Maximum Defective Biclique Search in Large Bipartite Graphs

Donghang Cui
Beijing Institute of Technology
Beijing, China
cuidonghang@bit.edu.cn

Rong-Hua Li
Beijing Institute of Technology
Beijing, China
lironghuabit@126.com

Qiangqiang Dai
Beijing Institute of Technology
Beijing, China
qiangd66@gmail.com

Hongchao Qin
Beijing Institute of Technology
Beijing, China
qhc.neu@gmail.com

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggrbit@126.com

ABSTRACT

The problem of identifying the maximum edge biclique in bipartite graphs has attracted considerable attention in bipartite graph analysis, with numerous real-world applications such as fraud detection, community detection, and online recommendation systems. However, real-world graphs may contain noise or incomplete information, leading to overly restrictive conditions when employing the biclique model. To mitigate this, we focus on a new relaxed subgraph model, called the k-defective biclique, which allows for up to k missing edges compared to the biclique model. We investigate the problem of finding the maximum edge k-defective biclique in a bipartite graph, and prove that the problem is NP-hard. To tackle this computation challenge, we propose a novel algorithm based on a new branch-and-bound framework, which achieves a worst-case time complexity of $O(m\alpha_k^n)$, where $\alpha_k < 2$. We further enhance this framework by incorporating a novel pivoting technique, reducing the worst-case time complexity to $O(m\beta_k^n)$, where $\beta_k < \alpha_k$. To improve the efficiency, we develop a series of optimization techniques, including graph reduction methods, novel upper bounds, and a heuristic approach. Extensive experiments on 11 large real-world datasets validate the efficiency and effectiveness of the proposed approaches. The results indicate that our algorithms consistently outperform state-of-the-art algorithms, offering up to 1000× speedups across various parameter settings.

PVLDB Reference Format:
Donghang Cui, Rong-Hua Li, Qiangqiang Dai, Hongchao Qin, and Guoren Wang. Maximum Defective Biclique Search in Large Bipartite Graphs. PVLDB, 19(11): 3259 - 3272, 2026.
doi:10.14778/3836663.3836687

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/pluv1a/MaximumDefectiveBiclique.

1 INTRODUCTION

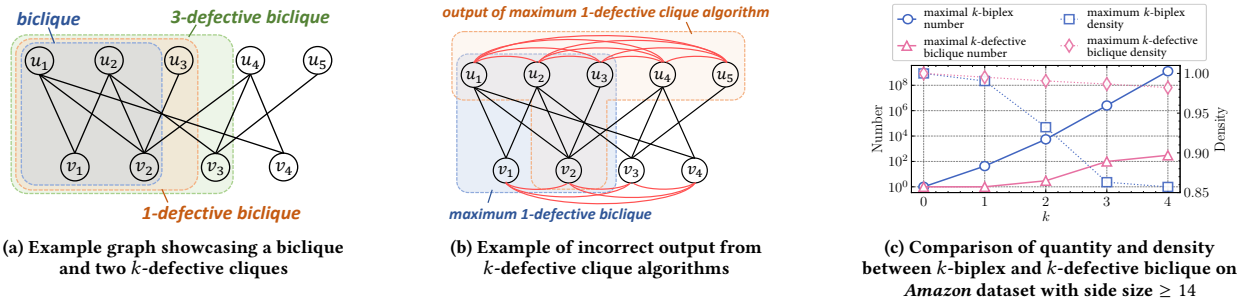
Bipartite graph is a commonly encountered graph structure consisting of two disjoint vertex sets and an edge set, where each edge connects vertices from the two different sets. In recent decades, bipartite graphs have been utilized as a fundamental tool for modeling various binary relationships, such as user-product interactions in e-commerce [31], authorship connections between authors and publications [30], and associations between genes and phenotypes in bioinformatics [16, 41]. Mining cohesive subgraphs from bipartite graphs has proven to be critically important across diverse domains [14, 49]. For instance, identifying cohesive subgraphs in reviewer-product graphs facilitates the identification of fraudulent reviews or camouflage attacks [23].

The biclique, which connects all vertices on both sides, serves as a fundamental cohesive subgraph model [32, 37, 50] due to its wide applications [4, 6, 28, 29]. Extensive methods for maximum biclique search have been developed in recent years [4, 14, 17, 21, 34, 36]. However, real-world graphs often contain noisy or incomplete information [33], making the biclique model overly restrictive for capturing cohesive subgraphs on such bipartite graphs. More relaxed biclique models offer a promising alternative to address this challenge. The k-biplex model, which allows each vertex to have at most k non-neighbors, has recently attracted significant attention [14, 54, 55]. However, due to its excessive structural relaxation, a slight increase in k can lead to: (1) a sharp rise in quantity, reducing the mining efficiency; and (2) a significant decrease in density, undermining its capability to represent cohesive communities.

Inspired by the widely studied k-defective clique in traditional graphs [12, 13, 53], we introduce a novel cohesive subgraph model for bipartite graphs, named k-defective biclique. Specifically, a k-defective biclique is a subgraph of a bipartite graph containing at most k missing edges. Fig. a illustrates an example graph showcasing two k-defective bicliques, where the 1-defective biclique has one missing edge (u_3, v_1), and the 3-defective biclique has three missing edges (u_1, v_3), (u_3, v_1) and (u_3, v_3).

The k-defective biclique offers greater flexibility over biclique, as its density can be controlled by adjusting k to suit specific requirements. When $k = 0$, the k-defective biclique degenerates to the biclique, thereby illustrating its generalization capability. In comparison to the k-biplex, k-defective biclique offers a more fine-grained relaxation of the biclique due to its stricter retention of edges. As illustrated in Fig. c, the k-defective biclique exhibits much closer quantity and density to the biclique compared to k-biplex, especially for larger k. Here, we refer to a k-defective biclique as "maximal" if it cannot be contained within any other k-defective biclique (or k-biplex), and as "maximum" when it is the maximal

Proceedings of the VLDB Endowment, Vol. 19, No. 11 ISSN 2150-8097.
doi:10.14778/3836663.3836687



(a) Example graph showcasing a biclique and two k-defective cliques

(b) Example of incorrect output from k-defective clique algorithms

(c) Comparison of quantity and density between k-biplex and k-defective biclique on *Amazon* dataset with side size ≥ 14

Figure 1: Illustrations of k-defective biclique: examples and comparisons

k-defective biclique (or k-biplex) with the largest number of edges. In this paper, we focus mainly on the problem of identifying a maximum k-defective biclique. We prove that the maximum k-defective biclique search problem is NP-hard, indicating that no polynomial-time algorithms exist for solving this problem unless NP=P.

Notably, many studies have addressed the maximum k-defective clique search problem on traditional graphs [8, 9, 12, 13, 19, 27], which mainly focus on designing branch-and-bound algorithms to identify the maximum k-defective clique. A straightforward idea to applying these algorithms to the maximum k-defective biclique search is to fully connect the vertices on each side of the bipartite graph and then use these algorithms to find a maximum k-defective clique. However, such an approach may lead to suboptimal results. For instance, in Fig. b, the maximum 1-defective biclique has 5 edges whereas the output of maximum 1-defective clique algorithms yields only 4 edges. In addition, several studies have proposed algorithms for the maximum biclique search problem [21, 36, 46, 50]. However, due to the presence of missing edges, identifying a maximum k-defective biclique through maximum biclique search is non-trivial, and adapting these algorithms to the maximum k-defective biclique search problem remains a challenging task.

Contributions. To address these issues, we comprehensively study the maximum k-defective biclique search problem, and make the following contributions:

A new cohesive subgraph model. We propose k-defective biclique, a new cohesive subgraph model in bipartite graph that allows up to k missing edges, and mathematically prove that the maximum k-defective biclique search problem is NP-hard.

Two Novel Search Algorithms. We propose two novel maximum k-defective biclique search algorithms with nontrivial worst-case time complexity guarantees. The first algorithm utilizes a branch-and-bound technique combined with a newly-developed binary branching strategy, which has a time complexity of $O(m\alpha_k^n)$, where n and m represent the number of vertices and edges of the bipartite graph respectively, and α_k is a positive real number strictly less than 2 (e.g., for $k = 1, 2, 3$, $\alpha_k = 1.911, 1.979$, and 1.995, respectively). The second algorithm employs an innovative pivoting-based branching strategy that effectively integrates with our binary branching strategy, which achieves a lower time complexity of $O(m\beta_k^n)$, where β_k is a positive real number strictly less than α_k. (e.g., when $k = 1, 2, 3$, $\beta_k = 1.717, 1.856$, and 1.931, respectively).

Nontrivial optimization techniques. We develop a series of nontrivial optimization techniques to improve the efficiency of our proposed algorithms, including graph reduction techniques, novel upper-bounding techniques, a heuristic approach, and parallelization techniques. Specifically, We propose graph reduction techniques to reduce the size of input graph. We devise tight vertex and edge upper bounds to reduce unnecessary branching and compute them in linear time. Additionally, we propose a heuristic approach to obtain an initial large solution. We also parallelize our algorithms to further enhance their practical efficiency.

Comprehensive experimental studies. We conduct extensive experiments to evaluate the efficiency, effectiveness, and scalability of our algorithms on 11 large real-world graphs. The experimental results show that our algorithms consistently outperform the state-of-the-art baselines across all parameter settings. In particular, our algorithm achieves improvements of up to three orders of magnitude on most datasets. For instance, on the *LKML* (9M vertices, 10M edges) dataset with $k = 3$, our algorithm takes 10 seconds while the best baseline takes about 2.5 hours. Additionally, we perform case studies on two real-world tasks to show the high effectiveness of our solutions in real-world applications.

2 PRELIMINARIES

Let $G = (U, V, E)$ be an undirected bipartite graph, where U and V are two disjoint sets of vertices, and $E \subseteq U \times V$ is the set of edges. Denote by $n = |U| + |V|$ and $m = |E|$ the total number of vertices and edges in G, respectively. Denote by $S = (U_S, V_S)$ a vertex subset of G, where $U_S \subseteq U$ and $V_S \subseteq V$. Let $v \in S$ indicate that $v \in U_S \cup V_S$. Let $G(S) = (U_S, V_S, E_S)$ be the subgraph of G induced by S, where $E_S = \{(u, v) \mid u \in U_S \wedge v \in V_S \wedge (u, v) \in E\}$. For $u \in U$, denote by $N(u) = \{v \in V \mid (u, v) \in E\}$ and $\overline{N}(u) = \{v \in V \mid (u, v) \notin E\}$ the set of neighbors and non-neighbors of u in G, respectively. Similar definitions apply for the vertices in V. Let $d(u) = |N(u)|$ and $\overline{d}(u) = |\overline{N}(u)|$ be the degree and non-degree of u in G, respectively. For a vertex subset (or a subgraph) S of G, let $N_S(u) = N(u) \cap (U_S \cup V_S)$ and $\overline{N}_S(u) = \{v \in S \setminus N(u) \mid u \text{ and } v \text{ are on opposite sides}\}$ be the set of u's neighbors and non-neighbors in S. respectively. Let $d_S(u) = |N_S(u)|$ and $\overline{d}_S(u) = |\overline{N}_S(u)|$ be the degree and non-degree of u in S, respectively. For a bipartite graph $G = (U, V, E)$, we refer to $(u, v) \in U \times V$ as a non-edge of G if $(u, v) \notin E$. We denote $\overline{E}_S$

the set of non-edges in $G(S)$, i.e., $\overline{E}_S = U_S \times V_S \setminus E_S$. The formal definition of k-defective biclique is given below.

Definition 2.1 (k-defective biclique). Given a bipartite graph $G = (U, V, E)$, a k-defective biclique $D = (U_D, V_D, E_D)$ of G is a subgraph of G that has at most k non-edges, i.e., $|U_D \times V_D \setminus E_D| \leq k$.

The k-defective biclique corresponds to the biclique if $k = 0$, which indicates that k-defective biclique represents a more general structure. If a k-defective biclique of G contains the most number of edges among all k-defective bicliques in G, we refer to it as a maximum k-defective biclique in G. To facilitate the illustration, we denote the maximum k-defective biclique as k-MDB.

In this paper, we focus on the problem of searching for a k-MDB in a bipartite graph. However, we observe that k-MDBs in many real-world graphs may exhibit a skewed structure, similar to that of maximum bicliques, where one side has a large number of vertices while the other side may contain only a few (or even one) vertices. Such k-MDBs often hold limited practical values in network analysis. To address this issue, we define a size threshold θ, representing the minimum number of vertices required on both sides of the k-MDB. Furthermore, we impose the condition $\theta > k$ to ensure that the k-MDB is connected, as demonstrated in the following lemma.

Lemma 2.2. *Given a bipartite graph G and a k-defective biclique $D = (U_D, V_D, E_D)$ of G, if $|U_D| > k$ and $|V_D| > k$, we establish that D is a connected subgraph in G.*

Proof. Assume that a k-defective biclique D is not connected when $|U_D| > k$ and $|V_D| > k$. In this case, D can be decomposed into two nonempty subgraphs $D_1 = (U_{D_1}, V_{D_1}, E_{D_1})$ and $D_2 = (U_{D_2}, V_{D_2}, E_{D_2})$, with the results of $E_{D_1} \cup E_{D_2} = E_D$.The number of non-edges between D_1 and D_2 is given by $|U_{D_1}| \cdot |V_{D_2}| + |U_{D_2}| \cdot |V_{D_1}|$, which reaches at least $|U_D|$ or $|V_D|$ and is thus larger than k. This contradicts the assumption that D is a k-defective biclique. Therefore, the proof is complete. □

Based on Lemma 2.2, we formally define the problem addressed in this paper as follows.

Problem statement (MDB search problem)**.** Given a bipartite graph $G = (U, V, E)$ and two integers k and θ with $\theta > k$, the MDB search problem aims to find a connected k-MDB $D = (U_D, V_D, E_D)$ of G satisfying $|U_D| \geq \theta$ and $|V_D| \geq \theta$.

Theorem 2.3. *The MDB search problem is NP-hard.*

Proof. We establish the NP-hardness of the MDB search problem through a polynomial-time reduction from the classical NP-complete maximum clique search problem. Due to space limitation, we present the complete version of this proof and all the omitted proofs of this paper in [1]. □

2.1 Related Work

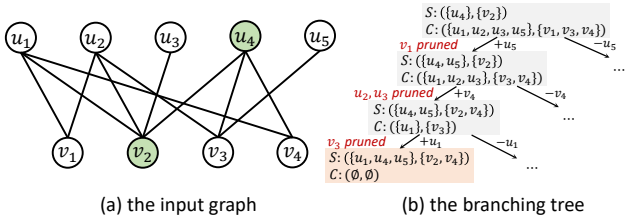
Maximum k-defective clique search. This problem is closely related to our work and has been extensively studied [8, 9, 12, 13, 19, 22, 27, 35, 45, 47]. Early exact algorithms adopted the Russian Doll search scheme together with preprocessing and verification techniques [22, 47]. Subsequent studies mainly developed branch-and-bound algorithms by introducing coloring-based upper bounds [12], vertex and edge reduction rules [19], and improved branching and ordering strategies [8, 9], progressively tightening both theoretical time complexities and practical performance. Most recently, Dai et al. [13] proposed a pivot-based branch-and-bound algorithm equipped with carefully designed pivoting technique and several optimization techniques, achieving the state-of-the-art theoretical and empirical performance. Nevertheless, these algorithms maximize the number of vertices in a k-defective clique, whereas the MDB problem maximizes the number of edges, preventing their direct application to MDB search.

Maximum biclique search. Maximum biclique search has also been widely investigated [5, 10, 15, 17, 21, 36, 38, 40, 43, 46, 56]. Existing studies consider three main objectives: maximizing the number of edges, maximizing the number of vertices, and finding a balanced biclique. Maximum vertex biclique is polynomial-time solvable via minimum cut [20], whereas maximum balanced biclique has been addressed using evolutionary search, local search, upper-bound propagation, and branch-and-bound techniques [10, 51, 56, 58]. For maximum edge biclique, earlier studies established its NP-hardness and inapproximability [5, 40], and developed probabilistic and integer-programming approaches [43, 44]. Most recently, Lyu et al. [36] proposed a branch-and-bound algorithm that combines progressive bounding with graph reductions, enabling exact search on billion-scale graphs. However, these methods assume complete connectivity and therefore cannot directly handle the missing edges allowed in k-MDB.

Branch-and-bound and pivoting techniques. The branch-and-bound and pivoting techniques has been widely used for mining cohesive subgraph models, including k-defective clique [8, 9, 13], k-biplex [54, 55], and biclique [2, 3, 11]. The state-of-the-art maximum k-defective clique search algorithm [13] utilizes a pivoting technique that selects a pivot fully adjacent to the current solution, but such a vertex cannot exist in a bipartite graph because vertices on the same side are always nonadjacent; explicitly adding these intra-side edges would incur quadratic overhead. The state-of-the-art maximum k-biplex search algorithm [54] branches on a pivot with more than k non-neighbors and generates at most $k + 1$ sub-branches, whereas a k-defective biclique may contain no such pivot, causing substantially more branches when this strategy is adapted. The state-of-the-art maximal biclique enumeration algorithm [2] selects a pivot from only one side according to common-neighbor counts and a heuristic topological order, which may overlook a better pivot on the opposite side. Although these methods can be adapted to MDB search, their pivoting strategies do not exploit its bipartite defective structure and remain inefficient in practice.

3 OUR PROPOSED ALGORITHMS

In this section, we propose two novel algorithms to solve the k-MDB search problem. Specifically, we first propose a new branch-and-bound algorithm that utilizes a newly-designed binary branching strategy, achieving the worst-case time complexity of $O(m\alpha_k^n)$, where $\alpha_k < 2$ (e.g. $k = 1, 2$, and 3, we have $\alpha_k = 1.911, 1.979$, and 1.995, respectively). To further improve the efficiency, we propose a novel pivoting-based algorithm that incorporates a well-designed pivoting-based branching strategy with our proposed binary branching strategy. We prove that the time complexity of



(a) the input graph (b) the branching tree

Figure 2: An example of the new binary branching strategy, where $k = 2$ and current instance is $S = (\{u_4\}, \{v_2\})$ (green vertices) and $C = (\{u_1, u_2, u_3, u_5\}, \{v_1, v_3, v_4\})$ (uncolored vertices).

the improved algorithm is bounded by $O(m\beta_k^n)$, where $\beta_k < \alpha_k$ (e.g., when $k = 1, 2, 3$, $\beta_k = 1.717, 1.856$, and 1.931, respectively). To the best of our knowledge, the algorithms we proposed in this section are the first two k-MDB search algorithms that surpass the trivial time complexity of $O^*(2^n)$.

3.1 A New Branch-and-bound Algorithm

We firstly introduce a branch-and-bound algorithm for the k-MDB problem. It employs a straightforward branch-and-bound approach by selecting a vertex v from the graph and dividing the k-MDB search problem into two subproblems: one seeks a k-MDB that includes v, while the other seeks a k-MDB that excludes v. Specifically, given a bipartite graph $G = (U, V, E)$, let $S = (U_S, V_S)$ denote a partial set, where $G(S)$ is a k-defective biclique of G. Let $C = (U_C, V_C)$ denote a candidate set, where for any vertex $u \in C$, $G(S \cup \{u\})$ forms a larger k-defective biclique of G. Let (S, C) denote an instance of the k-MDB problem, where the goal is to find a k-MDB in $G(S \cup C)$ containing S. In particular, by setting $S = (\emptyset, \emptyset)$ and $C = (U, V)$, the initial instance $((\emptyset, \emptyset), (U, V))$ represents the global search for a k-MDB of G. When C becomes empty, the instance $(S^*, (\emptyset, \emptyset))$ yields a possible solution $G(S^*)$.

Given an instance (S, C), our branch-and-bound algorithm selects a vertex $u \in C$ and generates two new sub-instances: $(S \cup \{u\}, C \setminus \{u\})$ and $(S, C \setminus \{u\})$. To clarify, we refer to such a vertex u as a *branching vertex*. The algorithm then recursively processes each sub-instance in the same manner until C becomes empty, where S yields a possible solution. Finally, the k-MDB of G is obtained as the yielded solution with most edges. The simplest approach to obtain a branching vertex is to randomly select one from C. However, this strategy may attempt to add all the vertices in C to S, leading to many unnecessary computations. To improve the efficiency, we propose a new binary branching strategy.

The key question to improve the efficiency is how to choose a branching vertex so that the size of C can be reduced as early as possible. Intuitively, suppose there are already k non-edges in S. According to the definition of k-defective biclique, any vertex in C that has at least one non-neighbor in S can be directly pruned. Furthermore, if $v \in C$ has no non-neighbor in S, then adding v to S causes all of v's non-neighbors in C to be pruned. Consequently, we prioritize selecting a branching vertex that can introduce more non-edges to S, so that the number of non-neighbors in S reaches k as early as possible. Based on these observations, we give our new binary branching strategy as follows.

Algorithm 1: The branch-and-bound algorithm

```
   Input: Bipartite graph G = (U, V, E), integers k and θ
   Output: A k-MDB of G
1  D* ← an empty graph;
2  BranchB((∅, ∅), (U, V));
3  return D*;
4  Procedure BranchB(S = (U_S, V_S), C = (U_C, V_C)):
5      if C = ∅ then
6          if |E_{G(S)}| > |E_{D*}| and |U_S| ≥ θ and |V_S| ≥ θ then
7              D* ← G(S);
8          return;
9      u ← arg max_{v∈C} d̄_S(v);
10     if d̄_S(u) = 0 then u ← arg max_{v∈C} d̄_C(v) ;
11     (S', C') ← Update(S, C, u);
12     BranchB(S' ∪ {u}, C'); BranchB(S, C \ {u});
13 Function Update(S, C, u):
14     C' ← {v ∈ C \ {u} | d̄_{S∪{u}}(v) ≤ k − d̄(S)};
15     C'_0 ← {v ∈ C' | d̄_{S∪{u}∪C'}(v) = 0};
16     return (S ∪ C'_0, C' \ C'_0);
```

New binary branching strategy. Given an instance (S, C), let $u = \arg\max_{u \in C} \overline{d}_S(u)$. The sub-instances are generated as follows:

- If $\overline{d}_S(u) > 0$, select u as the branching vertex and generate two sub-instances: $(S \cup \{u\}, C \setminus \{u\})$, $(S, C \setminus \{u\})$.
- Otherwise, select $v \in C$ with the maximum $\overline{d}_C(v)$ as the branching vertex, and generate two sub-instances: $(S \cup \{v\}, C \setminus \{v\})$, $(S, C \setminus \{v\})$.

Example 3.1. Fig. 2 illustrates an example of our new binary branching strategy for finding a 2-MDB. The input graph is depicted in Fig. 2(a). Let the current instance be $S = (\{u_4\}, \{v_2\})$ and $C = (\{u_1, u_2, u_3, u_5\}, \{v_1, v_3, v_4\})$. Fig. 2(b) illustrates the search tree guided by our strategy. At the first level, u_5 and v_1 each have one non-neighbor in S; lexicographic order selects u_5. Adding it introduces one missing edge to S and makes v_1 infeasible because v_1 then has two non-neighbors in $S \cup \{u_5\}$. At the second level, v_4 is the only candidate with a non-neighbor in $S \cup \{u_5\}$. Adding v_4 to S introduces another non-edge to S, reaching the limit of $k = 2$. Therefore, u_2 and u_3 are pruned as they still have non-neighbors in S. The remaining vertices in C are fully adjacent to S, and u_1 is selected by maximum $\overline{d}_C(\cdot)$. Finally, adding u_1 to S prunes v_3 from C, which efficiently terminates this branch.

Implementation details. Algorithm 1 outlines the pseudocode of our proposed branch-and-bound algorithm. It takes a bipartite graph G and two integers k and θ as input parameters, and returns a k-MDB D^* of G. It first initializes D^* as an empty graph, and then starts to search k-MDB by invoking the procedure `BranchB` where all vertices in G serve as the candidate set (line 2). Specifically, `BranchB` holds an instance (S, C) as its parameter, where S denotes the partial set and C denotes the candidate set. Within `BranchB`, the algorithm first determines whether a larger k-defective biclique has been found (lines 5–8). After that, it selects a branching vertex u from C using the new binary branching strategy (lines 9–10) and generates two sub-instance $(S' \cup \{u\}, C')$ and $(S, C \setminus \{u\})$ (lines 11–12). Here, S' and C' are the new partial set and new candidate set derived by invoking `Update` (line 11), ensuring the candidate set keeps all valid branching vertex (lines 14–15). We next analyze the time complexity of Algorithm 1.

Theorem 3.2. *Given a bipartite graph $G = (U, V, E)$ and an integer k, the time complexity Algorithm 1 is given by $O(m\alpha_k^n)$, where α_k is the largest real root of equation $x^{2k+5} - 2x^{2k+4} + x^3 - x^2 + 1 = 0$. Specifically, when $k = 1, 2$ and 3, we have $\alpha_k = 1.911, 1.979$ and 1.995, respectively.*

Proof. Let $T(n, l)$ denote the maximum number of leaf instances generated by BranchB (i.e., instances with an empty candidate set C), where $n = |C|$ represents the number of candidate vertices and $l = k - |\overline{E}(S)|$ represents the remaining number of non-edges can be added to S. As $l \leq k$, we have that $T(n, l) \leq T(n, k)$. Since each recursive call of BranchB takes $O(m)$ time, the total time complexity of BranchB is $O(m \cdot T(n, k))$.

We next analyze $T(n, k)$ based on the binary branching strategy. There are two cases in each branch of BranchB:

(1) **Case 1:** $\max_{v \in C} \overline{d}_S(v) \geq 1$**.** We select $u = \arg\max_{v \in C} \overline{d}_S(v)$. Adding u to S introduces at least one non-edge to S, so the corresponding recurrence is $T(n, k) \leq T(n-1, k) + T(n-1, k-1) \leq \sum_{i=1}^{k} T(n-i, k-i+1) + T(n-k, 0)$. As any $v \in C$ satisfying $\overline{d}_S(v) \geq 1$ can be pruned when $l = 0$, the term $T(n-k, 0)$ is eliminated. Therefore, the recurrence for this case is given by

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

(2) **Case 2:** $\max_{v \in C} \overline{d}_S(v) = 0$**.** We select $u = \arg\max_{v \in C} \overline{d}_C(v)$, and the corresponding recurrence is $T(n, k) \leq T(n-1, k) + T(n-1, k)$. Note that $\overline{d}_C(u) > 0$ (otherwise, $S \cup C$ already forms a valid k-MDB), which ensures that the sub-branch adding u to S triggers Case 1. Consequently, the recurrence for this case is given by $T(n, k) \leq T(n-1, k) + T(n-2, k) + T(n-2, k-1) \leq \sum_{i=1}^{2k} T(n-i, k - \lfloor \frac{i-1}{2} \rfloor) + T(n-2k, 0)$. Considering that when $l = 0$, adding any $v \in C$ with $\overline{d}_C(u)$ to S prunes at least one vertex of C, so we have $T(n-2k, 0) \leq T(n-2k-1, 0) + T(n-2k-2, 0) \leq T(n-2k-1, 0) + T(n-2k-3, 0) + T(n-2k-4, 0)$. Therefore, the recurrence for this case is given by

$$T(n, k) \leq \sum_{i=1}^{2k+1} T(n-i, k - \lfloor \tfrac{i-1}{2} \rfloor) + T(n-2k-3, 0) + T(n-2k-4, 0). \tag{2}$$

The overall worst-case complexity is governed by recurrence (2). By utilizing the algebraic technique from [18], $T(n, k)$ can be expressed in the form of $O(\alpha_k^n)$, where the value of α_k is the largest real root of the equation $x^n = x^{n-1} + x^{n-2} + \cdots + x^{n-2k-1} + x^{n-2k-3} + x^{n-2k-4}$, which simplifies to $x^{2k+5} - 2x^{2k+4} + x^3 - x^2 + 1 = 0$. This completes the proof. □

Lemma 3.3. *Given a bipartite graph $G = (U, V, E)$, the time complexity of Algorithm 1 for finding a 0-MDB of G is $O(m \times 1.618^n)$.*

Proof. When $k = 0$, the vertex $v \in C$ with maximum $\overline{d}_C(v)$ is selected as the branching vertex. Moving v from C to S results in the deletion of v's non-neighbors in C, leading to the recurrence $T(n) \leq T(n-1) + T(n-2)$, which yields $T(n) = 1.618^n$. □

3.2 A Novel Pivoting-based Algorithm

We observe that Algorithm 1 may still generate many unnecessary sub-instances. Consider an instance (S, C) with a branching vertex $v \in C$ such that $\overline{d}_S(v) = 0$. If v and all its non-neighbors are excluded, the remaining vertices cannot form a k-MDB, since any resulting k-defective biclique can be enlarged by adding v. Thus, Algorithm 1 may explore many non-maximum solutions. To avoid such redundant searches, we develop a pivoting-based algorithm and formalize this observation in the following lemma.

Lemma 3.4. *Given an instance (S, C) and any vertex $v \in C$ with $\overline{d}_S(v) = 0$, the k-MDB of instance (S, C) must contain either v or a vertex in $\overline{N}_C(v)$.*

Based on Lemma 3.4, we develop a pivoting strategy that provides a new way to generate sub-instances. The details are presented as follows.

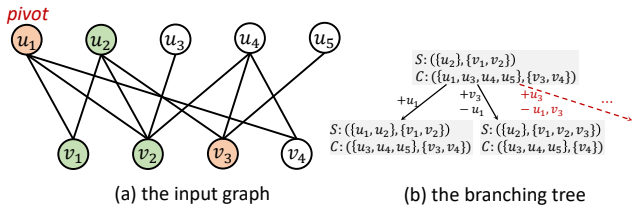
Pivoting strategy. Consider an instance (S, C) and a vertex $u \in C$ with $\overline{d}_S(u) = 0$. Let u be the *pivot vertex*. Assume $\overline{N}_C(u) = \{v_1, v_2, \ldots, v_r\}$, where $r = \overline{d}_C(u)$. The pivoting strategy treats each vertex in $\{u\} \cup \overline{N}_C(u)$ as a branching vertex and generates exactly $r + 1$ sub-instances. The first sub-instance is given by $(S \cup \{u\}, C \setminus \{u\})$, and for $i \geq 2$, the i-th sub-instance is given by $(S \cup \{v_{i-1}\}, C \setminus \{u, v_1, v_2, \ldots, v_{i-1}\})$.

With this strategy, S is expanded only with the vertex u and vertices in $\overline{N}_C(u)$, avoiding redundant sub-instances that would lead to non-maximum k-defective biclique. Next, we consider the selection of the pivot vertex. Let $C_0 \subseteq C$ be the set of vertices fully connected to S, i.e., $C_0 = \{v \in C \mid \overline{d}_S(v) = 0\}$. According to the pivoting strategy, any vertex from C_0 can serve as a pivot vertex. To minimize the number of sub-instances, the vertex $u \in C_0$ with the fewest non-neighbors in C is chosen as the pivot vertex.

In the case where C_0 is empty, every vertex in C has at least one non-neighbor in S, making the pivoting strategy inapplicable. Fortunately, as discussed in Sec. 3.1, continuously branching k times by adding vertices from C to S will result in S containing at least k non-edges. Moreover, C becomes an empty set after this process, which significantly reduces unnecessary computations. Motivated by this observation, we integrate the pivoting strategy with the binary branching strategy to develop a novel pivoting-based branching strategy as outlined below.

Novel pivoting-based branching strategy. Given an instance (S, C), let $C_0 = \{v \in C \mid \overline{d}_S(v) = 0\}$. The new sub-instances of (S, C) are generated in the following way:

- If C_0 is empty or $|C \setminus C_0| > k - |\overline{E}_S|$, let $u = \arg\max_{v \in C} \overline{d}_S(v)$ be the branching vertex and generate two sub-instances: $(S \cup \{u\}, C \setminus \{u\})$ and $(S, C \setminus \{u\})$.
- Otherwise, let $u = \arg\min_{v \in C_0} \overline{d}_C(v)$, and then:
  - If $\overline{d}_C(u) > k - |\overline{E}_S| > 0$, generate two sub-instances with u as the branching vertex: $(S \cup \{u\}, C \setminus \{u\})$ and $(S, C \setminus \{u\})$.
  - Otherwise, assume $\overline{N}_C(u) = \{v_1, v_2, \ldots, v_r\}$, where $r = \overline{d}_C(u)$, and generate $\overline{d}_C(u) + 1$ sub-instances with u as the pivot vertex, where the first sub-instance is given by $(S \cup \{u\}, C \setminus \{u\})$, and for $i \geq 2$, the i-th sub-instance is given by $(S \cup \{v_{i-1}\}, C \setminus \{u, v_1, v_2, \ldots, v_{i-1}\})$.



(a) the input graph (b) the branching tree

Figure 3: An example of the pivoting-based branching strategy, where $k = 2$ and current instance is set as $S = (\{u_2\}, \{v_1, v_2\})$ (green vertices) and $C = (\{u_1, u_3, u_4, u_5\}, \{v_3, v_4\})$ (orange and uncolored vertices).

Example 3.5. Fig. 3 shows an example of the our pivoting-based branching strategy for the 2-MDB search. The example graph is given by Fig. 3(a), with the current instance set as $S = (\{u_2\}, \{v_1, v_2\})$ (green vertices) and $C = (\{u_1, u_3, u_4, u_5\}, \{v_3, v_4\})$ (orange and uncolored vertices). According to the pivoting-based branching strategy, $C_0 = (\{u_1\}, \{v_3\})$, and u_1 is selected as the pivot vertex since it has the fewest non-neighbors in C among vertices in C_0. Then, only u_1 and all non-neighbors of u_1 in C are used to generate new sub-instances (orange vertices), while other vertices of C are not considered (uncolored vertices). This results in only two sub-instances generated by our pivot-based branching strategy. The branching tree for this example is illustrated in Fig. 3(b), where the black arrows represent the generated branches while the red arrows indicate the unnecessary branches.

Implementation details. Algorithm 2 presents the pseudocode of our pivot-based algorithm. It initializes the partial solution as empty, sets all vertices as candidates, and invokes BranchP (lines 1–3). At each recursive call, BranchP updates the global solution D^* when $C = \emptyset$ and $G(S)$ is a better feasible solution (lines 5–7). It then computes $C_0 = \{v \in C \mid \overline{d}_S(v) = 0\}$ and applies one of three branching rules. If $C_0 = \emptyset$ or $|C \setminus C_0| > k - |\overline{E}_S|$, it performs binary branching on a vertex maximizing $\overline{d}_S(v)$ (lines 10–13). Otherwise, it selects $u \in C_0$ minimizing $\overline{d}_C(u)$. When $\overline{d}_C(u) > k - |\overline{E}_S|$, it still performs binary branching on u (lines 15–18); otherwise, the algorithm performs pivoting-based branching strategy by branching on u and its non-neighbors in C, which generates $\overline{d}_C(u) + 1$ sub-instances (lines 20–22). Whenever a vertex is added to S, Update presented in Algorithm 1 is invoked to update the corresponding sub-instance.

Algorithm 2: The pivoting-based algorithm

Input: Bipartite graph $G = (U, V, E)$, integers k and θ
Output: A k-MDB of G

```
1  D* ← an empty graph;
2  BranchP((∅, ∅), (U, V));
3  return D*;
4  Procedure BranchP(S = (U_S, V_S), C = (U_C, V_C)):
5      if C = ∅ then
6          if |E_{G(S)}| > |E_{D*}| and |U_S| ≥ θ and |V_S| ≥ θ then
7              D* ← G(S);
8          return;
9      C_0 ← {v ∈ C | d̄_S(v) = 0};
10     if C_0 = ∅ or |C \ C_0| > k − |Ē_S| then
11         u ← arg max_{v∈C} d̄_S(v);
12         (S', C') ← Update(S, C, u);
13         BranchP(S' ∪ {u}, C'); BranchP(S, C \ {u});
14     else
15         u ← arg min_{v∈C_0} d̄_C(v);
16         if d̄_C(u) > k − |Ē_S| > 0 then
17             (S', C') ← Update(S, C, u);
18             BranchP(S' ∪ {u}, C'); BranchP(S, C \ {u});
19         else
20             for v ∈ {u} ∪ N̄_C(u) do
21                 (S', C') ← Update(S, C, v);
22                 BranchP(S' ∪ {v}, C'); C ← C \ {v};
```

Theorem 3.6. *Given a bipartite graph $G = (U, V, E)$ and an integer k, the time complexity of Algorithm 2 is given by $O(m\beta_k^n)$, where β_k is the largest real root of equation $x^{2k+5} - 2x^{2k+4} + x^{k+3} - 2x + 2 = 0$. Specifically, when $k = 1, 2$ and 3, we have $\beta_k = 1.717, 1.856$ and 1.931, respectively.*

Proof. Let $T(n, l)$ denote the maximum number of leaf instances generated by BranchP, where $n = |C|$ and $l = k - |\overline{E}_S|$ (i.e. the remaining number of non-edges allowed in S). Since $l \le k$, we have $T(n, l) \le T(n, k)$. Finding the vertex with the most (or least) non-neighbors takes $O(m)$ time per recursive call, so BranchP takes $O(m \cdot T(n, k))$ time in total. We analyze $T(n, k)$ based on the three cases of the pivoting-based branching strategy:

(1) **Case 1: $C_0 = \emptyset$ or $|C \setminus C_0| > l$.** BranchP continuously applies the binary branching strategy. After moving at most k vertices from C to S, $k - |\overline{E}(S)|$ reaches 0. At this point, at least one vertex is pruned from C due to having non-neighbors in S. Thus, the recurrence is bounded by:

$$T(n, k) \le \sum_{i=1}^{k+1} T(n - i, k - i + 1). \tag{3}$$

(2) **Case 2: $l = 0$.** The problem degenerates to maximum biclique search using the pivoting strategy. Let $r = \overline{d}_C(u)$. The strategy generates $r + 1$ sub-instances, yielding $T(n, 0) \le T(n - r - 1, 0) + \sum_{i=1}^{r} T(n - r - i, 0)$. If $r = 1$, the recurrence simplifies to $T(n, 0) \le 2T(n - 2, 0)$. If $r \ge 2$, following existing algebraic technique [18], solving this recurrence equivalents to solving $x^n = x^{n-r-1} + \sum_{i=1}^{r} x^{n-r-i}$, which yields a maximum real root of $\gamma_2 = 1.395 < \sqrt{2}$ (for $r \ge 2$). Therefore, the recurrence for this case can always be bounded by:

$$T(n, 0) \le 2T(n - 2, 0). \tag{4}$$

(3) **Case 3: $C_0 \ne \emptyset$ and $0 < |C \setminus C_0| \le l$.** Let the vertex $u = \arg\min_{v \in C_0} \overline{d}_C(v)$. If $\overline{d}_C(u) \le l$, BranchP uses u as the pivot vertex, generating at most $\overline{d}_C(u) + 1 \le l + 1$ sub-instances, yielding $T(n, k) \le \sum_{i=1}^{k+1} T(n - i, k)$. Otherwise, if $\overline{d}_C(u) > l$, BranchP uses u for binary branching. Adding u to S triggers Case 1 in the next level since $|C \setminus C_0| > l - |\overline{E}_{S \cup \{u\}}|$. The alternative branches eventually reduce to $l = 0$ (Case 2). Substituting $T(n - 2k - 2, 0) \le 2T(n - 2k - 4, 0)$ from Case 2, the worst-case recurrence is bounded by:

$$T(n, k) \le \sum_{i=1}^{k+1} T(n - i, k) + 2T(n - 2k - 4, 0). \tag{5}$$

The overall worst-case complexity is governed by Case 3. By utilizing the algebraic technique from [18], let $T(n,k) = O(\beta_k^n)$, where β_k is the largest real root of the equation $x^n = \sum_{i=1}^{k+1} x^{n-i} + 2x^{n-2k-4}$. This simplifies to $x^{2k+5} - 2x^{2k+4} + x^{k+3} - 2x + 2 = 0$, which completes the proof. □

Lemma 3.7. *Given a bipartite graph $G = (U, V, E)$, the time complexity of Algorithm 2 for finding a 0-MDB of G is $O(m \cdot 1.414^n)$.*

Proof. When setting $k = 0$, Algorithm 2 will always execute the pivoting-based branching strategy (lines 20-22), corresponding to a recurrence of $T(n) < 2T(n-2)$ according to the proof of Theorem 3.6. Therefore, we have $T(n) = 1.414^n$. □

4 OPTIMIZATION TECHNIQUES

In this section, we propose various optimization techniques, including graph reduction techniques to eliminate redundant vertices, upper-bounding techniques to prune unnecessary instances, and a heuristic approach for identifying an initial large k-defective biclique. These techniques can be easily integrated into our proposed algorithms in Sec. 3, and can significantly enhance their efficiency.

4.1 Graph Reduction Techniques

Common-neighbor-based reduction. For two vertices u and v on the same side, let $cn(u,v)$ (or $cn_S(u,v)$) denote the number of common neighbors of u and v in G (or in S). In a k-MDB $D = (U_D, V_D, E_D)$ of G, every edge $(u,v) \in E_D$ satisfies that (1) $d_D(u) \geq \theta - k$, and (2) for all $w \in N_D(v)$, $cn_D(u,w) \geq \theta - k$. By combining the two conditions, we can traverse all edges $(u,v) \in E$ before k-MDB search. If v has fewer than $\theta - k$ neighbors w such that $cn(u,w) \geq \theta - k$, the edge (u,v) can be pruned in G. Additionally, if the degree of u (or v) drops below $\theta - k$ due to the deletion of (u,v), the algorithm will also prune u (or v) from G.

One-non-neighbor reduction. For an instance (S,C) and a vertex u, let C_u denote the subset of C where all vertices are on the same side as u. If a branching vertex has only one non-neighbor in $S \cup C$, removing it from C can lead to the pruning of additional vertices. The following lemma explains this idea in detail.

Theorem 4.1. *Given an instance (S,C) and a vertex $u \in C$ with $\overline{d}_{S\cup C}(u) = 1$, removing u from C allows pruning every $v \in C_u$ with $\overline{d}_S(v) \geq 1$. Moreover, if $\overline{d}_C(u) = 1$ and w is the unique non-neighbor of u, then $\overline{N}_C(w)$ can also be pruned.*

Proof. Assume D is a k-MDB derived from (S,C) excluding u. For any $v \in C_u$ such that $v \in D$ and $\overline{d}_S(v) \geq 1$, $D \setminus \{v\} \cup \{u\}$ is also a k-MDB. When $\overline{d}_C(u) = 1$ and w is the unique non-neighbor of u, if $w \notin D$, then $D \cup \{u\}$ forms a larger k-MDB. Otherwise, for any $v \in \overline{N}_C(w)$ such that $v \in D$, $D \setminus \{v\} \cup \{u\}$ is also a k-MDB. □

Ordering-based reduction. We first extend Lemma 2.2 to constrain the vertex distance in k-MDB, as shown in the following lemma.

Lemma 4.2. *Given a k-MDB $D = (U_D, V_D, E_D)$, if $|U_D| > k$ and $|V_D| > k$, the distance between any two vertices of D is at most 3.*

Let $u_1, u_2, \ldots, u_{|U|}$ be an order of U. Let $N^2(u) = \{w \in N(v) \mid v \in N(u)\}$ denote the 2-hop neighbors of u, and let $N^2_+(u_i) = N^2(u_i) \cap \{u_{i+1}, u_{i+2}, \ldots, u_{|U|}\}$. By Lemma 4.2, the k-MDB containing u_i but no earlier vertex can be found from the instance $S_i = (\{u_i\}, \emptyset)$ and $C_i = (N^2_+(u_i), \bigcup_{u_j \in N^2_+(u_i)} N(u_j))$. The global k-MDB is the largest solution among these $|U|$ instances. In practice, we process U in descending degree order to find a strong incumbent early, and apply the common-neighbor reduction within each (S_i, C_i) to remove additional vertices and edges.

Progressive bounding reduction. Setting a small initial threshold θ (e.g. $k+1$) weakens the effectiveness of our optimization techniques. Inspired from the maximum biclique problem [36], we provide a series of tighter size thresholds for the k-MDB search problem. Specifically, let θ_U^t and θ_V^t denote the size thresholds in t-th round. The objective of t-th round is to identify a k-MDB $D_t = (U_{D_t}, V_{D_t}, E_{D_t})$ satisfying $|U_{D_t}| \geq \theta_U^t$ and $|V_{D_t}| \geq \theta_V^t$. Let $D^* = \max\{D_1, D_2, \ldots, D_{t-1}\}$ be the currently largest k-defective biclique. The progressive bounding reduction generates θ_U^t and θ_V^t in the following way:

$$\theta_U^t = \max\{\theta, \lfloor \theta_U^{t-1}/2 \rfloor\}, \quad \theta_V^t = \max\{\theta, \lfloor |E_{D^*}| / \theta_U^{t-1} \rfloor\}. \tag{6}$$

Specially, $\theta_U^0 = \max_{u \in U} d(u) + k$, and the iteration terminates when $\theta_U^t = \theta$. In many real-world graphs, θ_U^t and θ_V^t are often significantly larger than θ, which helps facilitate more effective graph reduction, such as a smaller reduced graph by performing the common-neighbor-based reduction.

Discussion. Although progressive bounding is adapted from other problems [48, 54, 57], our common-neighbor-based and one-non-neighbor reductions are technically distinct from prior approaches. Existing core-based reductions [54, 57] often provide limited pruning, while explicitly storing pairwise common-neighbor coexistence requires $O(n^2)$ space. Our common-neighbor rule combines degree and local coexistence conditions without this quadratic storage. The one-non-neighbor rule also differs from its variant for maximum defective clique search [13], where a vertex with exactly one non-neighbor can be safely inserted. This insertion is unsafe for k-MDB search because the objective depends on the two side sizes. For example, consider a 1-MDB instance (S,C) with $|U_S| = 2$, $|V_S| = 3$, $U_C = \{u\}$, and $V_C = \{v\}$, where $G(S)$ has one non-edge and u and v are non-neighbors. Adding v produces only 7 edges, whereas adding u produces 8. Our rule therefore deletes only replaceable candidates on the same side, as guaranteed by Theorem 4.1.

4.2 Novel Upper-Bounding Techniques

In this subsection, we propose a novel upper-bounding technique that provides effective upper bounds for both vertices and edges. Formally, given an instance (S,C), assume $u_1, u_2, \cdots, u_{|U_C|}$ and $v_1, v_2, \cdots, v_{|V_C|}$ are the vertices of U_C and V_C in ascending order of $\overline{d}_S(\cdot)$, respectively. Let $\overline{d}_S^i(U_C) = \sum_{t=1}^{i} \overline{d}_S(u_t)$ and $\overline{d}_S^i(V_C) = \sum_{t=1}^{i} \overline{d}_S(v_t)$, i.e., the total number of non-neighbors in S among the first i vertices of U_C and V_C. Based on this, the vertex upper bound is presented in the following lemma.

Lemma 4.3 (Vertex upper bounds). *For any k-MDB D derived from (S,C), we have $|U_D| \leq |U_S| + i$ and $|V_D| \leq |V_S| + j$, where i and j are the largest values satisfying $\overline{d}_S^i(U_C) \leq k - |\overline{E}_S|$ and $\overline{d}_S^j(V_C) \leq k - |\overline{E}_S|$, respectively.*

Algorithm 3: UpperBound($(S, C), k$)

Input: Instance $(S = (U_S, V_S), C = (U_C, V_C))$ and integer k
Output: two vertex upper bounds and an edge upper bound

1 $k' \leftarrow k - |\overline{E}_S|$;
2 Let $u_1, u_2, \cdots, u_{|U_C|}$ and $v_1, v_2, \cdots, v_{|V_C|}$ be the vertices of U_C and V_C in ascending order of $\overline{d}_S(\cdot)$;
3 $\overline{d}_U \leftarrow 0; \overline{d}_V \leftarrow 0; c_U \leftarrow 0; c_V \leftarrow 0$;
4 **for** $i \leftarrow 1 \ldots |U_C|$ *s.t.* $\overline{d}_U + \overline{d}_S(u_i) \leq k'$ **do**
5 $\overline{d}_U \leftarrow \overline{d}_U + \overline{d}_S(u_i)$;
6 $c_U \leftarrow c_U + 1$;
7 $e \leftarrow (|U_S| + c_U) \cdot |V_S| - |\overline{E}_S| - \overline{d}_U$;
8 $i \leftarrow c_U$;
9 **for** $j \leftarrow 1 \ldots |V_C|$ *s.t.* $\overline{d}_V + \overline{d}_S(v_j) \leq k'$ **do**
10 **while** $\overline{d}_U + \overline{d}_V + \overline{d}_S(v_j) > k'$ **do**
11 $\overline{d}_U \leftarrow \overline{d}_U - \overline{d}_S(u_i)$;
12 $i \leftarrow i - 1$;
13 $\overline{d}_V \leftarrow \overline{d}_V + \overline{d}_S(v_j)$;
14 $c_V \leftarrow c_V + 1$;
15 $e \leftarrow \max\{e, (|U_S| + i) \cdot (|V_S| + j) - |\overline{E}_S| - \overline{d}_U - \overline{d}_V\}$;
16 **return** $|U_S| + c_U, |V_S| + c_V, e$;

Proof. For the side of U, it is obvious that i is the maximum number of vertices that can be added from U_C to U_S such that $(U_S \cup \{u_1, u_2, \ldots, u_i\}, V_S)$ remains a valid k-defective biclique. For any k-MDB derived from (S, C) consisting of $(U_S \cup U_{C'}, V_S \cup V_{C'})$, $(U_S \cup U_{C'}, V_S)$ is still a k-defective biclique. Therefore, we have that $|U_{C'}| \leq i$. The proof on the other side is similar. □

Based on Lemma 4.3, an edge upper bound can be calculated as $(|U_S| + i) \times (|V_S| + j) - |\overline{E}_S| - \overline{d}_S^i(U_C) - \overline{d}_S^j(V_C)$. However, this upper bound does not account for the coexistence of vertices in U_C and V_C. To solve the issue, we present a tighter edge upper bound in the following lemma.

Lemma 4.4 (Edge upper bound). *For any k-MDB D derived from (S, C), we have that $|E_D| \leq \max_{0 \leq i \leq |U_C|}(|U_S| + i) \times (|V_S| + j) - |\overline{E}_S| - \overline{d}_S^i(U_C) - \overline{d}_S^j(V_C)$, where j is the largest value satisfying $\overline{d}_S^i(U_C) + \overline{d}_S^j(V_C) \leq k - |\overline{E}_S|$.*

Proof. Consider constructing (S, C') from (S, C) by fully connecting $U_{C'}$ and $V_{C'}$, i.e., $\overline{E}_{C'} = \emptyset$. The order of vertices in C' by $\overline{d}_S(\cdot)$ remains unchanged. A k-MDB derived from (S, C') consists of S, a prefix of $\{u_1, u_2, \ldots, u_{|U_C|}\}$, and a prefix of $\{v_1, v_2, \ldots, v_{|V_C|}\}$; otherwise, for any vertex not in a prefix, a preceding vertex can always be found to replace it. Moreover, the k-MDB derived from (S, C') has at least as many edges as any k-MDB derived from (S, C). Consequently, Lemma 4.4 provide a correct edge upper bound. □

Implementation details. Algorithm 3 presents a pseudocode of our upper-bounding techniques. It first computes the remaining number of missing edges k' and sorts U_C and V_C by $\overline{d}_S(\cdot)$ using bucket sort (lines 1–2). Then, it calculates the vertex upper bound in U by finding the longest prefix of U_C that can be added to U_S (lines 4–6), where c_U records the prefix length. After that, the algorithm derives the upper bound c_V for vertices in V using the same way, and simultaneously calculates the edge upper bound e by removing vertices from the end of the prefix of U_C while computing the longest prefix of V_C (lines 9–15). We present the time complexity of the algorithm in the following lemma.

Lemma 4.5. *The time complexity of Algorithm 3 is $O(n)$.*

4.3 A Heuristic Approach

We propose a heuristic approach to efficiently calculate a large k-defective biclique. We observe that in many real-world graphs, one side of the k-MDB with size threshold θ contains exactly θ vertices, while the other side contains much more than θ vertices. Motivated by this, our heuristic approach aims to find a large k-defective biclique with one side has θ vertices. Specifically, given a graph $G = (U, V, E)$ and an integer k, we initially set two vertex subset $U' = \emptyset$ and $V' = V$. let $u \in U \setminus U'$ be the vertex with largest $d_{V'}(u)$. If $G(U' \cup \{u\}, V')$ has more than k non-edges, we iteratively delete vertices $v \in V'$ with largest $\overline{d}_{U' \cup \{u\}}(v)$ until $G(U' \cup \{u\}, V')$ has no more than k non-edges, and then we add u to U'. We repeat this process until $|U'| = \theta$, and yield a large k-MDB $G(U', V')$ if $|V'| \geq \theta$. Otherwise, we yield an empty answer. It is easy to verify that this approach can be finished in $O(\theta n + m)$, which is linear as θ is always a small value.

4.4 Parallelization

To further enhance the scalability of our algorithms for handling larger-scale graphs, we implement parallelization strategy based on our proposed algorithms and optimization techniques. The parallelization of our branch-and-bound and pivoting-based algorithms is based on the search tree of the algorithm (as shown in Fig. 2(b) and Fig. 3(b)). Specifically, for a given branching strategy, once an instance (S, C) corresponding to a branch is determined, the sub-branches generated by this branch are uniquely defined and mutually independent. Therefore, we parallelize the processing of sub-branches at lower levels of the search tree. Additionally, the parallelization process is controlled by a global counter that tracks the total number of parallel tasks, and when the counter reaches the number of limited threads, the sub-branches generated by each branch are processed sequentially. Additionally, we also perform parallelization on the ordering-based reduction through the traversal of ordered vertices, and on the common-neighbor-based reduction through the traversal of all edges.

4.5 Optimized Algorithms

We apply all the proposed optimizations to our branch-and-bound algorithm and pivoting-based algorithm, which are denoted by MDBB and MDBP, respectively. An implementation of MDBP is presented in Algorithm 4. Specifically, it first obtains an initial solution using the heuristic approach (line 1) and then performs progressive bounding (lines 2–4), generating θ_U and θ_V via Eq. (6). In each round, it applies the parallel common-neighbor reduction to G (line 5) and the parallel ordering reduction (lines 6–10). Vertices in U are processed in descending degree order to construct instances (S, C), whose candidate sets are further reduced by Reduce using the common-neighbor rule (lines 9 and 13–14). The algorithm then invokes BranchP*, which augments BranchP in Algorithm 2 with graph reductions and the upper bounds in Algorithm 3. The largest solution returned by all invocations is output as D^* (line 11). MDBB is implemented analogously by replacing BranchP* with BranchB from Algorithm 1, equipped with the same optimizations.

Algorithm 4: MDBP

Input: A bipartite graph $G = (U, V, E)$, integers k, θ
Output: A k-MDB of G

$D^* \leftarrow$ Heuristic(G, k, θ);
$\theta_U \leftarrow \max_{u \in U} d(u) + k$;
while $\theta_U > \theta$ **do**
  $\theta_V \leftarrow \max\{\theta, \lfloor |E_{D^*}|/\theta_U \rfloor\}$; $\theta_U \leftarrow \max\{\theta, \lfloor \theta_U/2 \rfloor\}$;
  $G' = (U', V', E') \leftarrow$ **parallel** common-neighbor-based reduction on G;
  $\{u_1, u_2, \ldots, u_{|U'|}\} \leftarrow$ the descending degree order of U';
  parallel for $i \in 1 \ldots |U'|$ **do**
    $S \leftarrow (\{u_i\}, \emptyset)$; $C \leftarrow (U' \cap N_+^2(u_i),\ V' \cap \bigcup_{v \in N_+^2(u_i)} N(v))$;
    $C \leftarrow$ Reduce$(C, u_i, \theta_U, \theta_V)$;
    BranchP*(S, C); // BranchP equipped with graph reduction techniques, upper-bounding techniques and parallelization
return D^*;
Function Reduce$(C = (U_C, V_C), u, \theta_U, \theta_V)$:
  $U_{C'} \leftarrow \{v \in U_C \mid cn_{G'}(u, v) \geq \theta_V - k\}$;
  $V_{C'} \leftarrow \{v \in V_C \mid d_{G'}(v) \geq \theta_U - k\}$;
  return $(U_{C'}, V_{C'})$

5 EXPERIMENTS

5.1 Experimental Setup

Datasets. We collect 11 real-world bipartite graphs from diverse categories (http://konect.cc/networks, accessed January 2, 2026), which are widely used as benchmark datasets in other bipartite subgraph model problems [14, 36, 54]. The statistics of these graphs are shown in Table 1, where Δ and $\overline{d}$ represent the maximum degree and average degree of vertices on each side, respectively.

Algorithms. We implement two proposed algorithms MDBB and MDBP, corresponding to Algorithm 1 and Algorithm 2, with all optimization techniques in Sec. 4. We adapt 4 baseline algorithms from other problems: MDBC from the state-of-the-art maximum vertex defective biclique algorithm [52] with the maximum-vertex-based reductions removed, MDC from the maximum defective clique enumeration algorithm [13] by treating all vertices on the same side adjacent, MBP and MaxBC from the maximum biplex search algorithm [54] and maximal biclique enumeration algorithm [2] by reconstructing their branch-and-bound and pivoting techniques to suit our MDB problem. For fairness, these algorithms incorporate the optimization techniques outlined in Sec. 4.1 and 4.3. All algorithms are implemented in C++ and executed on a Linux machine with a 2.2GHz CPU and 128GB RAM.

5.2 Performance Studies

Exp–1: Efficiency comparison among different algorithms. We evaluate the efficiency of our algorithms, MDBP and MDBB, against four baselines: MDBC, MDC, MBP, and MaxBC, as shown in Table 2. Here, $|E^*|$ denotes the number of edges in k-MDB, and "-" indicates a timeout exceeding 3 hours. As observed, MDBP is up to two orders of magnitude faster than MDBC and solves nearly all instances with $k \geq 3$, for which the baselines frequently time out. For example, on *LKML* with $k = 3$ and $\theta = 7$, MDBP completes in only 9.53s, yielding a $\approx$ 1000× speedup over the fastest baseline MaxBC, which takes 9, 223s. The two proposed algorithms MDBP and MDBB are comparable when $k \leq 2$ and $\theta \geq 5$, but MDBP becomes substantially faster when $k \geq 3$ or $\theta \leq 4$. For instance, on *Twitter* with $k = 4, \theta = 5$, MDBB takes 3, 550s while MDBP finished in only 21s, achieving a speedup of over 160×. These results show the superiority of our pivoting-based branching strategy in Sec. 3.2. Moreover, MDBB consistently outperforms all baselines, validating the effectiveness of our binary branching strategy in Sec. 3.1.

Table 1: Datasets used in experiments

Dataset	$\lvert U\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	Δ	$\overline{d}$
LKML	42,045	337,509	599,858	(31,719, 6,627)	(14, 1)
Team	901,130	34,461	1,366,466	(17, 2,671)	(1, 39)
Mummun	175,214	530,418	1,890,661	(968, 19805)	(10, 3)
Citeu	153,277	731,769	2,338,554	(189,292, 1,264)	(15, 3)
IMDB	303,617	896,302	3,782,463	(1,334, 1,590)	(12, 4)
Enwiki	18,038	2,190,091	4,129,231	(324,254, 1,127)	(228, 1)
Amazon	2,146,057	1,230,915	5,743,258	(12,180, 3,096)	(2, 4)
Twitter	244,537	9,129,669	10,214,177	(640, 18,874)	(41, 1)
Aol	4,811,647	1,632,788	10,741,953	(100,629, 84,530)	(2, 6)
Google	5,998,790	4,443,631	20,592,962	(423, 95,165)	(3, 4)
LiveJournal	3,201,203	7,489,073	112,307,385	(300, 1,053,676)	(35, 14)

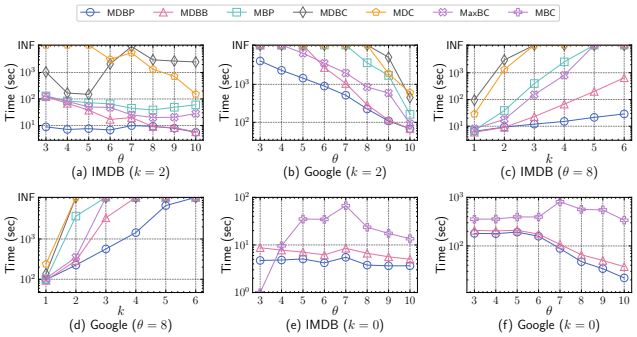


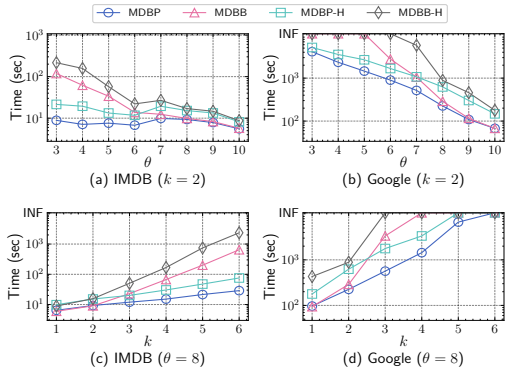
Figure 4: Runtime with varying θ and k

Exp–2: Runtime with varying θ and k. We evaluate MDBB and MDBP, along with 4 baseline algorithms, with varying θ and k on two representative datasets, *IMDB* and *Google*. The results are shown in Fig. 4, where "INF" denotes a timeout over 3 hours. As observed, both MDBP and MDBB outperform baseline algorithms, with MDBP achieving up to a 500× speedup. For example, in Fig. 4(c), MDBP complete in 20s at $\theta = 7$, while the baselines time out. When $\theta \geq 8$ or $k \leq 2$, MDBP and MDBB have similar runtimes, demonstrating the efficacy of our optimization techniques in reducing graph size. As k increases or θ decreases, the runtime of MDBP grows more slowly than those of the other algorithms, indicating lower sensitivity to parameter changes and better scalability. We further set $k = 0$ to evaluate MDBB and MDBP for maximum biclique search and compare them with the state-of-the-art algorithm MBC [36], as shown in Figs. 4(e) and 4(f). Both algorithms outperform MaxBC because they have better time complexities than its $O^*(2^n)$ complexity and use stronger heuristic and graph reduction techniques.

Exp–3: Efficacy of the heuristic approach. To evaluate the effectiveness of our proposed heuristic approach, we compare MDBB and MDBP with their versions without the heuristic approach, referred to as MDBB-H and MDBP-H, respectively. The results are shown in Fig. 5. As observed, MDBB and MDBP always outperform MDBB-H and MDBP-H, respectively. This is because the large solution provided

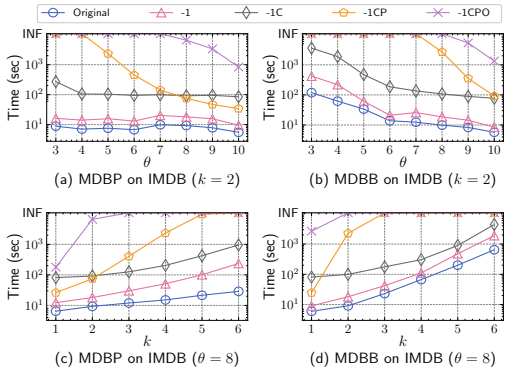
Table 2: Results of runtime among different algorithms on 10 real-world graphs (in seconds)

Dataset	k	θ	$\lvert E^*\rvert$	MDBP	MDBB	MDBC	MDC	MBP	MaxBC	k	θ	$\lvert E^*\rvert$	MDBP	MDBB	MDBC	MDC	MBP	MaxBC	k	θ	$\lvert E^*\rvert$	MDBP	MDBB	MDBC	MDC	MBP	MaxBC
LKML	1	2	1,781	**13**	17	-	-	42	229	2	3	460	**44**	-	-	-	-	-	3	4	149	**267**	-	-	-	-	-
		5	84	**1.61**	2.15	32	28	3.19	5.57		6	70	**7.60**	30	4,647	-	3,024	364		7	60	**9.53**	37	-	-	-	9,223
	4	7	66	**103**	1,216	-	-	-	-	5	8	67	**107**	1,224	-	-	-	-	6	9	0	**248**	1,732	-	-	-	-
		8	60	**9.36**	46	-	-	-	-		9	0	**13**	52	-	-	-	-		10	0	**12**	46	-	-	-	-
Team	1	2	701	**19**	39	-	-	48	240	2	3	265	**45**	-	-	-	-	1,783	3	4	149	**154**	-	-	-	-	-
		5	99	**7.14**	7.19	17	66	13	8.67		6	64	**9.53**	14	1,578	500	-	69		7	0	**10**	15	2,473	-	1,578	429
	4	5	111	**591**	-	-	-	-	-	5	6	79	**2,684**	-	-	-	-	-	6	8	0	**927**	-	-	-	-	-
		8	0	**9.92**	15	1,516	-	-	3,568		9	0	**10**	16	1,830	-	-	-		10	0	**7.65**	**11**	2,059	-	-	-
Mummun	1	2	10,315	**18**	19	-	-	36	286	2	3	8,803	**23**	1,831	-	-	8,519	1,309	3	4	4,185	**40**	-	-	-	-	-
		5	2,469	**13**	14	331	-	17	22		6	1,768	**21**	37	-	-	46	38		7	1,145	**33**	916	-	-	4,685	1,077
	4	5	2,481	**45**	-	-	-	-	-	5	6	1,783	**69**	-	-	-	-	-	6	7	1,163	**161**	-	-	-	-	-
		8	878	**38**	5,539	-	-	-	8,031		9	886	**41**	6,737	-	-	-	-		10	804	**53**	-	-	-	-	-
Citeu	1	2	12,425	**141**	147	-	-	1,796	3,962	2	3	12,427	**148**	188	-	-	1,841	-	3	4	9,989	**137**	4,333	-	-	-	9,183
		5	7,394	**5.48**	5.69	8,174	-	19	47		6	4,960	**7.88**	77	-	-	4,257	87		7	3,063	**18**	3,501	-	-	-	443
	4	5	7,406	**142**	-	-	-	-	-	5	6	4,975	**158**	-	-	-	-	-	6	7	3,081	**253**	-	-	-	-	-
		8	2,600	**26**	-	-	-	-	609		9	2,600	**36**	-	-	-	-	-		10	2,614	**134**	-	-	-	-	-
IMDB	1	2	775	5.99	**5.39**	447	-	17	60	2	3	562	**8.86**	17	1,041	-	30	126	3	4	522	**16**	436	7,924	-	-	158
		5	514	6.12	**5.35**	7.74	23	6.93	11		6	514	**6.76**	6.91	2,096	1,123	27	17		7	482	**15**	25	2,504	-	280	24
	4	5	526	**20**	594	-	-	-	267	5	6	529	**31**	684	-	-	-	901	6	7	498	**57**	7,936	-	-	-	-
		8	436	**15**	67	-	-	2,632	123		9	436	**19**	171	-	-	-	8,211		10	426	**23**	403	-	-	-	-
Enwiki	1	2	203,569	**3,630**	-	-	-	-	-	2	3	133,759	**4,978**	-	-	-	-	-	3	4	80,461	**3,260**	-	-	-	-	-
		5	21,939	**157**	159	965	387	378	457		6	4,240	**200**	583	-	-	-	-		7	3,224	**264**	-	-	-	-	3,382
	4	5	21,951	**3,826**	-	-	-	-	-	5	6	4,255	**5,744**	-	-	-	-	-	6	8	2,330	**6,558**	-	-	-	-	-
		8	2,316	**340**	-	-	-	-	-		9	1,696	**395**	-	-	-	-	-		10	1,274	**640**	-	-	-	-	-
Amazon	1	2	1,319	**43**	45	2,698	-	51	407	2	3	427	**57**	646	-	-	1,810	1,029	3	4	423	**89**	1,196	-	-	-	5,720
		5	413	12	**11**	14	50	16	17		6	418	13	**12**	38	-	20	29		7	357	**13**	15	1,184	-	629	155
	4	5	428	**100**	850	-	-	-	-	5	6	433	**250**	-	-	-	-	-	6	7	390	**268**	-	-	-	-	-
		8	368	**14**	16	-	-	-	-		9	379	**16**	20	-	-	-	-		10	390	**19**	26	-	-	-	-
Twitter	1	2	5,369	**11**	13	-	-	82	95	2	3	5,383	**11**	51	-	-	282	132	3	4	5,397	**15**	321	-	-	5,259	337
		5	5,369	**6.27**	7.60	1,229	8,976	15	33		6	5,383	**6.63**	18	6,693	-	87	64		7	5,397	**9.76**	54	-	-	751	158
	4	5	5,411	**21**	3,550	-	-	-	-	5	6	5,425	**31**	-	-	-	-	-	6	7	5,439	**58**	-	-	-	-	-
		8	5,411	**13**	292	-	-	-	597		9	5,425	**24**	3,268	-	-	-	6,206		10	5,439	**44**	9,895	-	-	-	-
Aol	1	2	10,869	1,033	**971**	-	-	1,427	-	2	3	3,208	**1,636**	-	-	-	-	-	3	4	853	**3,123**	-	-	-	-	-
		5	334	50	**49**	214	5,703	227	336		6	238	**94**	182	-	-	6,458	-		7	207	**123**	409	-	-	-	-
	4	5	346	**9,272**	-	-	-	-	-	5	7	219	**2,388**	-	-	-	-	-	6	9	174	-	-	-	-	-	-
		8	188	**470**	1,011	-	-	-	-		9	166	**4,497**	5,116	-	-	-	-		10	164	**5,585**	8,636	-	-	-	-
Google	1	2	10,829	**2,163**	2,526	-	-	2,582	-	2	3	3,370	**4,121**	-	-	-	-	-	3	4	1,201	**6,250**	-	-	-	-	-
		5	439	**275**	376	1,858	-	353	362		6	184	**898**	2,772	-	-	-	-		7	151	**1,351**	-	-	-	-	-
	4	7	157	**5,331**	-	-	-	-	-	5	8	155	**6,750**	-	-	-	-	-	6	9	156	**8,756**	-	-	-	-	-
		9	140	**613**	9,387	-	-	-	-		10	145	**684**	-	-	-	-	-		10	145	**1,978**	-	-	-	-	-



(a) IMDB ($k = 2$) (b) Google ($k = 2$) (c) IMDB ($\theta = 8$) (d) Google ($\theta = 8$)

Figure 5: Runtime without using the heuristic approach



(a) MDBP on IMDB ($k = 2$) (b) MDBB on IMDB ($k = 2$) (c) MDBP on IMDB ($\theta = 8$) (d) MDBB on IMDB ($\theta = 8$)

Figure 6: Runtime without graph reduction techniques

by our heuristic approach enhances the effectiveness of our upper-bounding techniques. Additionally, we observe that MDBP-H still performs better than MDBB when $\theta \leq 6$ or $k \geq 3$, highlighting the efficiency of our pivoting techniques.

Exp–4: Efficacy of graph reduction techniques. We evaluate the graph reduction techniques by implementing 4 ablation variants of MDBP and MDBB, obtained by successively removing the one-non-neighbor reduction, common-neighbor-based reduction, progressive bounding reduction and ordering-based reduction. The results are presented in Fig. 6, which label the original algorithm as "Original", and the 4 ablation versions as "-1", "-1C", "-1CP", "-1CPO". As observed, all graph reduction techniques contribute to efficiency improvements. The one-non-neighbor and common-neighbor reductions provide stable gains across different θ and k, whereas progressive bounding and ordering become more effective as θ decreases or k increases. For $k = 1$ or $\theta = 10$, removing progressive bounding can even improve performance, indicating that its overhead may exceed its pruning benefit on easier instances.

Exp–5: Efficacy of upper-bounding techniques. We evaluate the upper-bounding techniques by comparing MDBB and MDBP with

Table 3: Runtime on synthetic graphs (in seconds)

Distribution	ρ	Δ	$\overline{d}$	$k=1,\theta=2$			$k=3,\theta=4$			$k=5,\theta=6$		
				$\lvert E^*\rvert$	MDBP	MDBB	$\lvert E^*\rvert$	MDBP	MDBB	$\lvert E^*\rvert$	MDBP	MDBB
Power-law	0.2	(97, 82)	(19, 19)	197	**0.10**	0.18	181	**0.29**	0.96	175	**8.03**	11.82
	0.3	(77, 99)	(29, 29)	269	**0.11**	0.33	277	**0.56**	9.52	292	**16.02**	133.29
	0.4	(95, 100)	(36, 36)	349	**0.27**	**0.51**	357	**1.76**	26.01	375	**33.84**	432.58
	0.5	(100, 99)	(45, 45)	482	**0.92**	**1.94**	494	**5.83**	105.49	506	**189.92**	2,030.16
Normal	0.2	(29, 29)	(17, 17)	27	**0.26**	0.47	29	**31.74**	75.84	32	**59.74**	153.87
	0.3	(43, 43)	(29, 29)	44	**1.24**	3.04	45	**179.49**	397.23	49	**312.55**	1,225.26
	0.4	(48, 51)	(39, 39)	56	**3.78**	10.73	67	**489.87**	1,015.43	65	**1,300.01**	-
	0.5	(62, 61)	(49, 49)	65	**15.03**	43.81	65	**2,095.58**	-	-	-	-

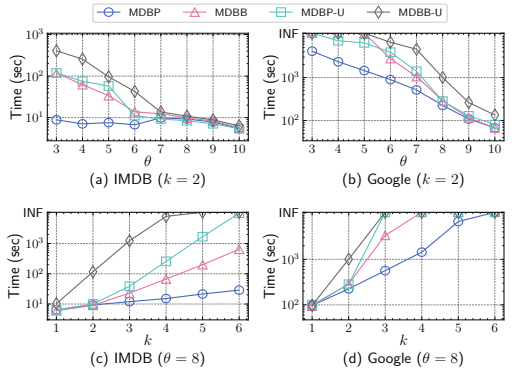


Figure 7: Runtime without upper-bounding techniques

Table 4: Runtime of parallelized algorithms (in seconds)

Dataset	k	θ	MDBP with threads				MDBB with threads			
			1	10	20	30	1	10	20	30
Google	1	2	2,163	425	56	54	3,052	870	223	224
	3	4	6,250	1,324	722	795	-	-	10,649	10,413
	5	6	-	-	8,947	9,062	-	-	-	-
LiveJournal	1	2	119	89	47	44	-	-	9,314	9,335
	3	4	3,996	997	850	845	-	-	-	-
	5	6	-	-	10,016	9,943	-	-	-	-

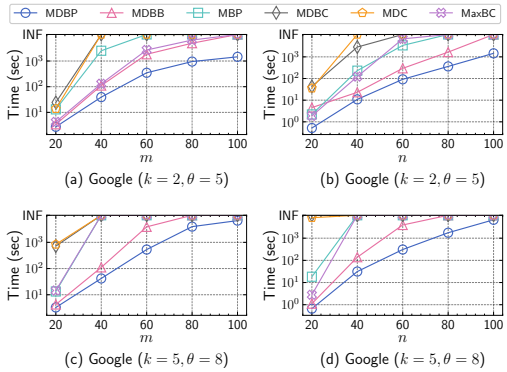


Figure 8: Scalability of different algorithms

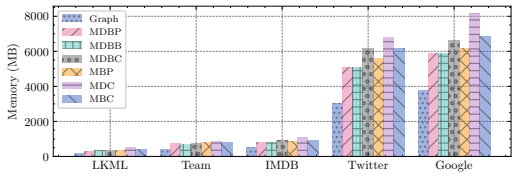


Figure 9: Memory usage of various algorithms

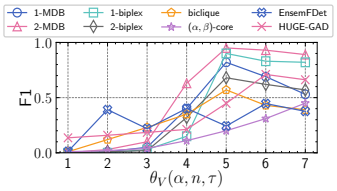
their variants without Algorithm 3, denoted by MDBB−U and MDBP−U, respectively. The results are shown in Fig. 7. As observed, the upper bounds significantly improve the efficiency of both MDBB and MDBP, especially as k increases or θ decreases, indicating the strong effectiveness and efficiency of our proposed upper-bounding techniques. Additionally, the improvement of MDBP by performing upper-bounding techniques is more substantial compared to MDBB, because the pivoting-based branching strategy introduces more non-edges to the partial set S earlier, resulting in tighter vertex and edge upper bounds derived from the upper-bounding techniques, which helps prune more unnecessary instances.

Exp–6: Performance on synthetic graphs. To discover the relationship between algorithm efficiency and graph properties, we evaluate MDBB and MDBP on 8 synthetic graphs with 100 vertices per side, using either power-law or normal degree distribution, with density $\rho = \frac{|E|}{|U|\cdot|V|}$ varies from 0.2 to 0.5. The results are presented in Table 3. As observed, both algorithms are consistently efficient on power-law graphs, while their performance on normal graphs is more affected by ρ and k. This is because (1) power-law graphs exhibit a larger gap between Δ and $\overline{d}$, which benefits our branching strategies and graph reduction techniques, and (2) higher number of k-MDB edges in power-law graphs benefits our upper bound techniques. This explains why our approaches are highly efficient on large-scale real-world graphs with significantly gaps between Δ and $\overline{d}$ or lots of edges in k-MDB, as shown in Table 1 and 2.

Exp–7: Scalability testing. To evaluate the scalability of algorithms, we test the runtime of MDBB, MDBP, MDBC, MDC, MBP, MaxBC on *Google* using random sampling of vertices or edges. Results on the other datasets are consistent. We prepare 8 proportionally sampled subgraphs of *Google* by sampling 20%, 40%, 60% and 80% vertices or edges from the original graph. The results are shown in Fig. 8. As observed, MDBP shows the slowest runtime growth as the graph size increases, while MDBC and MDC exhibit the steepest rise. This result indicates that MDBP achieves the best scalability among all the tested algorithms. Furthermore, regardless of graph size, both MDBP and MDBB significantly outperform other algorithms, indicating the advancement of our approach.

Exp–8: Efficiency of parallelized algorithms. We evaluate parallel MDBB and MDBP on two challenging large-scale graphs *Google* and *LiveJournal*, with varying the number of threads as 1, 10, 20 and 30, which also serves as the threshold of global counter. The results are shown in Table 4. As observed, the runtime of MDBP and MDBB consistently decreases as the number of threads increases to 20, achieving speedups of up to 40× on *Google* ($k=1$, $\theta=2$) and 10× on *LiveJournal* ($k=5$, $\theta=6$). The improvement becomes marginal beyond 20 threads, making 20 threads a practical choice. Notably, parallelization solves the challenging $k=5, \theta=6$ instances for which the sequential algorithms time out after three hours.

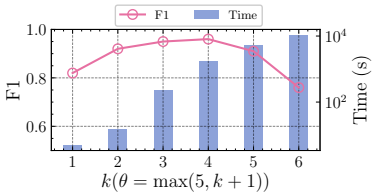
Exp–9: Memory usage. We evaluate the maximum physical memory usage (in MB) of the algorithms MDBB, MDBP, MDBC, MBP, MDC and MaxBC on five representative graphs, with the parameters $k=2$ and $\theta=3$. Similar trends hold for other datasets and parameters.



(a) F1-score with varying parameters

Method	F1	Modularity	Time(s)
1-MDB	0.82	0.33	0.30
2-MDB	**0.95**	**0.45**	15.44
1-biplex	0.92	0.37	35.15
2-biplex	0.78	0.24	1187.82
biclique	0.57	0.15	0.23
(α,β)-core	0.45	0.04	0.14
EnsemFDet	0.45	0.17	59.60
HUGE-GAD	0.71	0.20	882.67

(b) Optimal performance



(c) Performance of k-MDB with varying k

Figure 10: Performance on fraud detection task

The results are shown in Fig. 9. As observed, all algorithms except MDC and MaxBC consume less than twice the original graph's memory. This is mainly because MDC adds massive intra-side edges, and MaxBC maintains an extra DAG structure during pivoting. These results indicate that our proposed algorithms are space efficient.

5.3 Case Study

Case–1: Fraud Detection. We simulate a camouflage attack following [24] on the *Appliance* dataset (https://amazon-reviews-2023.github.io/, accessed January 2, 2026), which contains 602,777 reviews by 515,650 users on 30,459 products. We inject 500 fake users and 500 fake products, connect them with 10,000 fake reviews, and add 10,000 camouflage reviews from fake users to real products. We adapt MDBP as in [54] to extract the top 2,000 k-MDBs and flag all contained users and products as fake. We compare with maximum biclique [36], maximum k-biplex [54], (α,β)-core [7], the densest-subgraph method EnsemFDet [42], and the GNN-based HUGE-GAD [39]; we also improve the performance of EnsemFDet with Spade and Dupin parallel frameworks [25, 26]. The thresholds θ_U and θ_V constrain user and product side sizes for k-MDB, maximum biclique and maximum k-biplex. We evaluated F1-scores ($\frac{2\cdot\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$) with varying θ_V, the core threshold α, the sampled block count n of EnsemFDet, and the shared-product threshold τ of HUGE-GAD from 1 to 7, while fixing $\theta_U = \beta = 4$. As shown in Fig. 10(a), 2-MDB performs best when $\theta_V \geq 5$. Increasing k from 1 to 2 improves k-MDB but weakens k-biplex, showing that k-MDB provides a more suitable relaxation. When $\theta_V \leq 3$, biclique, EnsemFDet, and HUGE-GAD perform better because k-MDB and k-biplex produce overly sparse structures. However, biclique loses recall for $\theta_V \geq 5$ due to its strict definition, while the k-MDB and k-biplex maintain high F1-scores. We further assess runtime and subgraph quality using the widely adopted modularity ($\frac{1}{2m}\sum_{u\in U_S, v\in V_S}(A_{uv} - \frac{d(u)d(v)}{2m})$), which measures internal connectivity relative to a random graph. As shown in Fig. 10(b), 2-MDB obtains the highest modularity and F1-score, indicating that its cohesive structure better matches the fraud block. Although 1-biplex attains a similar F1-score, it is significantly slower than 2-MDB. To determine the optimal value of k, we evaluate k-MDB with varying k from 1 to 6. As shown in Fig.10(c), setting $k = 4$ gives the highest F1-score of 0.97 but takes about 30 minutes, making it appropriate for offline analysis. Setting $k = 2$ reduces F1-score only to 0.95 while finishing in about 15s, providing a practical choice for real-time detection.

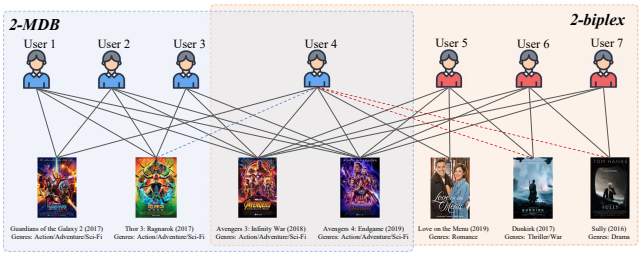


Figure 11: Recommendation results on *MovieLens*

Case–2: Online Recommendation. We evaluate k-MDB in an online recommendation task using the *MovieLens* dataset (https://grouplens.org/datasets/movielens/32m/, accessed March 19, 2026), which contains 32 million ratings applied to 87,585 movies by 200,948 users. We retain movies released since 2016 and represent 5.0 ratings as edges. We leverage the non-edges within the 2-MDB and maximum 2-biplex to formulate recommendations, as illustrated in Fig. 11. Specifically, the 2-MDB identifies a cohesive user community centered around "Action/Adventure/Sci-Fi" genres, and recommends a genre-matched movie, "Thor: Ragnarok", to User 4 (blue dashed line). In contrast, the maximum 2-biplex recommends two unrelated movies "Dunkirk" and "Sully" to User 4 (red dashed line). This is mainly because the 2-biplex's degree-based relaxation may form sparse user groups linked only by mainstream films (e.g., The Avengers), introducing unrelated recommendations. In contrast, 2-MDB's global-density constraint identifies cohesive communities and unobserved, genre-consistent movies.

6 CONCLUSION

In this paper, we investigate the problem of maximum k-defective biclique search on bipartite graphs. We propose two novel algorithms to tackle the problem. The first algorithm employs a new binary branching strategy that reduces the time complexity to below $O^*(2^n)$, while the second algorithm utilizes a novel pivoting-based branching strategy, achieving a notably lower time complexity. Additionally, we propose a series of nontrivial optimization techniques to further improve the efficiency of our algorithms. Extensive experiments conducted on real-world graphs demonstrate the notable efficiency of our algorithms and optimization techniques, and two case studies validate the effectiveness of maximum k-defective biclique model in real-world applications.

ACKNOWLEDGMENTS

This work was partially supported by (1) the New Generation Artificial Intelligence-National Science and Technology Major Project 2025ZD0123403, (2) NSFC Grants 62394332, 62394333, 62402044, 62572055, 62372342 and (3) the Beijing Institute of Technology Research Fund Program for Young Scholars. Rong-Hua Li is the corresponding author of this paper.

REFERENCES

[1] Online. *Maximum Defective Biclique Search in Large Bipartite Graphs (full version).* https://github.com/pluv1a/MaximumDefectiveBiclique/blob/main/docs/defbiclique-full.pdf

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

[3] Gabriela Alexe, Sorin Alexe, Yves Crama, Stephan Foldes, Peter L Hammer, and Bruno Simeone. 2004. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics* 145, 1 (2004), 11–21.

[4] Taher Alzahrani and Kathy Horadam. 2019. Finding maximal bicliques in bipartite networks using node similarity. *Applied Network Science* 4, 1 (2019), 21.

[5] Christoph Ambühl, Monaldo Mastrolilli, and Ola Svensson. 2011. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM J. Comput.* 40, 2 (2011), 567–596.

[6] Fabian Braun, Olivier Caelen, Evgueni N. Smirnov, Steven Kelk, and Bertrand Lebichot. 2017. Improving Card Fraud Detection Through Suspicious Pattern Discovery. In *Advances in Artificial Intelligence: From Theory to Practice*, Salem Benferhat, Karim Tabia, and Moonis Ali (Eds.). Springer International Publishing, 181–190.

[7] Monika Cerinšek and Vladimir Batagelj. 2015. Generalized two-mode cores. *Social Networks* 42 (2015), 80–87.

[8] Lijun Chang. 2023. Efficient Maximum k-Defective Clique Computation with Improved Time Complexity. *Proceedings of the ACM on Management of Data* 1, 3 (2023), 1–26.

[9] Lijun Chang. 2024. Maximum Defective Clique Computation: Improved Time Complexities and Practical Performance. *arXiv preprint arXiv:2403.07561* (2024).

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

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

[12] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. 2021. Computing maximum k-defective cliques in massive graphs. *Computers & Operations Research* 127 (2021), 105131.

[13] Qiangqiang Dai, Ronghua Li, Donghang Cui, and Guoren Wang. 2024. Theoretically and Practically Efficient Maximum Defective Clique Search. *Proceedings of the ACM on Management of Data* 2, 4 (2024), 1–27.

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

[15] Milind Dawande, Pinar Keskinocak, Jayashankar M. Swaminathan, and Sridhar R. Tayur. 2001. On Bipartite and Multipartite Clique Problems. *J. Algorithms* 41, 2 (2001), 388–403.

[16] 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.

[17] Qilong Feng, Shaohua Li, Zeyang Zhou, and Jianxin Wang. 2018. Parameterized algorithms for edge biclique and related problems. *Theoretical Computer Science* 734 (2018), 105–118.

[18] Fedor V Fomin and Petteri Kaski. 2013. Exact exponential algorithms. *Commun. ACM* 56, 3 (2013), 80–88.

[19] Jian Gao, Zhenghang Xu, Ruizhi Li, and Minghao Yin. 2022. An exact algorithm with new upper bounds for the maximum k-defective clique problem in massive sparse graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 10174–10183.

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

[21] Nicolas Gillis and François Glineur. 2014. A continuous characterization of the maximum-edge biclique problem. *Journal of Global Optimization* 58, 3 (2014), 439–464.

[22] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. 2018. Maximum weight relaxed cliques and Russian Doll Search revisited. *Discrete Applied Mathematics* 234 (2018), 131–138.

[23] Bryan Hooi, Kijung Shin, Hyun Ah Song, Alex Beutel, Neil Shah, and Christos Faloutsos. 2017. Graph-Based Fraud Detection in the Face of Camouflage. *ACM Trans. Knowl. Discov. Data* 11, 4 (2017), Article 44.

[24] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 895–904.

[25] Jiaxin Jiang, Yuan Li, Bingsheng He, Bryan Hooi, Jia Chen, and Johan Kok Zhi Kang. 2022. Spade: A real-time fraud detection framework on evolving graphs. *Proceedings of the VLDB Endowment* 16, 3 (2022), 461–469.

[26] Jiaxin Jiang, Siyuan Yao, Yuchen Li, Qiange Wang, Bingsheng He, and Min Chen. 2025. Dupin: A parallel framework for densest subgraph discovery in fraud detection on massive graphs. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.

[27] Mingming Jin, Jiongzhi Zheng, and Kun He. 2024. KD-Club: An Efficient Exact Algorithm with New Coloring-based Upper Bound for the Maximum k-Defective Clique Problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 20735–20742.

[28] MA Langston, Elissa J Chesler, and Y Zhang. 2008. On finding bicliques in bipartite graphs: a novel algorithm with application to the integration of diverse biological data types. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)(HICSS)*, Vol. 1. 473.

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

[30] Michael Ley. 2002. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives *(String Processing and Information Retrieval)*. Springer Berlin Heidelberg, 1–10.

[31] Jingdong Li, Zhao Li, Xiaoling Wang, Xingjian Lu, Ji Zhang, and Hongyang Chen. 2023. GPU-Accelerated Maximal Bicliques Mining Framework for Large E-commerce Networks. In *2023 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 539–544.

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

[33] Jinyan Li, Kelvin Sim, Guimei Liu, and Limsoon Wong. 2008. Maximal quasi-bicliques with balanced noise tolerance: Concepts and co-clustering applications. In *Proceedings of the 2008 SIAM International Conference on Data Mining*. SIAM, 72–83.

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

[35] Chunyu Luo, Yi Zhou Zhengren Wang, and Mingyu Xiao. 2024. A Faster Branching Algorithm for the Maximum k-Defective Clique Problem. *arXiv preprint arXiv:2407.16588* (2024).

[36] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum biclique search at billion scale. *Proceedings of the VLDB Endowment* 13, 9 (May 2020), 1359–1372. https://doi.org/10.14778/3397230.3397234

[37] Sara C Madeira and Arlindo L Oliveira. 2004. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM transactions on computational biology and bioinformatics* 1, 1 (2004), 24–45.

[38] Pasin Manurangsi. 2018. Inapproximability of Maximum Biclique Problems, Minimum k-Cut and Densest At-Least-k-Subgraph from the Small Set Expansion Hypothesis. *Algorithms* 11, 1 (2018), 10.

[39] Junjun Pan, Yixin Liu, Xin Zheng, Yizhen Zheng, Alan Wee-Chung Liew, Fuyi Li, and Shirui Pan. 2025. A label-free heterophily-guided approach for unsupervised graph fraud detection. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence (AAAI'25/IAAI'25/EAAI'25)*. AAAI Press, Article 1383, 9 pages. https://doi.org/10.1609/aaai.v39i12.33356

[40] René Peeters. 2003. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics* 131, 3 (2003), 651–654.

[41] 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.

[42] Yuxiang Ren, Hao Zhu, Jiawei Zhang, Peng Dai, and Liefeng Bo. 2021. EnsemFDet: An Ensemble Approach to Fraud Detection based on Bipartite Graph. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 2039–2044. https://doi.org/10.1109/ICDE51399.2021.00197

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

[44] Shahram Shahinpour, Shirin Shirvani, Zeynep Ertem, and Sergiy Butenko. 2017. Scale reduction techniques for computing maximum induced bicliques. *Algorithms* 10, 4 (2017), 113.

[45] Vladimir Stozhkov, Austin Buchanan, Sergiy Butenko, and Vladimir Boginski. 2022. Continuous cubic formulations for cluster detection problems in networks. *Mathematical Programming* 196, 1 (2022), 279–307.

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

[47] Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. 2013. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization*

and Applications 56 (2013), 113–130.

[48] Jianhua Wang, Jianye Yang, Ziyi Ma, Chengyuan Zhang, Shiyu Yang, and Wenjie Zhang. 2023. Efficient Maximal Biclique Enumeration on Large Uncertain Bipartite Graphs. *IEEE Trans. on Knowl. and Data Eng.* 35, 12 (2023), 12634–12648. https://doi.org/10.1109/tkde.2023.3272110

[49] Kai Wang, Yiheng Hu, Xuemin Lin, Wenjie Zhang, Lu Qin, and Ying Zhang. 2021. A Cohesive Structure Based Bipartite Graph Analytics System. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) *(CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 4799–4803. https://doi.org/10.1145/3459637.3481963

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

[51] Yiyuan Wang, Shaowei Cai, and Minghao Yin. 2018. New heuristic approaches for maximum balanced biclique problem. *Information Sciences* 432 (2018), 362–375.

[52] Zhiyi Wang, Lijun Chang, and Jeffrey Xu Yu. 2025. Identifying Maximum Defective Bicliques in Large Bipartite Graphs. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. 3710–3723. https://doi.org/10.1109/ICDE65448.2025.00277

[53] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. 2006. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics* 22, 7 (2006), 823–829.

[54] 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), Article 49.

[55] Kaiqiang Yu, Cheng Long, Shengxin Liu, and Da Yan. 2022. Efficient Algorithms for Maximal k-Biplex Enumeration. In *Proceedings of the 2022 International Conference on Management of Data*. 860–873.

[56] Bo Yuan, Bin Li, Huanhuan Chen, and Xin Yao. 2015. A New Evolutionary Algorithm with Structure Mutation for the Maximum Balanced Biclique Problem. *IEEE Transactions on Cybernetics* 45, 5 (2015), 1054–1067.

[57] Hongru Zhou, Shengxin Liu, and Ruidi Cao. 2025. Efficient Maximum Vertex (k,l)-Biplex Computation on Bipartite Graphs. *Tsinghua Science and Technology* 30, 2 (2025), 569–584. https://doi.org/10.26599/TST.2024.9010009

[58] Yi Zhou and Jin-Kao Hao. 2019. Tabu search with graph reduction for finding maximum balanced bicliques in bipartite graphs. *Eng. Appl. Artif. Intell.* 77 (2019), 86–97.