0.09	0.17		21	0.74×10^2	**0.68**	3.74	18325		25	0	**0.21**	0.52	—
	2	17	9.75×10^7	**950.81**	—	20346	4	21	2.58×10^5	**6896.7**	—	—	6	25	0.40×10^2	**5779.9**	—	—
		20	8	**0.16**	0.18	49.11		22	2.37×10^2	**54.27**	1325.4	—		26	0	**49.17**	338.64	—
Amazon	1	10	9.63×10^4	**3.32**	—	626.8	3	12	5.45×10^8	**14719**	—	—	5	16	5.58×10^8	**30172**	48029	—
		14	0.43×10^2	**0.47**	0.52	1.68		15	4.93×10^4	**1.71**	4565.1	—		18	1.33×10^4	**2.82**	4.50	—
	2	10	6.04×10^7	**1403.3**	—	—	4	14	1.27×10^9	**31501**	—	—	6	18	3.01×10^7	21375	**9109.1**	—
		14	5.53×10^3	**0.74**	16.55	5283		16	7.57×10^6	**18.62**	—	—		19	1.37×10^4	120.24	**81.76**	—
Twitter	1	16	1.47×10^9	**2211.1**	—	—	3	22	1.63×10^7	**116.64**	—	23062	5	26	6.63×10^6	**156.16**	—	2304.8
		20	1.62×10^5	**0.38**	—	0.78		25	4.45×10^4	**1.63**	—	69.60		30	4.93×10^3	**0.68**	—	147.94
	2	18	6.23×10^8	**4315.5**	—	—	4	23	2.01×10^9	**50997**	—	—	6	27	1.03×10^7	**139.46**	—	2365.4
		20	2.67×10^6	**18.99**	—	1062.3		25	1.02×10^6	**8.66**	—	400.36		30	8.47×10^3	**1.447**	—	476.33
Google	1	10	4.01×10^3	**4.31**	38.89	12.01	3	12	5.50×10^8	**11177**	—	—	5	16	1.51×10^7	**11881**	—	—
		13	0	**0.83**	1.68	1.06		14	3.83×10^3	**1.53**	7.97	66231		17	4.07×10^2	**44.37**	139.03	—
	2	10	1.49×10^8	**1458.0**	—	—	4	14	3.21×10^8	**15561**	—	—	6	18	4.24×10^3	**13839**	—	—
		13	1.13×10^2	**0.94**	4.29	113.61		15	2.73×10^5	**63.0**	822.29	—		19	0	**10.80**	17.41	—

5.2 Experimental Results

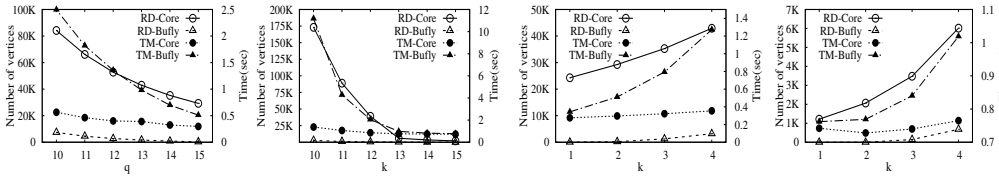
Exp-1: Runtime of various algorithms. In this experiment, we evaluate the performance of various algorithms in terms of their runtime for enumerating maximal k-biplexes. Table. 2 shows results obtained by varying the values of q and k on all tested datasets, where the label "—" indicates cases where the algorithm can not complete the computations within 24 hours, #Nums denotes the number of maximal k-biplexes with the sizes of both sides no less than q. From Table. 2, the runtime of all algorithms increases dramatically with decreasing values of q or increasing values of k. This behavior is attributable to the fact that the number of maximal k-biplexes is highly sensitive to the parameters q and k. For example, on the Google dataset, when $k = 2$ and $q = 13$, there are only 113 maximal k-biplexes. However, when $q = 10$, the number of maximal k-biplexes increases drastically to 1.49×10^8. Nevertheless, our improved algorithm BPPivot consistently outperforms the state-of-the-art solution BPEA across all parameter settings. More specifically, our algorithm BPPivot achieves at least three orders of magnitude faster computation times than BPEA on most testing datasets. For instance, on the Amazon dataset with $k = 3$ and $q = 15$, our algorithm BPPivot takes a mere 1.71 seconds to finish the computations, while the state-of-the-art algorithm BPEA cannot terminate within 24 hours. The results showcase demonstrate the high efficiency of the proposed techniques in enumerating maximal k-biplexes, even in scenarios with large values of k (k grows to 6). In addition, when comparing with BPPivot and BPEA, we can observe that the runtime of BPBnB is usually higher than that of BPPivot. However, it can still be significantly faster than BPEA across most parameter settings. These results further confirm the efficiency of the proposed pivoting technique and branching rules presented in Section 3.2.

Exp-2: Results of algorithms with or without graph reduction techniques. To evaluate the effectiveness of graph reduction techniques, we also implement four algorithms, BPPivotN, BPPivotC, BPEAN, and BPEAB, for enumerating maximal k-biplexes. Here BPPivotN and BPEAN refer to algorithms BPPivot and BPEA without any graph reductions, respectively. BPPivotC and BPEAB denote the algorithm BPPivot equipped with the core reduction only and the algorithm BPEA equipped with our proposed butterfly-based reduction, respectively. Fig. 4 shows the runtime



(a) Amazon ($k = 2$) (b) Google ($k = 2$) (c) Amazon ($q = 15$) (d) Google ($q = 15$)

Fig. 4. Effects of graph reduction techniques.

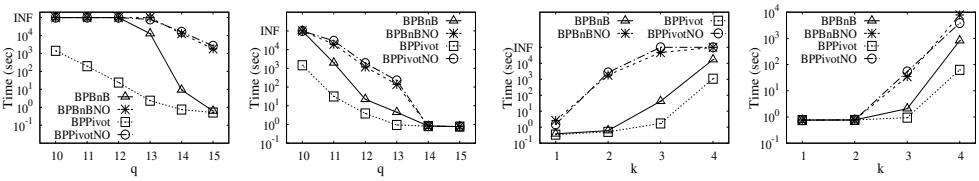


(a) Amazon ($k = 2$) (b) Google ($k = 2$) (c) Amazon ($q = 15$) (d) Google ($q = 15$)

Fig. 5. Number of remaining vertices and time overhead incurred when employing graph reduction techniques.

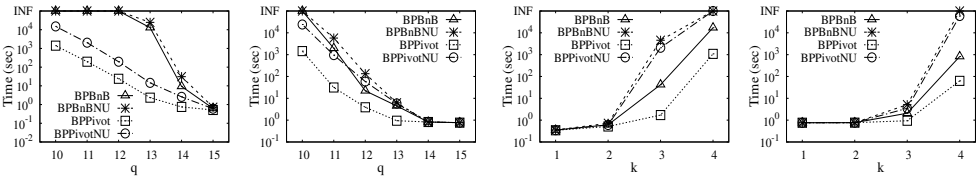
of each algorithm on Amazon and Google datasets with varying values of k and q. Results on the other datasets are consistent. In this experiment, we omit the results of algorithm BPBnB as they follow a similar trend to those of BPPivot. From Fig. 4, we note that the runtime of BPPivot constantly outperforms those algorithms that do not incorporate the core-based reduction or the butterfly-based reduction. Moreover, the runtime of BPPivot (with all graph reduction techniques) can be several times faster than that of BPPivotC (only with the core-based graph reduction). For instance, on Google, when $k = 4$ and $q = 20$, BPPivot only takes 63 seconds to complete the computations, while BPPivotC requires more than 1000 seconds. These results confirm the effectiveness of the graph reduction proposed in Sec. 4.1. Additionally, we observe that both BPEAN and BPPivotN are inefficient in enumerating relatively large k-biplexes when no graph reduction techniques are applied. This is due to the direct utilization of the original graphs as input, resulting in the enumeration of numerous unnecessary maximal k-biplexes with small sizes. It is worth noting that even equipped with our proposed butterfly-based reduction, the runtime of the state-of-the-art algorithm (BPEAB) remains orders of magnitude less efficient than our proposed algorithm BPPivot on most parameter settings. This is attributed to the remarkable performance of our pivot-based branching rule in reducing unnecessary computations, further demonstrating the superiority of the proposed pivoting techniques and graph reductions over existing solutions.

Exp-3: Efficiency of graph reduction techniques. In this experiment, we further evaluate the efficiency of two graph reduction techniques. Here, we denote by RD-Core and RD-Bufly the number of remaining vertices obtained by core-based reduction and butterfly-based reduction, respectively, denote by TM-Core and TM-Bufly the time spend of core-based reduction and butterfly-based reduction in processing bipartite graphs, respectively. Fig. 5 shows the detailed results on Amazon and Google datasets, with varying the values of k and q. As can be seen, RD-Bufly significantly outperforms RD-Core in terms of vertex reduction. For instance, when $k = 2$ and $q = 10$ on the Amazon dataset, RD-Bufly yields fewer than 7500 remaining vertices, whereas RD-Core retains at least 84000 vertices under the same parameter settings. We also note that the pruning performance of RD-Bufly varies relative smoothly against a decrease in q (or an increase in k) compared to that of RD-Core. Moreover, although the time spend of the butterfly-based reduction (TM-Bufly) can higher than that of core-based reduction (TM-Core), TM-Bufly still consumes a few seconds at most on most parameter settings. Nevertheless, we can observe that the runtime of TM-Bufly is negligible on most parameter settings when comparing with that of BPPivot (based



(a) Amazon ($k = 2$) (b) Google ($k = 2$) (c) Amazon ($q = 15$) (d) Google ($q = 15$)

Fig. 6. Effects of ordering optimizations.



(a) Amazon ($k = 2$) (b) Google ($k = 2$) (c) Amazon ($q = 15$) (d) Google ($q = 15$)

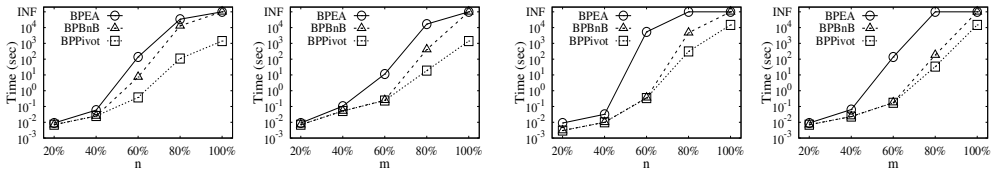
Fig. 7. Effects of upper-bound techniques.

on the results shown in Fig. 4). These results demonstrate the excellent ability of the proposed butterfly-based graph reduction technique in eliminating unnecessary vertices from the given bipartite graph.

Exp-4: Results of ordering techniques. To evaluate the effectiveness of ordering techniques, we introduce two additional algorithms BPBnBNO and BPPivotNO, where BPBnBNO and BPPivotNO are the variants of our BPBnB and BPPivot algorithms, respectively, without utilizing ordering optimizations. Fig. 6 presents the experimental results of these algorithms on Amazon and Google with varying values of k and q. As can be seen, the algorithms without ordering techniques exhibit significantly lower performance compared to the algorithms with these techniques. For instance, on Amazon, when $k = 2$ and $q = 12$, the algorithm BPPivot completes the computations in less than 100 seconds, while BPPivotNO fails to terminate within 24 hours. Thus, the results highlight the important impact of ordering techniques on the efficiency of the algorithms for enumerating maximal k-biplexes. The effect stems from the fact that these techniques greatly reduce the search space by dividing the original problem into a series of subproblems over small bipartite graphs, thereby transforming it into a more manageable enumeration problem.

Exp-5: Results of upper-bound techniques. In this experiment, we evaluate the effectiveness of the proposed upper-bound techniques. Fig. 7 displays the results of each algorithm on Amazon and Google with varying values of k and q, where the algorithms BPBnBNU and BPPivotNU are the variants of our BPBnB and BPPivot algorithms, respectively, that do not utilize upper-bound techniques. Similar trends are observed in the other datasets. As can be seen, each algorithm with the upper-bound techniques consistently outperforms than the corresponding algorithm without these techniques on all tested datasets with varying parameters. Moreover, as the size of k increases, the speedup of BPPivot (resp. BPBnB) compared to BPPivotNU (resp. BPBnBNU) also increases in terms of runtime. Specifically, on Google, when $k = 3$ and $q = 15$, BPPivotNU takes less than 5 times as long as BPPivot, but this ratio increases to over 100 when k is increased to 4. This behavior can be attributed to the tightness of the proposed upper-bound technique, which allows for a near-linear time computation. As a result, the proposed algorithms with the upper-bound techniques exhibit excellent performance.

Exp-6: Scalability testing. To evaluate the scalability of our proposed algorithms, we randomly sample 20-80% of vertices and edges from Amazon to generate 8 subgraphs. Subsequently, we test



(a) Amazon ($k = 2, q = 10$) (b) Amazon ($k = 2, q = 10$) (c) Amazon ($k = 3, q = 12$) (d) Amazon ($k = 3, q = 12$)

Fig. 8. Scalability testing for various algorithms.

the enumeration time of all algorithms on each subgraph, and the results are shown in Fig. 8. The results on the other datasets are consistent. As can be seen, the runtime of our proposed algorithms, BPBnB and BPPivot, increases smoothly as the sizes of bipartite graphs increase. Moreover, the algorithm BPPivot consistently demonstrates the best performance on each subgraph over all tested algorithms. More specifically, as the number of vertices or edges increases, the runtime of our algorithm BPPivot remains significantly lower than the state-of-the-art algorithm BPEA. For instance, when $k = 2$ and $q = 10$, BPPivot and BPEA respectively take 0.376, and 137 seconds to complete the computation on the graph Amazon with 60% vertices. However, when increasing to 80% vertices, BPPivot only requires 109 seconds, while BPEA takes more than 34,000 seconds. These results demonstrate the superior scalability of our proposed algorithms in handling large real-world bipartite graphs.

Exp-7: Memory usages of various algorithms. This experiment tests the memory usages of various algorithms on each dataset. The experimental results are shown in Fig. 9. From this figure, we can observe that the memory usages of all algorithms scales linearly with the size of the bipartite graph. Moreover, we note that the memory usage of our algorithms BPBnB and BPPivot is comparable to that of BPEA. This is expected since the worst-case space complexity of both our algorithms and BPEA are bounded by $O(\delta n + m)$. Despite similar memory usage, our algorithms demonstrate superior performance over BPEA when considering the results from Exp-1. Thus, these findings indicate that our algorithms are space-efficient and capable of effectively handling large real-world bipartite graphs.

6 RELATED WORKS

Maximal biclique enumeration. Our work is related to the maximal biclique enumeration problem which has been widely studied in the literature [1, 10, 16–18, 28, 32, 50]. Early approaches to this problem relied on a breadth-first search to find the power set of L (or R) [28, 32]. However, such methods usually suffer from poor performance, prompting the development of depth-first search methods in recent years [1, 10, 16, 18, 50]. Specifically, Zhang et al. [50] proposed an algorithm to enumerate maximal bicliques from the vertex set with a smaller number of vertices, based on the feature that all maximal bicliques can be identified by only detecting the subsets of the vertex set L (or R) only. Das et al. [18] observed that the performance can be improved by dividing the original enumeration problem into a series of subproblems using an ordering technique. Abidi et al. [1] developed a pivot-based algorithm, based on a dominating technique, to enumerate maximal bicliques. Chen et al. [10] combined the ordering technique and the dominating technique to further improve the efficiency of enumerating maximal bicliques. Recently, Dai et al. [16] presented a novel pivot-based algorithm, which not only achieves the near-optimal worst-case time complexity but also has the polynomial delay time complexity. However, all the above mentioned techniques are mainly tailored to biclique enumeration. Extending such techniques to solve maximal k-biplex enumeration problem is much less than our proposed algorithms as shown in our experiments.

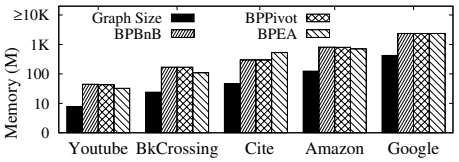


Fig. 9. Memory usages of various algorithms.

Cohesive subgraph mining on bipartite graphs In addition to the biclique [1, 10, 16, 18, 50] and k-biplex [16, 38, 48, 49] models, many other cohesive subgraphs on bipartite graphs have also been studied, including quasi-biclique [22, 33, 42], (l, r)-core [9, 31], k-bitruss [41, 52], and others. All of these mentioned cohesive subgraph models can be regarded as relaxations of the biclique model. The problem of mining quasi-bicliques was shown to be NP-hard [22, 42], while mining (l, r)-core [9] and k-bitruss [41] can be achieved in polynomial time. Efficient enumeration algorithms for mining quasi-bicliques have been developed by [42] and [22]. The notion of (l, r)-core was originally proposed in [9]. The state-of-the-art algorithm for (l, r)-core computation is an index-based algorithm presented in [31]. The concept of k-bitruss was first proposed by Zou [52]. Wang et al. [41] developed an efficient index-based algorithm to compute the k-bitruss. Since all these models do not satisfy the hereditary property, all the solutions cannot extend to solve the problem of enumerating maximal k-biplexes.

Maximal k-plex enumerations on traditional graphs. The k-plex model has been extensively studied on traditional graphs [5, 13–15, 37, 43–45, 51], and it is highly relevant to the notion of k-biplex on bipartite graphs. The k-biplex model was first introduced in [37], and many advanced approaches have been developed to enumerate all maximal k-plexes [5, 43–45, 51]. Since the time cost of enumerating all maximal k-plexes is very high, many existing solutions turn to enumerate maximal k-plexes with size no less than a threshold q [14, 15, 43, 44, 51]. For instance, the algorithms based on the clique and k-core reduction and the pivoting technique were developed by Conte et al. [13, 14]. Recently, Zhou et al. [51] first developed a branch-and-bound algorithm with a non-trivial worst-case time complexity. Dai et al. and Wang et al. [15, 44] further improved the efficiency of enumerating maximal k-plexes while guaranteeing the similar time complexity as [51], respectively. However, the structure of k-biplex on the bipartite graph is very different from that of k-plex on traditional graphs, and thus it is quite non-trivial to extend existing k-plex enumeration techniques for k-biplex enumeration.

7 CONCLUSION

In this paper, we study the problem of enumerating maximal k-biplexes on bipartite graphs. To overcome the theoretical and practical inefficiencies of existing methods, we propose several novel and efficient solutions. Specifically, we first propose two efficient enumeration algorithms based on several newly-developed branching rules. We show that our best algorithm achieves a time complexity of $O(m\beta_k^n)$ ($\beta_k < 2$), which represents the current state-of-the-art according to our knowledge. Moreover, we also present several non-trivial optimization techniques, including graph reduction, upper-bounds based pruning, and ordering-based optimization, to further improve the efficiency of our algorithms. Finally, extensive experimental results on 5 real-world datasets demonstrate the high efficiency and scalability of the proposed algorithms.

ACKNOWLEDGMENTS

This work was partially supported by (i) National Key Research and Development Program of China 2020AAA0108503, (ii) NSFC Grants U2241211 and 62072034, and (iii) the China Postdoctoral Science Foundation Grant 2023M740245. Rong-Hua Li is the corresponding author of this paper.

REFERENCES

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

[2] Mohammad Allahbakhsh, Aleksandar Ignjatovic, Boualem Benatallah, Seyed-Mehdi-Reza Beheshti, Elisa Bertino, and Norman Foo. 2013. Collusion Detection in Online Rating Systems. In *APWeb*, Vol. 7808. 196–207.

[3] David Avis and Komei Fukuda. 1996. Reverse Search for Enumeration. *Discret. Appl. Math.* 65, 1-3 (1996), 21–46.

[4] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. *CoRR* cs.DS/0310049 (2003).

[5] Devora Berlowitz, Sara Cohen, and Benny Kimelfeld. 2015. Efficient Enumeration of Maximal k-Plexes. In *SIGMOD*. 431–444.

[6] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. CopyCatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*. 119–130.

[7] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. 2000. Graph structure in the web. *Computer networks* 33, 1-6 (2000), 309–320.

[8] Dongbo Bu, Yi Zhao, Lun Cai, Hong Xue, Xiaopeng Zhu, Hongchao Lu, Jingfen Zhang, Shiwei Sun, Lunjiang Ling, Nan Zhang, et al. 2003. Topological structure analysis of the protein–protein interaction network in budding yeast. *Nucleic acids research* 31, 9 (2003), 2443–2450.

[9] Monika Cerinsek and Vladimir Batagelj. 2015. Generalized two-mode cores. *Soc. Networks* 42 (2015), 80–87.

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

[11] Jens Clausen. 1999. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen* (1999), 1–30.

[12] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. 2008. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *J. Comput. Syst. Sci.* 74, 7 (2008), 1147–1159.

[13] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. 2017. Fast Enumeration of Large k-Plexes. In *SIGKDD*. 115–124.

[14] 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 *KDD*. 1272–1281.

[15] Qiangqiang Dai, Rong-Hua Li, Hongchao Qin, Meihao Liao, and Guoren Wang. 2022. Scaling Up Maximal k-plex Enumeration. In *CIKM*. 345–354.

[16] 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. *Proc. ACM Manag. Data* 1, 2 (2023), 138:1–138:26.

[17] Peter Damaschke. 2014. Enumerating maximal bicliques in bipartite graphs with favorable degree sequences. *Inf. Process. Lett.* 114, 6 (2014), 317–321.

[18] Apurba Das and Srikanta Tirthapura. 2019. Shared-Memory Parallel Maximal Biclique Enumeration. In *HiPC*. 34–43.

[19] Kemal Eren, Mehmet Deveci, Onur Küçüktunç, and Ümit V Çatalyürek. 2013. A comparative analysis of biclustering algorithms for gene expression data. *Briefings in bioinformatics* 14, 3 (2013), 279–292.

[20] Fedor V. Fomin and Dieter Kratsch. 2010. *Exact Exponential Algorithms*. Springer.

[21] Stephan Günnemann, Emmanuel Müller, Sebastian Raubach, and Thomas Seidl. 2011. Flexible Fault Tolerant Subspace Clustering for Data with Missing Values. In *ICDM*. 231–240.

[22] Dmitry I. Ignatov. 2019. Preliminary Results on Mixed Integer Programming for Searching Maximum Quasi-Bicliques and Large Dense Biclusters. In *ICFCA*, Vol. 2378. 28–32.

[23] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. 1999. Trawling the web for emerging cyber-communities. *Computer networks* 31, 11-16 (1999), 1481–1493.

[24] Ailsa H. Land and Alison G. Doig. 2010. An Automatic Method for Solving Discrete Programming Problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. 105–132.

[25] Sune Lehmann, Martin Schwartz, and Lars Kai Hansen. 2008. Biclique communities. *Physical review E* 78, 1 (2008), 016108.

[26] Michael Ley. 2002. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. In *SPIRE*, Vol. 2476. 1–10.

[27] Haiquan Li, Jinyan Li, and Limsoon Wong. 2006. Discovering motif pairs at interaction sites from protein sequences on a proteome-wide scale. *Bioinform.* 22, 8 (2006), 989–996.

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

[29] Rong-Hua Li, Qiushuo Song, Xiaokui Xiao, Lu Qin, Guoren Wang, Jeffrey Xu Yu, and Rui Mao. 2022. I/O-Efficient Algorithms for Degeneracy Computation on Massive Networks. *IEEE Trans. Knowl. Data Eng.* 34, 7 (2022), 3335–3348.
[30] Don R. Lick and Arthur T. White. 1970. k-Degenerate graphs. *Canadian Journal of Mathematics* 22, 5 (1970), 1082–1096.
[31] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2020. Efficient (α, β)-core computation in bipartite graphs. *VLDB J.* 29, 5 (2020), 1075–1099.
[32] Guimei Liu, Kelvin Sim, and Jinyan Li. 2006. Efficient Mining of Large Maximal Bicliques. In *DaWaK*, Vol. 4081. 437–448.
[33] Xiaowen Liu, Jinyan Li, and Lusheng Wang. 2008. Quasi-bicliques: Complexity and Binding Pairs. In *COCOON*, Vol. 5092. 255–264.
[34] Fragkiskos D. Malliaros, Apostolos N. Papadopoulos, and Michalis Vazirgiannis. 2016. Core Decomposition in Graphs: Concepts, Algorithms and Applications. In *EDBT*. 720–721.
[35] Ardian Kristanto Poernomo and Vivekanand Gopalkrishnan. 2009. Towards efficient mining of proportional fault-tolerant frequent itemsets. In *KDD*. 697–706.
[36] Amela Prelić, Stefan Bleuler, Philip Zimmermann, Anja Wille, Peter Bühlmann, Wilhelm Gruissem, Lars Hennig, Lothar Thiele, and Eckart Zitzler. 2006. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics* 22, 9 (2006), 1122–1129.
[37] Stephen B. Seidman and Brian L. Foster. 1978. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology* 6, 1 (1978), 139–154.
[38] 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. *Stat. Anal. Data Min.* 2, 4 (2009), 255–273.
[39] Oliver Voggenreiter, Stefan Bleuler, and Wilhelm Gruissem. 2012. Exact biclustering algorithm for the analysis of large gene expression data sets. *BMC bioinformatics* 13, Suppl 18 (2012), A10.
[40] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR*. 501–508.
[41] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient Bitruss Decomposition for Large-scale Bipartite Graphs. In *ICDE*. 661–672.
[42] Lusheng Wang. 2010. Near Optimal Solutions for Maximum Quasi-bicliques. In *COCOON*, Vol. 6196. 409–418.
[43] Zhuo Wang, Qun Chen, Boyi Hou, Bo Suo, Zhanhuai Li, Wei Pan, and Zachary G. Ives. 2017. Parallelizing maximal clique and k-plex enumeration over graph data. *J. Parallel Distributed Comput.* 106 (2017), 79–91.
[44] Zhengren Wang, Yi Zhou, Mingyu Xiao, and Bakhadyr Khoussainov. 2022. Listing Maximal k-Plexes in Large Real-World Graphs. In *WWW*. 1517–1527.
[45] Bin Wu and Xin Pei. 2007. A Parallel Algorithm for Enumerating All the Maximal k-Plexes. In *PAKDD*, Vol. 4819. Springer, 476–483.
[46] Mihalis Yannakakis. 1981. Node-Deletion Problems on Bipartite Graphs. *SIAM J. Comput.* 10, 2 (1981), 310–327.
[47] Kaiqiang Yu and Cheng Long. 2023. Maximum k-Biplex Search on Bipartite Graphs: A Symmetric-BK Branching Approach. *Proc. ACM Manag. Data* 1, 1 (2023), 49:1–49:26.
[48] Kaiqiang Yu, Cheng Long, Shengxin Liu, and Da Yan. 2022. Efficient Algorithms for Maximal k-Biplex Enumeration. In *SIGMOD*. 860–873.
[49] Kaiqiang Yu, Cheng Long, Deepak P, and Tanmoy Chakraborty. 2023. On Efficient Large Maximal Biplex Discovery. *IEEE Trans. Knowl. Data Eng.* 35, 1 (2023), 824–829.
[50] Yun Zhang, Charles A. Phillips, Gary L. Rogers, Erich J. Baker, Elissa J. Chesler, and Michael A. Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinform.* 15 (2014), 110.
[51] Yi Zhou, Jingwei Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. 2020. Enumerating Maximal k-Plexes with Worst-Case Time Guarantee. In *AAAI*. 2442–2449.
[52] Zhaonian Zou. 2016. Bitruss Decomposition of Bipartite Graphs. In *DASFAA*, Vol. 9643. 218–233.

Received October 2023; revised January 2024; accepted February 2024