The VLDB Journal (2022) 31:1365–1389
https://doi.org/10.1007/s00778-021-00681-6

SPECIAL ISSUE PAPER

Maximum and top-*k* diversified biclique search at scale

Bingqing Lyu[1] · Lu Qin[2] · Xuemin Lin[3] · Ying Zhang[2] · Zhengping Qian[1] · Jingren Zhou[1]

Received: 30 November 2020 / Revised: 14 May 2021 / Accepted: 4 June 2021 / Published online: 18 April 2022

Abstract
Maximum biclique search, which finds the biclique with the maximum number of edges in a bipartite graph, is a fundamental problem with a wide spectrum of applications in different domains, such as E-Commerce, social analysis, web services, and bioinformatics. Unfortunately, due to the difficulty of the problem in graph theory, no practical solution has been proposed to solve the issue in large-scale real-world datasets. Existing techniques for maximum clique search on a general graph cannot be applied because the search objective of maximum biclique search is two-dimensional, i.e., we have to consider the size of both parts of the biclique simultaneously. In this paper, we divide the problem into several subproblems each of which is specified using two parameters. These subproblems are derived in a progressive manner, and in each subproblem, we can restrict the search in a very small part of the original bipartite graph. We prove that a logarithmic number of subproblems is enough to guarantee the algorithm correctness. To minimize the computational cost, we show how to reduce significantly the bipartite graph size for each subproblem while preserving the maximum biclique satisfying certain constraints by exploring the properties of one-hop and two-hop neighbors for each vertex. Furthermore, we study the diversified top-k biclique search problem which aims to find k maximal bicliques that cover the most edges in total. The basic idea is to repeatedly find the maximum biclique in the bipartite graph and remove it from the bipartite graph k times. We design an efficient algorithm that considers to share the computation cost among the k results, based on the idea of deriving the same subproblems of different results. We further propose two optimizations to accelerate the computation by pruning the search space with size constraint and refining the candidates in a lazy manner. We use several real datasets from various application domains, one of which contains over 300 million vertices and 1.3 billion edges, to demonstrate the high efficiency and scalability of our proposed solution. It is reported that 50% improvement on recall can be achieved after applying our method in Alibaba Group to identify the fraudulent transactions in their e-commerce networks. This further demonstrates the usefulness of our techniques in practice.

Keywords Biclique search · Bipar title graph · Graph algorithms

✉ Lu Qin
lu.qin@uts.edu.au

Bingqing Lyu
bingqing.lbq@alibaba-inc.com

Xuemin Lin
lxue@cse.unsw.edu.au

Ying Zhang
ying.zhang@uts.edu.au

Zhengping Qian
zhengping.qzp@alibaba-inc.com

Jingren Zhou
jingren.zhou@alibaba-inc.com

[1] Alibaba Group, Hangzhou, China

[2] The University of Technology Sydney, Ultimo, Australia

[3] The University of New South Wales, Sydney, Australia

1 Introduction

A bipartite graph is denoted by $G = (U, V, E)$ where $U(G)$ and $V(G)$ denote the two disjoint vertex sets and $E(G) \in U \times V$ denotes the edge set. Bipartite graph is a popular data structure, which has been widely used for modelling the relationship between two sets of entities in many real world applications. For example, in E-Commerce, a bipartite graph can be used to model the purchasing relationship between customers and products; In web applications, a bipartite graph can be used to model the visiting relationship between users and websites; In bioinformatics, a bipartite graph can be used to model the acting relationship between genes and roles in biological processes.

A subgraph C is a *biclique* if it is a complete bipartite subgraph of G that for every pair $u \in U(C)$ and $v \in V(C)$, we

have $(u, v) \in E(C)$. Like a clique in general graph, biclique is a fundamental structure in a bipartite graph, and has been widely used to capture cohesive bipartite subgraphs in a wide spectrum of bipartite graph applications. Below are several representative examples.

(1) Anomaly detection [4,7] In E-commerce such as Ebay and Alibaba, the behavior of a large group of customers purchasing a set of products together is considered as an anomaly because there is a high probability that the group of people is making fraudulent transactions to increase the rankings of their businesses selling the corresponding products. This can be modeled as bicliques in a bipartite graph. Similarly, in web services, bicliques can be used to detect a group of web spammers who click a set of webpages together to promote their rankings.

(2) Gene expression analysis [16,18,25,45,59] In gene expression data analysis, different genes will respond in different conditions. The group of genes that have a number of common responses over multiple conditions is considered as a significant gene group.

(3) Social recommendation [23] In social analysis, there may exist a group of users who have the same set of interests, such as swimming, hiking, and fishing. Such groups and interests can be naturally captured by a biclique, which is helpful in social recommendation and advertising.

In practice, we cannot directly enumerate the bicliques of the bipartite graphs as the number of bicliques is prohibitively large in the above applications. In this paper, we investigate the problem of **maximum biclique search**, i.e., finding the biclique with the largest number of edges, for the following two reasons:

(1) Given the biclique model, it is a very natural problem to find the maximum biclique, which is not only theoretically interesting but also useful in many real-life scenarios. For instance, the maximum biclique may represent the largest suspicious click farm in the e-commerce networks, the most significant gene group in a gene-condition bipartite graph, and the user group with the largest potential market value in the social network.

(2) In some scenarios, one may need to enumerate a set of bicliques. For instance, the fraud transactions cannot be fully covered by the maximum biclique in the e-commerce network. To reduce the number of output bicliques, we may consider the *maximal biclique* where none of its superset is also a biclique. Unfortunately, as shown in our initial empirical study, the number of maximal biclique is still large (e.g., over 10^9 maximal bicliques have been output after 24 h running of maximal biclique enumeration algorithm on a e-commerce bipartite graph obtained from Alibaba). Thus, we have to consider the diversified top-k bicliques. Inspired by the well-studied diversified top-k clique search problem (e.g., [57]), we can follow the same procedure by repeatedly removing the current maximum biclique from the bipartite graph k times. Clearly, the efficient computation of maximum biclique is the key of this problem.

Challenges and motivations Despite its wide range of applications, finding the maximum biclique is an NP-hard problem [38]. In the literature, there are many solutions to solve another related NP-hard problem: the maximum clique search problem in a general graph [17,19,20,26,31,46–49]. The main idea is to use graph coloring and core decomposition to obtain an upper bound for the maximum clique size and use this upper bound to prune vertices that cannot be contained in the maximum clique.

A natural question raised is: can we use the above graph coloring and core decomposition techniques to search the maximum biclique in a bipartite graph? Unfortunately, the answer is negative. First, in a bipartite graph, only two colors are needed to color the whole bipartite graph. Obviously, we cannot obtain an upper bound for the maximum biclique size using graph coloring. Second, in a large biclique, it is possible for a vertex to have a very small degree/core number. For example, suppose the maximum biclique C is a star where $|U(C)| = 1$ and $|V(C)|$ is large, we only require the degree/core number for each vertex in $V(C)$ to be ≥ 1. Consequently, even a vertex has a small degree/core number, it still cannot be pruned. Therefore, the core decomposition technique also fails in maximum biclique search.

The main reason for the challenges in maximum biclique search is that the size of a biclique C depends on two factors: $|U(C)|$ and $|V(C)|$; so, it is difficult to find a one-dimensional indicator, such as color number, degree, or core number, to prune vertices that cannot participate in the maximum biclique. Due to this challenge, existing solutions [38,59] can only handle small bipartite graphs and will face serious efficiency issues when the bipartite graph scales up in size. Motivated by this, in this paper, we tackle the above challenges and aim to solve the maximum biclique search problem on bipartite graphs at billion scale.

Furthermore, based on the maximum biclique search, we can find the diversified top-k bicliques which is desired in some applications such as fraudulent transaction detection. Instead of computing the top-k bicliques based on the maximal biclique enumeration algorithm which may output exponential number of bicliques and is not practical on large-scale bipartite graphs, we adopt a simple but effective method by removing the maximum biclique from the bipartite graph k times to obtain the diversified top-k results. However, in this way, we still need to compute the maximum biclique k times independently, which is costly. One may wonder if we can share the computation costs among the diversified top-k bicliques. It is quite challenging because there is no overlap among the k diversified results.

Our solution Based on the above discussion, existing coloring and core decomposition-based approaches cannot yield

effective pruning in maximum biclique search. Our paper aims for a new way to solve the problem. Our main idea is as follows: instead of finding the upper bounds for pruning, we try to guess a lower bound of $|U(C^*)|$ as well as a lower bound of $|V(C^*)|$ for the maximum biclique C^*. If the guess is correct and tight, we can search on a much smaller bipartite graph by eliminating a large number of vertices based on the two lower bounds. However, we cannot guarantee that our guess is always correct. Therefore, instead of guessing only once, we guess multiple times which results in a list of lower-bound pairs $(\tau_U^0, \tau_V^0), (\tau_U^1, \tau_V^1), \ldots$. To gain high pruning power, the list of pairs should satisfy four conditions: (1) $\tau_U^0 \times \tau_V^0$ should be as large as possible but not larger than the number of edges in the optimal biclique C^*; (2) The pairs are derived in a progressive manner so that $\tau_U^i \times \tau_V^i \geq \tau_U^{i-1} \times \tau_V^{i-1}$ for any $i > 0$; (3) There exists at least one pair τ_U^k and τ_V^k that are the true lower bounds of $|U(C^*)|$ and $|V(C^*)|$; and (4) The number of pairs should be well-bounded.

To make this idea practically applicable, two issues need to be addressed: (1) How to guess the list of lower-bound pairs so that they satisfy the above four conditions; and (2) Given a lower-bound pair, how to eliminate as many vertices as possible while preserving the corresponding maximum biclique to optimize the computational cost.

Following the idea of the maximum biclique search problem, in the diversified top-k biclique search, we try to share the computation cost among the k results by taking advantage of the derived subspaces with lower-bound pairs. Our main idea is as follows: instead of guessing tight lower bounds only for the maximum biclique, we try to preserve more results within one list of lower-bound pairs by slightly relaxing the constraints in each lower bound pair. By doing this, we can share the computation cost among the preserved results, without computing lower-bound lists and eliminating vertices w.r.t. each lower-bound pair independently for every single result.

Contributions In this paper, we answer the above questions and make the following contributions:

- *The first work to practically study maximum biclique search on big real datasets* Although the maximum biclique search problem is NP-hard, we aim to design practical solutions to solve the problem in real-world large bipartite graphs with billions of edges. To the best of our knowledge, this is the first work to solve this important problem on real datasets at billion scale.
- *A novel progressive-bounding framework* We propose a progressive bounding framework to obtain the lower-bound pairs (τ_U^i, τ_V^i). We analyze the framework by projecting the problem into a two-dimensional space, and we show that the set of lower-bound pairs forms a skyline in the two-dimensional space, and only logarithmic lower-bound pairs are enough to guarantee the correctness.
- *Maximum-biclique preserved graph reduction* Given a certain pair of lower bounds, we study how to eliminate vertices while preserving the maximum biclique. We investigate the vertex properties and derive pruning rules by exploring the one-hop and two-hop neighbors for each vertex. Based on the pruning rules, we can significantly reduce the size of the bipartite graph.
- *Diversified top-k biclique search with computation sharing* We formalize the diversified top-k biclique search as a problem to maximize the total number of edges covered by the top-k bicliques, which takes both size and diversity into consideration. Instead of computing the k results independently, we propose an efficient algorithm by considering the computation sharing among them. Based on the progressive bounding framework, we generate the subspaces by slightly relaxing the lower-bound constraints to preserve more results within one subspace set, such that we can share the computation among the preserved results. Two optimizations are proposed to further accelerate the computation by pruning search spaces and lazy refining candidates.
- *Extensive performance studies on billion-scale bipartite graphs* We conduct extensive performance studies using 18 real datasets from different application domains. The experimental results demonstrate the efficiency and scalability of our proposed approaches. Remarkably, in a user-product bipartite graph from Alibaba with over 300 million vertices and over 1.3 billion edges, our approach can find the maximum biclique within 15 min. It is also reported that 50% improvement on recall can be achieved after applying our proposed method in Alibaba Group to identify the fraudulent transactions.

Outline The remainder of this paper is organized as follows. Section 2 provides the preliminaries that formally defines the maximum biclique search problem and shows its hardness. Section 3 introduces the baseline solution based on the branch-and-bound framework. In Sect. 4, we analyze the reason for the inefficiency of the baseline solution, and propose the progressive bounding framework. Section 5 presents the maximum-biclique preserved graph reduction techniques and its optimizations. In Sect. 6, we study the problem of diversified top-k biclique search and propose an efficient algorithm by sharing the computation cost among the k results. In Sect. 7, we evaluate our proposed algorithms using extensive experiments. We review the related work in Sect. 8 and conclude the paper in Sect. 9.

This paper is extended from our previous work [28] to give a more comprehensive study. First, we add the introduction and motivation of the diversified top-k biclique search problem. Then, we add algorithms TopKBasic and TopK to find the

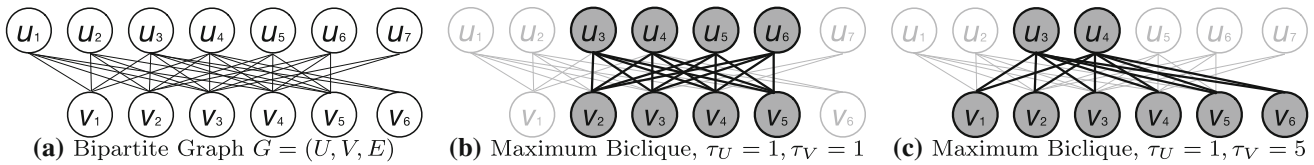


Fig. 1 An example of a bipartite graph and its maximum biclique

diversified top-k bicliques, with two optimizations to further accelerate the computation. Finally, we add more experiments on maximum and diversified top-k biclique search to show the efficiency of the proposed algorithms.

2 Preliminaries

We consider an unweighted and undirected bipartite graph, $G = (U, V, E)$ where $U(G)$ and $V(G)$ denote the two disjoint vertex sets and $E(G) \in U \times V$ denotes the edge set in G. For each vertex $u \in U(G)$, we use $N(u, G)$ to denote the set of neighbors of u in G, i.e., $N(u, G) = \{v|(u, v) \in E(G)\}$. The degree of a vertex $u \in U(G)$, denoted as $d(u, G)$, is the number of neighbors of u in G, i.e., $d(u, G) = |N(u, G)|$. We use $d^U_{\max}(G)$ to denote the maximum degree for all vertices in $U(G)$, i.e., $d^U_{\max}(G) = \max_{u \in U(G)} d(u, G)$. We have symmetrical definition for each vertex $v \in V(G)$. The size of a bipartite graph G, denoted as $|G|$, is defined as the number of edges in G, i.e., $|G| = |E(G)|$.

Definition 1 *(Biclique)* Given a bipartite graph $G = (U, V, E)$, a biclique C is a complete bipartite subgraph of G, i.e., for each pair of $u \in U(C)$ and $v \in V(C)$, we have $(u, v) \in E(C)$.

In this paper, given a bipartite graph G, we aim to find a biclique C^* in G with the maximum size. Considering that many real applications (e.g., fraud transaction detection) require that the number of vertices in each part of the biclique C^* is not below a certain threshold, we add size constraints τ_U and τ_V on $|U(C^*)|$ and $|V(C^*)|$ s.t. $|U(C^*)| \geq \tau_U$ and $|V(C^*)| \geq \tau_V$. Such a size constraint can also provide the users with more flexibility to control the size of each side of the biclique or avoid generating a too skewed biclique (e.g., a biclique with a single vertex of the highest degree at one side and all its neighbors at the other side). As a special case, when $\tau_U = 1$ and $\tau_V = 1$, the problem will find the maximum biclique without any constraint. The maximum biclique problem studied in this paper is defined as follows:

Problem statement Given a bipartite graph $G = (U, V, E)$, and a pair of positive integers τ_U and τ_V, the problem of maximum biclique search aims to find a biclique C^* in G, s.t. $|U(C^*)| \geq \tau_U$ and $|V(C^*)| \geq \tau_V$, and $|C^*|$ is maximized. We use $C^*_{\tau_U, \tau_V}(G)$ to denote such a biclique.

Example 1 Figure 1a shows a bipartite graph G with $U(G) = \{u_1, u_2, ..., u_7\}$, $V(G) = \{v_1, v_2, ..., v_6\}$. Given thresholds $\tau_U = 1$ and $\tau_V = 1$, the maximum biclique $C^*_{1,1}(G) = C_1$ is shown in Fig. 1b, where $U(C_1) = \{u_3, u_4, u_5, u_6\}$ and $V(C_1) = \{v_2, v_3, v_4, v_5\}$. Given thresholds $\tau_U = 1$ and $\tau_V = 5$, the maximum biclique $C^*_{1,5}(G) = C_2$ is shown in Fig. 1c, where $U(C_2) = \{u_3, u_4\}$ and $V(C_2) = \{v_1, v_2, ..., v_6\}$.

NP-hardness and inapproximability As shown in [38], the maximum biclique problem is NP-hard, and as proved in [5] and [30], it is difficult to find a polynomial time algorithm to solve the maximum biclique problem with a promising approximation ratio. Due to the inapproximability, in this paper, we aim to find the exact maximum biclique and will propose several techniques to make our algorithm practical in handling large real-world bipartite graphs.

3 The baseline solution

In the literature, the state-of-the-art algorithm proposed in [59] resorts to the branch-and-bound framework, aiming to list all maximal bicliques by pruning non-maximal candidates from the search space. To obtain a reasonable baseline, in this section, we extend the algorithm proposed in [59], and design an algorithm to compute the maximum biclique by adding some pruning rules in the branch-and-bound process.

The branch-and-bound algorithm We briefly introduce the branch-and-bound algorithm. The algorithm maintains a partial biclique $(U, V, U \times V)$ and recursively adds vertices into V. When V is fixed, U can be simply computed as the set of common neighbors of all vertices in V, i.e.,

$$U = \{u|(u, v) \in E(G)\ \forall v \in V\} \tag{1}$$

Therefore, we only need to consider V to determine the biclique. Based on this idea, the key to reducing the cost is to prune the useless vertices to be added into V. According to Eq. 1, when V is expanded, U will be contracted.

The pseudocode of the algorithm is shown in Algorithm 1. The input of the algorithm includes the bipartite graph G, the thresholds τ_U and τ_V, and an initial biclique C. Here, C is

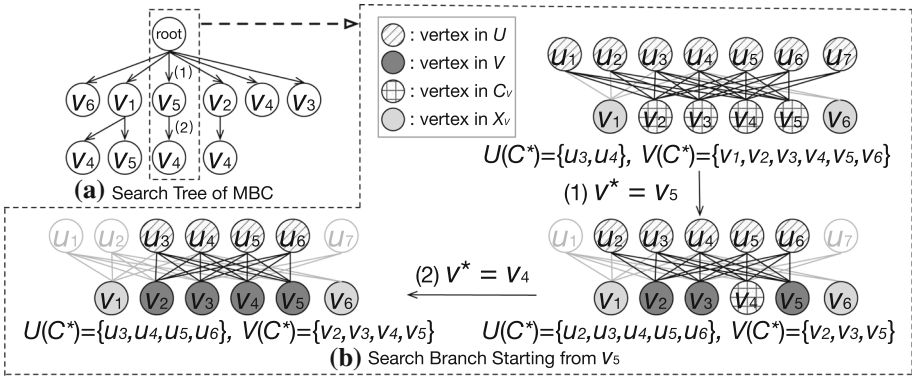


Fig. 2 An example of MBC searching

Algorithm 1: MBC(G, τ_U, τ_V, C)

Input : Bipartite graph G, τ_U and τ_V, initial biclique C
Output : The maximum biclique C^*

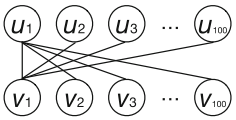
1 $C^* \leftarrow C$;
2 BranchBound($U(G), \emptyset, V(G), \emptyset$);
3 **return** C^*;
4 **Procedure** BranchBound(U, V, C_V, X_V)
5 **if** $|V| \geq \tau_V$ **and** $|U| \times |V| > |C^*|$ **then**
6 $C^* \leftarrow (U, V, U \times V)$;
7 **while** $C_V \neq \emptyset$ **do**
8 $v^* \leftarrow C_V.pop()$;
9 $U' \leftarrow \{u \in U | (u, v^*) \in E(G)\}$;
10 $V' \leftarrow V \cup \{v^*\} \cup \{v \in C_V | U' \subseteq N(v, G)\}$;
11 $C'_V \leftarrow \{v \in C_V \setminus V' \mid |N(v, G) \cap U'| \geq \tau_U\}$;
12 $X'_V \leftarrow \{v \in X_V \mid |N(v, G) \cap U'| \geq \tau_U\}$;
13 **if** $|U'| \geq \tau_U$ **and** $|V'| + |C'_V| \geq \tau_V$ **and** $|U'| \times (|V'| + |C'_V|) > |C^*|$ **and** $\nexists\, v \in X_V$ s.t. $U' \subseteq N(v, G)$ **then**
14 BranchBound(U', V', C'_V, X'_V);
15 $X_V \leftarrow X_V \cup \{v^*\}$;

used when a biclique is obtained before invoking the algorithm, or can be set as $\emptyset$ otherwise. The algorithm initializes C^* as C (line 1), invokes the BranchBound procedure to update C^* (line 2), and returns C^* as the answer (line 3).

The recursive procedure BranchBound has four parameters U, V, C_V, and X_V, initialized as $U(G)$, $\emptyset$, $V(G)$ and $\emptyset$, respectively. Here, $(U, V, U \times V)$ defines a partial biclique. C_V is the set of candidate vertices that can be possibly added to V, and X_V is the set of vertices that has been used and should be excluded from V. The procedure BranchBound updates C^* using $(U, V, U \times V)$ if it is larger than the current C^* and satisfies the threshold constraints (line 5–6). Then, it iteratively adds vertex v^* from C_V to expand V (line 7–8).

Then, U' is updated by selecting the vertices from U that are neighbors of v^*; V' includes vertices in V, v^*, and vertices in C_V that are neighbors of all vertices in U'; C'_V includes the vertices in C_V by excluding the vertices in V' as well as the vertices with number of neighbors in U' no larger than τ_U; X'_V includes all vertices in X_V by excluding the vertices with number of neighbors in U' no larger than τ_U (line 9–12).

Fig. 3 Drawbacks of MBC



The new search branch by including v^* will be created after considering the following pruning conditions (line 13–14):

(1) τ_U pruning The size of U' should be $\geq \tau_U$ since U will only be contracted in the branch.

(2) τ_V pruning The size of $V' \cup C'_V$ should be $\geq \tau_V$.

(3) Size pruning The value of $|U'| \times (|V'| + |C'_V|)$ should be $\geq |C^*|$. Without it, exploiting the current branch will not result in a larger biclique.

(4) Non-maximality pruning The non-maximality pruning is based on the fact that a maximum biclique should be a maximal biclique. If there is a vertex v in the exclusion set X_V that are neighbors of all vertices in U' (i.e., $U' \subseteq N(v, G)$), the resulting biclique cannot be maximal and thus the branch can be pruned.
After searching bicliques with v^*, we add v^* into X_v (line 15).

Example 2 Given the bipartite graph G in Fig. 1a and thresholds $\tau_U = 1$ and $\tau_V = 1$, we show the search tree of MBC in Fig. 2a. The vertices in V are processed in non-descending order of degree [59], and each tree node represents v^* selected in the branch. We illustrate the details in search branch from v_5 in Fig. 2b. At first, we have $X_V = \{v_6, v_1\}$, $C_V = \{v_5, v_2, v_4, v_3\}$, $U(C^*) = \{u_3, u_4\}$, and $V(C^*) = \{v_1, v_2, v_3, v_4, v_5, v_6\}$. In step (1), we select $v^* = v_5$ and refine $U' = \{u_2, u_3, u_4, u_5, u_6\}$. V' is the vertices in C_V that connect to all vertices in U', i.e., $V' = \{v_2, v_3, v_5\}$. Then, we refine $C'_V = \{v_4\}$ and $X'_V = \{v_1, v_6\}$. By now, we update $U(C^*) = U'$, $V(C^*) = V'$ and $|C^*| = 15$. In step (2), we further select $v^* = v_4$, refine corresponding sets in a similar way as shown in Fig. 2, and update $|C^*| = 16$.

4 A progressive bounding method

In this section, we first analyze the reason for the large search space of the baseline solution, and then introduce our approach using search space partitioning based on a progressive bounding framework to significantly reduce the computational cost.

4.1 Problem analysis

Why costly? Although four pruning conditions are used to reduce the search space for maximum biclique search in Algorithm 1, it will still result in a huge search space in real large bipartite graphs due to the following two drawbacks:

- *Drawback 1: loose pruning bounds* Most pruning conditions in Algorithm 1 rely on τ_U and τ_V. However, τ_U and τ_V are user given parameters which can be small. In this way, the pruning power by τ_U and τ_V can be rather limited. For size pruning, the constraint of $|U'| \times (|V'| + |C'_V|) > |C^*|$ can be very loose because C'_V is filtered using τ_U and thus $|C'_V|$ can be large when τ_U is small.
- *Drawback 2: large candidate size* The size of a biclique C, calculated as $|U(C)| \times |V(C)|$, depends on two factors: $|U(C)|$ and $|V(C)|$. It is possible that the optimal solution C^* is unbalanced, i.e., either with a large $|U(C^*)|$ and a small $|V(C^*)|$ or with a small $|U(C^*)|$ and a large $|V(C^*)|$. Therefore, during the branch-and-bound process, even if the degrees of all candidates in C_V are small (where $|U|$ is small), we cannot stop branching when $V \cup C_V$ is large, because we may still generate a large biclique in this situation. Similarly, we cannot remove a vertex from U when its degree is small. This can result in a huge search space on a large bipartite graph.

Example 3 Figure 3 shows a bipartite graph G with $U = \{u_1, u_2, ..., u_{100}\}$ and $V = \{v_1, v_2, ..., v_{100}\}$. Specifically, u_1 connects to all vertices in V and v_1 connects to all vertices in U. Given $\tau_U = 1$ and $\tau_V = 1$, the size of maximum biclique C^* is 100. By adopting MBC, we firstly select v_1 into V'. As v_1 connects to all vertices in U, $U' = \{u_1, u_2, ..., u_{100}\}$. Furthermore, as u_1 connects to all vertices in V, $C'_V = \{v_2, v_3, ..., v_{100}\}$. However, we cannot prune any vertices with $\tau_U = 1$ and $\tau_V = 1$, and neither can we prune search branches with size constraint since $|U'| \times (|V'| + |C'_V|)$ is larger than $|C^*|$. Moreover, we can not prune candidate vertices in C'_V, though the degrees of vertices are 1s, which leads to large candidate size and a huge search space.

Our idea Based on the above analysis and to significantly improve the algorithm, we consider two aspects:

- To resolve drawback 1, we need to improve the pruning bounds to achieve the stop conditions in early stages of the branch-and-bound process;
- To resolve drawback 2, we need to remove as many vertices as possible from the graph to reduce the number of candidates that may participate in the optimal solution.

Our idea is as follows: instead of using the thresholds τ_U and τ_V for pruning, we enforce two new thresholds τ^*_U and τ^*_V for $U(C^*)$ and $V(C^*)$, respectively, with $\tau^*_U \geq \tau_U$ and $\tau^*_V \geq \tau_V$. To tighten the bounds, we try to make $\tau^*_U \times \tau^*_V$ as large as possible but ensure that $\tau^*_U \times \tau^*_V$ is no larger than the size of the optimal solution. With τ^*_U and τ^*_V, we are able to obtain a smaller bipartite graph G^* by removing as many vertices as possible that will not participate in the maximum biclique. On the smaller graph G^* with tighter bounds τ^*_U and τ^*_V, the algorithm will be much more efficient. Suppose C^* is the optimal solution, if we can guarantee that $\tau^*_U \leq |U(C^*)|$ and $\tau^*_V \leq |V(C^*)|$, the algorithm on graph G^* with thresholds τ^*_U and τ^*_V will output the optimal solution.

However, to make our idea practically applicable, the following two issues need to be addressed:

- First, we do not know the size of the maximum biclique C^* before the search.
- Second, it is difficult to find a single pair τ^*_U and τ^*_V to guarantee that $\tau^*_U \leq |U(C^*)|$ and $\tau^*_V \leq |V(C^*)|$.

In the following, we will introduce a progressive bounding framework to resolve the two issues.

4.2 The progressive bounding framework

We propose a progressive bounding framework to address the two issues raised as follows:

- To address the first issue, instead of using the size of the optimal solution $|C^*|$, we use a lower bound $lb(C^*)$ of $|C^*|$, i.e., $lb(C^*) \leq |C^*|$. The lower bound can be quickly initialized and will be updated progressively to make the thresholds τ^*_U and τ^*_V tighter.
- To address the second issue, instead of using a single pair τ^*_U and τ^*_V, we use multiple pairs (τ^1_U, τ^1_V), (τ^2_U, τ^2_V), ..., (τ^k_U, τ^k_V). We will guarantee that for any possible biclique C with $U(C) \times V(C) \geq lb(C^*)$, there exists a pair (τ^i_U, τ^i_V) for $1 \leq i \leq k$ s.t. $\tau^i_U \leq |U(C)|$ and $\tau^i_V \leq |V(C)|$. Then, for each (τ^i_U, τ^i_V) for $1 \leq i \leq k$, we compute a biclique C^*_i with maximum size s.t. $|U(C^*_i)| \geq \tau^i_U$ and $|V(C^*_i)| \geq \tau^i_V$. Among the computed bicliques, the biclique with the maximum size is the answer for the original problem.

Algorithm 2: MBC*(G, τ_U, τ_V)

Input : Bipartite graph G, thresholds τ_U and τ_V
Output : The maximum biclique C^*

1 $C_0^* \leftarrow$ InitMBC(G, τ_U, τ_V);
2 $\tau_V^0 \leftarrow d_{\max}^U(G)$;
3 $k \leftarrow 0$;
4 **while** $\tau_V^k > \tau_V$ **do**
5 $\quad \tau_U^{k+1} \leftarrow \max(\left\lfloor \frac{|C_k^*|}{\tau_V^k} \right\rfloor, \tau_U)$;
6 $\quad \tau_V^{k+1} \leftarrow \max(\left\lfloor \frac{\tau_V^k}{2} \right\rfloor, \tau_V)$;
7 $\quad G_{k+1} \leftarrow$ Reduce$(G, \tau_U^{k+1}, \tau_V^{k+1})$;
8 $\quad C_{k+1}^* \leftarrow$ MBC$(G_{k+1}, \tau_U^{k+1}, \tau_V^{k+1}, C_k^*)$;
9 $\quad k \leftarrow k+1$;
10 **return** C_k^*;

The algorithm framework The progressive bounding framework is shown in Algorithm 2. For any valid biclique C with $|U(C)| \geq \tau_U$ and $|V(C)| \geq \tau_V$, $|C|$ is a lower bound of the optimal solution C^*. Based on this, we first use InitMBC to obtain an initial biclique, denoted as C_0^*, s.t. $|C_0^*| \leq |C^*|$ (line 1). Then, we set τ_V^0 to be an upper bound of $|V(C)|$ for any possible biclique C. Here, a natural upper bound is the maximum degree for any nodes in $U(G)$, i.e., $d_{\max}^U(G)$ (line 2). k is used to denote the number of iterations and initialized as 0 (line 3). The progressive bounding framework will finish in logarithmic iterations. Each iteration will generate a pair τ_U^{k+1} and τ_V^{k+1} based on the values of τ_V^k and the the lower bound of the optimal solution $|C_k^*|$. When τ_V^{k+1} (τ_U^{k+1} resp.) is smaller than τ_V (τ_U resp.), it will be set to be τ_V (τ_U resp.) (line 5–6). We will analyze the rationale later. With τ_U^{k+1} and τ_V^{k+1}, we aim to obtain a graph G_{k+1} that is much smaller than G using procedure Reduce$(G, \tau_U^{k+1}, \tau_V^{k+1})$, and the maximum biclique w.r.t. thresholds τ_U^{k+1} and τ_V^{k+1} is preserved in G_{k+1} (line 7). After this, we find the maximum biclique w.r.t. τ_U^{k+1} and τ_V^{k+1} on G_{k+1} with C_k^* as an initiation in MBC (line 8).

The rationale Next, we address the rationale of the progressive bounding framework. Note that the size of a biclique C is determined by $|U(C)|$ and $|V(C)|$. Therefore, to analyze the problem, we define a two-dimensional space as follows:

Definition 2 *(Search Space $\mathcal{S}(G)$)* Given a bipartite graph G, a two-dimensional space $\mathcal{S}(G)$ has two axes $|U|$ and $|V|$. Given any biclique C in G, we can represent it as a two-dimensional point $(|U(C)|, |V(C)|)$ in the space $\mathcal{S}(G)$.

Given the search space $\mathcal{S}(G)$, the i-th search in line 7-8 of Algorithm 2 can be considered as to cover a certain subspace $([\tau_U^i, +\infty), [\tau_V^i, +\infty))$ in $\mathcal{S}(G)$. To show the search preserves the optimal solution, we define the optimal curve in $\mathcal{S}(G)$:

Definition 3 *(Optimal Curve)* Given a bipartite graph G and parameters τ_U and τ_V, suppose C^* is the maximum biclique w.r.t. τ_U and τ_V, we call the curve $|U| \times |V| = |C^*|$ the optimal curve in the two-dimensional space $\mathcal{S}(G)$.

Note that the optimal curve is unknown before the search. However, it can be used to analyze the correctness of the progressive bounding framework as followers.

Theorem 1 (Algorithm Correctness) *Given a bipartite graph G and parameters τ_U and τ_V, for any point (s_U, s_V) on the optimal curve with $s_U \in [\tau_U, d_{\max}^V(G)]$ and $s_V \in [\tau_V, d_{\max}^U(G)]$, there exists a certain (τ_U^i, τ_V^i) generated by Algorithm 2 s.t. $(s_U, s_V) \in ([\tau_U^i, +\infty), [\tau_V^i, +\infty))$.*

Proof Sketch: In Algorithm 2, τ_V^0 is set to be $d_{\max}^U(G)$, and when k increases, τ_V^k will be iteratively divided by 2 until it is smaller than τ_V. Therefore, we can always find a certain $i > 0$ s.t.

$$\tau_V^i \leq s_V \leq \tau_V^{i-1}$$

Based on Algorithm 2, we have $\tau_U^i = \max(\left\lfloor \frac{|C_{i-1}^*|}{\tau_V^{i-1}} \right\rfloor, \tau_U)$. We consider two cases:

- *Case 1:* $\tau_U^i = \tau_U$. In this case, we have:

  $$s_U \geq \tau_U = \tau_U^i$$

  Therefore, $(s_U, s_V) \in ([\tau_U^i, +\infty], [\tau_V^i, +\infty])$ holds.
- *Case 2:* $\tau_U^i = \left\lfloor \frac{|C_{i-1}^*|}{\tau_V^{i-1}} \right\rfloor$. Note that $|C_{i-1}^*|$ is a lower bound of the optimal value $|C^*|$ i.e.,

  $$|C_{i-1}^*| \leq |C^*|$$

  Since (s_U, s_V) is a point on the optimal curve, we have

  $$s_U \times s_V = |C^*|$$

  Consequently, we can derive the following inequalities:

  $$\tau_U^i = \left\lfloor \frac{|C_{i-1}^*|}{\tau_V^{i-1}} \right\rfloor \leq \left\lfloor \frac{|C^*|}{\tau_V^{i-1}} \right\rfloor \leq \left\lfloor \frac{|C^*|}{s_V} \right\rfloor = \lfloor s_U \rfloor \leq s_U$$

  Therefore, $(s_U, s_V) \in ([\tau_U^i, +\infty], [\tau_V^i, +\infty])$ holds.

According to the analysis above, Theorem 1 holds. □

Theorem 1 shows that all the points in the optimal curve within the range $([\tau_U, d_{\max}^V(G)], [\tau_V, d_{\max}^U(G)])$ are covered by the search spaces in Algorithm 2. Note that for any

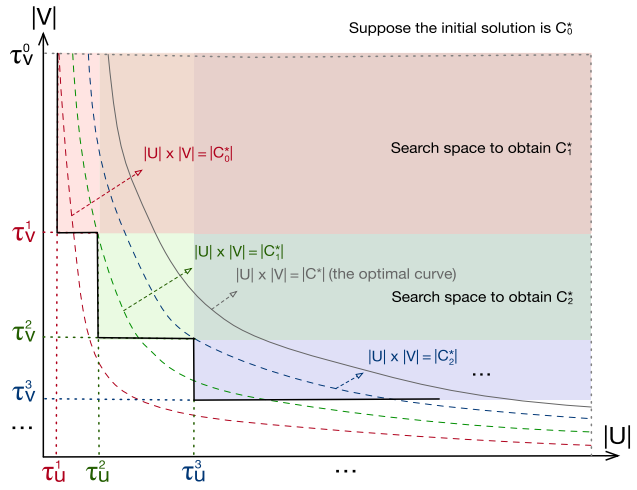


Fig. 4 Illustration of algorithm rationale

biclique C in G, we can guarantee that $|U(C)| \leq d^V_{\max}(G)$ and $|V(C)| \leq d^U_{\max}(G)$. Therefore, Algorithm 2 obtains the optimal solution.

The rationale of the progressive bound framework is shown in Fig. 4. Here, we draw the two-dimensional space $\mathcal{S}(G)$, and show the search spaces of the first three iterations of Algorithm 2 on $\mathcal{S}(G)$. We generate three search spaces using (τ^1_U, τ^1_V), (τ^2_U, τ^2_V), and (τ^3_U, τ^3_V), which obtains the bicliques C^*_1, C^*_2, and C^*_3, respectively. We use red, green, and blue colors to differentiate the three spaces respectively. As shown in Fig. 4, when i increases, the curve $|U| \times |V| = |C^*_i|$ progressively approaches the optimal curve $|U| \times |V| = |C^*|$, and the optimal curve $|U| \times |V| = |C^*|$ in $\mathcal{S}(G)$ for $|V| \geq \tau^3_V$ is totally covered by the three search spaces. This illustrates the correctness of the progressive bounding framework.

Example 4 Given the bipartite graph G in Fig. 1a and thresholds $\tau_U = 1$ and $\tau_V = 1$, we adopt Algorithm 2 to find the maximum biclique. Suppose we initiate biclique C^*_0 as shown in Fig. 1c that we have $|C^*_0| = 12$ and $\tau^0_V = 6$. Then, we search the optimal solution progressively:

(1) $\tau^1_U = 2$, $\tau^1_V = 3$. We adopt Reduce to filter vertices in G, e.g., we filter u_7 as $d(u_7, G) = 2$ and it cannot be involved in a biclique with $\tau^1_V = 3$. We will explain Reduce in detail later. We search for C^*_1 on G_1, and get $U(C^*_1) = \{u_3, u_4, u_5, u_6\}$, $V(C^*_1) = \{v_2, v_3, v_4, v_5\}$. Thus $|C^*_1| = 16$.
(2) $\tau^2_U = 5, \tau^2_V = 1$. Since we cannot find any larger biclique on reduced graph G_2, $|C^*_2| = 16$. As shown above, we progressively use multiple strict τ^k_U and τ^k_V threshold pairs to approach the optimal solution.

The effectiveness of the progressive bounding framework is further verified in our experiments. For example, Table 2 shows that the graph compression ratio in the bounding iterations varies from 0% (omitted in the table) to 2.05%. This reduces significantly the search space and computation cost in the maximum biclique search procedure.

To realize the algorithm framework MBC* in Algorithm 2, we still need to solve the following two components:

– *The initial biclique computation algorithm* InitMBC. We use a greedy strategy to obtain the initial biclique. Specifically, we initialize an empty biclique and iteratively add the vertex that can maximize the size of the current biclique until no vertex can be added. The biclique with the maximum size among the process is returned.
– The graph reduction algorithm Reduce. We will discuss the details of Reduce in the next section.

5 MBC-preserved graph reduction

As shown in Algorithm 2, one of the most important procedures is to reduce the size of the bipartite graph given certain τ^i_U and τ^i_V while preserving the maximum biclique. In this section, we show how to reduce the bipartite graph size by exploring some properties of the one-hop and two-hop neighbors for a certain vertex. We first introduce the MBC-preserved graph below.

Definition 4 *(MBC-Preserved Graph)* Given a bipartite graph G, and thresholds τ^i_U and τ^i_V, a bipartite graph G' is called a MBC-preserved graph w.r.t. τ^i_U and τ^i_V, if $U(G') \subseteq U(G)$, $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$ and $|C^*_{\tau^i_U,\tau^i_V}(G')| = |C^*_{\tau^i_U,\tau^i_V}(G)|$. In other words, the maximum biclique for G is preserved in G'. We use $G' \sqsubseteq_{\tau^i_U,\tau^i_V} G$ to denote that G' is an MBC-preserved graph of G.

We can easily derive the following lemma:

Lemma 1 (Transitive Property) *If* $G_1 \sqsubseteq_{\tau^i_U,\tau^i_V} G_2$ *and* $G_2 \sqsubseteq_{\tau^i_U,\tau^i_V} G_3$, *we have* $G_1 \sqsubseteq_{\tau^i_U,\tau^i_V} G_3$.

5.1 One-hop graph reduction

To reduce the size of the bipartite graph, we first consider a simple case by exploring the one-hop neighbors for each vertex. Specifically, we use the number of neighbors to reduce the bipartite graph. Besides, we eliminate a vertex u by removing u and all its adjacent edges from G, denoted as $G \ominus u$. We derive the following lemma:

Lemma 2 *Given a bipartite graph* G, *thresholds* τ^i_U *and* τ^i_V, *we have:*

(1) $\forall u \in U(G)$: $d(u, G) < \tau^i_V \implies G \ominus u \sqsubseteq_{\tau^i_U,\tau^i_V} G$;

Algorithm 3: Reduce1Hop(G, τ_U^i, τ_V^i)

Input : Bipartite graph G, thresholds τ_U^i and τ_V^i
Output : A graph G_i s.t. $G_i \sqsubseteq_{\tau_U^i, \tau_V^i} G$

```
1 G_i ← G; finish ← false;
2 while finish = false do
3     finish ← true;
4     if exists u ∈ U(G_i) s.t. d(u, G_i) < τ_V^i then
5         G_i ← G_i ⊖ u; finish ← false;
6     if exists v ∈ V(G_i) s.t. d(v, G_i) < τ_U^i then
7         G_i ← G_i ⊖ v; finish ← false;
8 return G_i;
```

(2) $\forall v \in V(G)$: $d(v, G) < \tau_U^i \implies G \ominus v \sqsubseteq_{\tau_U^i, \tau_V^i} G$.

Proof Sketch: We only prove (1), and (2) can be proved similarly. Given a certain vertex $u \in U(G)$ with $d(u, G) < \tau_V^i$, we need to prove that for any biclique C in G with $|U(C)| \geq \tau_U^i$ and $|V(C)| \geq \tau_V^i$, C is also a biclique in $G \ominus u$. That is, we only need to prove $u \notin U(C)$. Next, we prove $u \notin U(C)$ by contradiction. Suppose $u \in U(C)$, since C is a biclique with $|V(C)| \geq \tau_V^i$, u has at least τ_V^i neighbors in G, i.e., $d(u, G) \geq \tau_V^i$. This contradicts with the fact that $d(u, G) < \tau_V^i$. Therefore, the lemma holds. □

Lemma 2 provides a sufficient condition for a vertex to be eliminated s.t. the maximum biclique is preserved. Based on the Lemma 1, Lemma 2 can be iteratively applied to reduce the graph size until no vertices can be eliminated.

The one-hop graph reduction is shown in Algorithm 3. Given a bipartite graph G and thresholds τ_U^i and τ_V^i, the algorithm aims to compute a bipartite graph G_i s.t. $G_i \sqsubseteq_{\tau_U^i, \tau_V^i} G$ by applying the one-hop reduction rule in Lemma 2. We first initialize G_i to be G (line 1), and then we iteratively remove vertices from G_i that satisfy either case (1) (line 4–5) or case (2) (line 6–7) in Lemma 2. The algorithm terminates until no such vertices can be found in G_i. The following lemma shows the time complexity of Algorithm 3.

Lemma 3 *Algorithm 3 requires $O(|G|)$ time.*

Proof Sketch: To implement Algorithm 3 efficiently, we can use a queue Q to maintain the set of vertices satisfying Lemma 2. Each vertex is pushed into and poped from the queue Q at most once. For each vertex v, after removing it from G_i, we need to maintain the degrees of its neighbors and put those neighbors that can be eliminated using Lemma 2 due to decreasing of the degree into the queue Q. This requires $O(d(v, G))$ time. Therefore, the overall time complexity of Algorithm 3 is $O(\sum_{u \in U(G)} d(u, G) + \sum_{v \in V(G)} d(v, G)) = O(|G|)$. □

5.2 Two-hop graph reduction

Next, we explore the two-hop neighbors to further reduce the size of the bipartite graph. For each vertex u, suppose u' is a two-hop neighbor of u, i.e., $N(u', G) \cap N(u, G) \neq \emptyset$. To eliminate u by fully using the information involved within the two-hop neighbors, instead of only considering the degree of u', i.e., $|N(u', G)|$, we consider the number of common neighbors of u and u', i.e., $|N(u', G) \cap N(u, G)|$. To do so, we define the τ-neighbor and τ-degree as follows:

Definition 5 *(τ-Neighbor and τ-degree)* Given a bipartite graph G and a parameter τ, for any $u \in U(G)$ and $u' \in U(G)$, u' is a τ-neighbor of u iff

$$|N(u', G) \cap N(u, G)| \geq \tau$$

For any $u \in U(G)$, the set of τ-neighbors of u is defined as $N_\tau(u, G)$, i.e.,

$$N_\tau(u, G) = \{u' \mid |N(u', G) \cap N(u, G)| \geq \tau\}$$

and the τ-degree of u is defined as the number of vertices in $N_\tau(u, G)$, i.e.,

$$d_\tau(u, G) = |N_\tau(u, G)|$$

Similarly, we can define the τ-neighbor set $N_\tau(v, G)$ and the τ-degree $d_\tau(v, G)$ for any $v \in V(G)$.

Obviously, the τ-neighbor of any vertex u is a subset of a union of u itself and the two-hop neighbors of u. For example, in Fig. 5b, when $\tau = 4$, $N_\tau(v_1, G') = \{v_1, v_2, v_3\}$, because both v_2 and v_3 have ≥ 4 neighbors with v_1.

The following lemma shows how to use the τ-neighbor of a vertex to eliminate the vertex with the given thresholds.

Lemma 4 *Given a bipartite graph G, thresholds τ_U^i and τ_V^i, we have:*

(1) $\forall u \in U(G) : d_{\tau_V^i}(u, G) < \tau_U^i \implies G \ominus u \sqsubseteq_{\tau_U^i, \tau_V^i} G$;
(2) $\forall v \in V(G) : d_{\tau_U^i}(v, G) < \tau_V^i \implies G \ominus v \sqsubseteq_{\tau_U^i, \tau_V^i} G$.

Proof Sketch: We only prove (1), and (2) can be proved similarly. Given a certain vertex $u \in U(G)$ with $d_{\tau_V^i}(u, G) < \tau_U^i$, we need to prove that for any biclique C in G with $|U(C)| \geq \tau_U^i$ and $|V(C)| \geq \tau_V^i$, C is also a biclique in $G \ominus u$. That is, we only need to prove $u \notin U(C)$. Next, we prove $u \notin U(C)$ by contradiction. Suppose $u \in U(C)$, since C is a biclique with $|U(C)| \geq \tau_U^i$ and $|V(C)| \geq \tau_V^i$, for each $u' \in U(C)$, we have:

$$|N(u, C) \cap N(u', C)| = |V(C)| \geq \tau_V^i$$

Algorithm 4: Reduce2Hop(G, τ_U^i, τ_V^i)

Input : Bipartite graph G, thresholds τ_U^i and τ_V^i
Output : A graph G_i s.t. $G_i \sqsubseteq_{\tau_U^i, \tau_V^i} G$

1 $G_i \leftarrow G$;
2 $G_i \leftarrow$ Reduce2H($G_i, U(G_i), \tau_U^i, \tau_V^i$);
3 $G_i \leftarrow$ Reduce2H($G_i, V(G_i), \tau_V^i, \tau_U^i$);
4 **return** G_i;
5 **Procedure** Reduce2H($G_i, U, \tau_U^i, \tau_V^i$)
6 **for each** $u \in U$ **do**
7 $S \leftarrow \emptyset$;
8 **for each** $v \in N(u, G_i)$ **do**
9 **for each** $u' \in N(v, G_i)$ **do**
10 **if** $S.find(u') = \emptyset$ **then**
11 $S \leftarrow S \cup \{(u', 1)\}$;
12 **else**
13 $o \leftarrow S.find(u')$;
14 $o.cnt \leftarrow o.cnt + 1$;
15 $c \leftarrow |\{o \in S | o.cnt \geq \tau_V^i\}|$;
16 **if** $c < \tau_U^i$ **then**
17 $G_i \leftarrow G_i \ominus u$;
18 **return** G_i;

In other words, u' is a τ_V^i-neighbor of u in C, i.e., $u' \in N_{\tau_V^i}(u, C)$. Therefore,

$$|N_{\tau_V^i}(u, C)| = |U(C)| \geq \tau_U^i$$

Consequently, we can derive:

$$d_{\tau_V^i}(u, G) = |N_{\tau_V^i}(u, G)| \geq |N_{\tau_V^i}(u, C)| \geq \tau_U^i$$

This contradicts with the assumption that $d_{\tau_V^i}(u, G) < \tau_U^i$. As a result, the lemma holds. □

Based on Lemma 4 and the transitive property shown in Lemma 1, we are ready to design the two-hop graph reduction algorithm. The pseudocode of the algorithm is shown in Algorithm 4. Since Lemma 4 can be applied for vertices in both $U(G)$ and $V(G)$, the algorithm reduce the bipartite graph G twice, and each time the vertices in one side are reduced using the procedure Reduce2H (line 1–4).

In the Reduce2H procedure (line 5–18), we visit each vertex $u \in U$ to check whether u can be eliminated using Lemma 4 (line 6). We use S to maintain the set of two-hop neighbors of u along with the number of common neighbors with each two-hop neighbor. Specifically, for each two-hop neighbor u' of u, we create a unique entry $o = (u', cnt)$ in S where $o.cnt$ denotes the number of common neighbors for u and u'. In the algorithm, we first search the neighbors $v \in N(u, G_i)$ (line 8) and then search the neighbors $u' \in N(v, G_i)$ to obtain each two-hop neighbor u' (line 9). If the entry for u' does not exist in S, we add u' to S with $cnt = 1$ (line 10–11); otherwise, we obtain the entry o for u' and increase $o.cnt$ by 1 (line 13–14). After processing all two-hop neighbors of u, we maintain a counter c to count the number of τ_V^i-neighbor of u (line 15). Obviously, $c = d_{\tau_V^i}(u, G)$. Therefore, if $c < \tau_U^i$, we can eliminate u from G_i according to Lemma 4 (line 16–17).

Lemma 5 *Algorithm 4 requires* $O(\sum_{u \in U(G)} d(u, G)^2 + \sum_{v \in V(G)} d(v, G)^2)$ *time.*

Proof Sketch: When processing $U(G)$ (line 2), for each $u \in U(G)$ (line 6) and $v \in N(u, G)$ (line 8), we need to process all neighbors u' of v using $O(d(v, G))$ time. Therefore, the total time complexity of the procedure in line 2 is

$$\begin{aligned} &O\Big(\sum_{u \in U(G)} \sum_{v \in N(u,G)} d(v, G)\Big) \\ &= O\Big(\sum_{(u,v) \in E(G)} d(v, G)\Big) \\ &= O\Big(\sum_{v \in V(G)} \sum_{u \in N(v,G)} d(v, G)\Big) \\ &= O\Big(\sum_{v \in V(G)} d(v, G)^2\Big) \end{aligned}$$

Similarly, the total time complexity of the procedure in line 3 is $O(\sum_{u \in U(G)} d(u, G)^2)$. Consequently, the overall time complexity of Algorithm 4 is $O(\sum_{u \in U(G)} d(u, G)^2 + \sum_{v \in V(G)} d(v, G)^2)$. □

Optimizations However, Reduce2Hop is more costly than Reduce1Hop. So we introduce two heuristics, early pruning and early skipping, to further optimize the two-hop reduction algorithm as follows.

(1) Early pruning In Algorithm 4, there is no specific order to process vertices. However, if we process vertices that are more likely to be pruned first, the removal of these vertices may result in more vertices elimination in later iterations. Based on this, we design a score function so that vertices with small scores are more likely to be pruned. A straightforward score is the vertex degree. However, it only considers the vertices in one side and ignores those in the other side. Therefore, for each vertex u, we summarize the degrees for all u's neighbors, and design the score function as follows:

$$\mathsf{score}(u) = \sum_{v \in N(u,G)} d(v, G) \tag{2}$$

The score function considers both the number of neighbors u has and the degrees of the u's neighbors, and is cheap to compute. Given the score function, we can simply modify the algorithm by processing vertices in non-decreasing order of their scores to improve the algorithm performance.

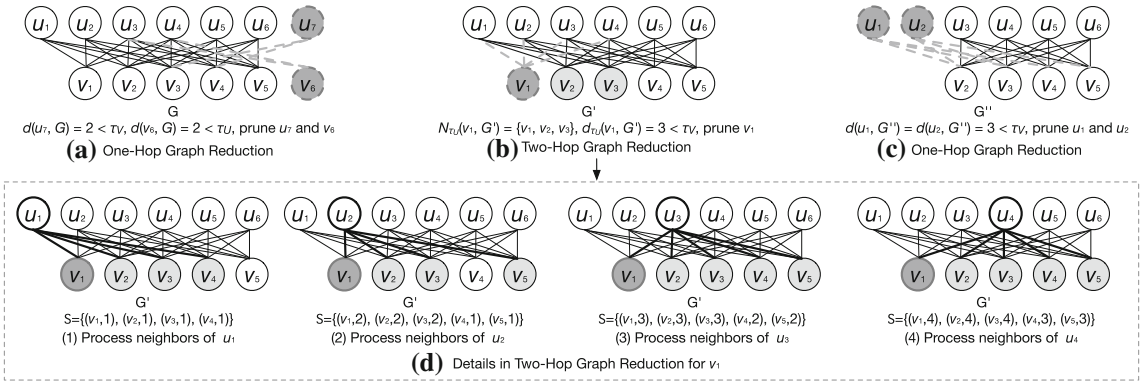


Fig. 5 An example of graph reduction with $\tau_U = 4$, $\tau_V = 4$

(2) Early skipping Then, we proceed to identify some vertices that cannot be pruned using Reduce2Hop before exploring their two-hop neighbors. These vertices can be skipped directly. The following lemma provides a way to do this:

Lemma 6 *For any vertices u, u' and threshold τ, we have:* $u' \in N_\tau(u, G) \iff u \in N_\tau(u', G)$

Proof Sketch: According to Definiton 5, $u' \in N_\tau(u, G)$ is equivalent to $N(u', G) \cap N(u, G) \geq \tau$, which is equivalent to $u \in N_\tau(u', G)$. □

Based on Lemma 6, for any vertex $u' \in U(G)$, if there are more than τ_U^i vertices u with $u' \in N_{\tau_V^i}(u, G)$, we can guarantee that $d_{\tau_V^i}(u', G) \geq \tau_U^i$, and therefore u' can be skipped by Lemma 4 without exploring the two-hop neighbors of u'. To realize this idea, for each vertex $u' \in U(G)$, we use $u'.c$ to maintain the number of processed vertices u s.t. $u' \in N_{\tau_V^i}(u, G)$. When processing u, for each two-hop neighbor u', if $u' \in N_{\tau_V^i}(u, G)$, we increase $u'.c$ by 1. Later on, when processing u', we check whether $u'.c + 1 \geq \tau_U^i$ before exploring the two-hop neighbors of u'. If so, we know that u' cannot be pruned and directly skip u'. Here, we use $u'.c + 1$ to take u' itself into consideration.

5.3 The overall reduction strategy

Based on the above analysis, we can use either one-hop or two-hop reduction to reduce the size of the bipartite graph G. The following lemma shows that the two-hop reduction rule in Lemma 4 has stronger pruning power than the one-hop reduction rule in Lemma 2.

Lemma 7 *Given a bipartite graph G, thresholds τ_U^i and τ_V^i, we have:*

(1) $\forall u \in U(G) : d(u, G) < \tau_V^i \implies d_{\tau_V^i}(u, G) < \tau_U^i$;
(2) $\forall v \in V(G) : d(v, G) < \tau_U^i \implies d_{\tau_U^i}(v, G) < \tau_V^i$.

Proof Sketch: We first prove (1). For any $u \in U(G)$, if $d(u, G) < \tau_V^i$, we know that there does not exist a two-hop neighbor u' of u s.t. $|N(u', G) \cap N(u, G)| \geq \tau_V^i$. Therefore, $d_{\tau_V^i}(u, G) = 0 < \tau_U^i$. (2) can be proved similarly. □

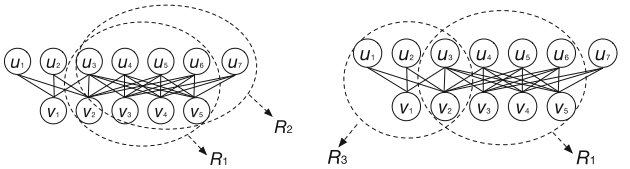
Nevertheless, based on Lemmas 3 and 5, applying one-hop reduction is much more efficient than applying two-hop reduction. Therefore, we design the overall graph reduction strategy as follows:

Reduce Given a bipartite graph G and thresholds τ_U^i and τ_V^i, Reduce iteratively applies one-hop and two-hop reduction strategies on G for MAX_ITER rounds where MAX_ITER is a small constant, and returns the reduced graph G_i. Specifically, in each round, Reduce first applies Reduce1Hop and then further applies Reduce2Hop on the reduced graph.

Example 5 We show the example of the complete graph reduction process in Fig. 5. Given the bipartite graph G in Fig. 1a and thresholds $\tau_U = 4$, $\tau_V = 4$ and MAX_ITER $= 2$, we first apply Reduce1Hop in Fig. 5a. Since $d(u_7, G) = 2 < \tau_V$ and $d(v_6, G) = 2 < \tau_U$, we prune u_7 and v_6. Then, we apply Reduce2Hop in Fig. 5b with the details shown in Fig. 5d. We traverse the one-hop and two-hop neighbors of v_1, and update the entries in S as shown in step (1) to step (4). For example, in step (1), we traverse v_1's neighbor u_1 and two-hop neighbors v_1, v_2, v_3 and v_4, and set $cnt = 1$ for each two-hop neighbor. After visiting all neighbors in step (4), we have three vertices with $cnt = 4$, i.e., $c = d_{\tau_U}(v_1, G') = 3$. According to Lemma 4, since $d_{\tau_U}(v_1, G') < \tau_V$, we prune v_1. After that, we further apply Reduce1Hop in Fig. 5c, and prune vertices u_1 and u_2. By applying Reduce, we save huge search space in biclique search.

6 Diversified top-*k* biclique search

In some applications, one may need to enumerate a set of bicliques. For example, in click farm detection in E-commerce such as Alibaba Group, the fraudulent transactions cannot be fully covered by the maximum biclique. Instead,



(a) Top-2 Maximal Bicliques **(b)** Diversified Top-2 Bicliques

Fig. 6 Top-2 bicliques in G with $\tau_U = 1$ and $\tau_V = 1$

we may need to consider the *maximal biclique*, where none of its superset is also a biclique. However, as the number of maximal bicliques may be exponential in the graph size [11], a possible solution is to compute the top-k results ranked by size, since maximal bicliques with larger size are always more important [23]. However, the top-k results ranked by size are usually highly overlapping, which significantly reduce the effective information of the k results. Motivated by this, we study the problem of the *Diversified Top-k Biclique Search* in this section, aiming to find top-k results that are distinctive and informationally rich.

Firstly, we formally define the *diversified top-k biclique search problem*.

Definition 6 *(Coverage* $\mathsf{cov}(\mathcal{D})$*)* Given a set of bicliques $\mathcal{D} = \{R_1, R_2, ...\}$ in a bipartite graph G, the *coverage* of $\mathcal{D}$, denoted by $\mathsf{cov}(\mathcal{D})$, is the set of edges in G covered by the bicliques in $\mathcal{D}$, i.e., $\mathsf{cov}(\mathcal{D}) = \bigcup_{R \in \mathcal{D}} E(R)$.

Problem statement Given a bipartite graph $G = (U, V, E)$, an integer k, and thresholds of τ_U and τ_V, the problem of diversified top-k biclique search aims to find a set $\mathcal{D}$, such that (1) each biclique $R \in \mathcal{D}$ is a maximal biclique with $|U(R)| \geq \tau_U$ and $|V(R)| \geq \tau_V$, (2) $|\mathcal{D}| \leq k$ and (3) $\mathsf{cov}(\mathcal{D})$ is maximized.

Example 6 We show an example of top-2 bicliques in bipartite graph G with $\tau_U = 1$ and $\tau_V = 1$ in Fig. 6. There are three maximal bicliques in G: R_1 with $U(R_1) = \{u_3, u_4, u_5, u_6\}$ and $V(R_1) = \{v_2, v_3, v_4, v_5\}$; R_2 with $U(R_2) = \{u_3, u_4, u_5, u_6, u_7\}$ and $V(R_2) = \{v_3, v_4, v_5\}$; and R_3 with $U(R_3) = \{u_1, u_2, u_3\}$ and $V(R_3) = \{v_1, v_2\}$. The result of top-2 maximal bicliques ranked by size is $\mathcal{D}_1 = \{R_1, R_2\}$, and the result of diversified top-2 bicliques is $\mathcal{D}_2 = \{R_1, R_3\}$. Although $|R_2| > |R_3|$, it is obvious that $\mathcal{D}_2$ is more favorable since R_2 is highly overlapping with R_1. In other words, $\mathsf{cov}(\mathcal{D}_2) > \mathsf{cov}(\mathcal{D}_1)$.

NP-hardness We show the hardness of the problem by considering the simple case: $k = 1$, $\tau_U = 1$, and $\tau_V = 1$. In this case, the problem becomes the maximum biclique search problem which is NP-hard [38]. Therefore, *the diversified top-k biclique search problem is an NP-hard problem*.

Algorithm 5: TopKBasic(G, k, τ_U, τ_V)

Input : Bipartite graph G, integer k, thresholds τ_U and τ_V
Output : The set of diversified top-k results $\mathcal{D}$

1 $\mathcal{D} \leftarrow \emptyset$;
2 **for** $i = 1$ to k **do**
3 $\quad R_i \leftarrow \mathsf{MBC}^*(G, \tau_U, \tau_V)$;
4 $\quad$ **if** R_i is empty **then**
5 $\quad\quad$ break;
6 $\quad \mathcal{D} \leftarrow \mathcal{D} \cup R_i$;
7 $\quad E(G) \leftarrow E(G) \setminus E(R_i)$;
8 **return** $\mathcal{D}$;

6.1 Baseline solution

In the literature, the problem of maximal biclique enumeration is widely studied [15,29,35–37,41,59]. This leads to a straightforward solution of diversified top-k biclique search: firstly, we can enumerate all the maximal bicliques satisfying the thresholds of τ_U and τ_V, and then we formulate the problem of diversified top-k biclique search as a max k-cover problem. However, in a large-scale bipartite graph, the enumeration is costly and may not be able to terminate. Besides, it is infeasible to keep all the maximal bicliques in memory due to the exponential number of maximal bicliques in large bipartite graphs.

Fortunately, by taking advantage of our efficient maximum biclique search method, we can find the diversified top-k results by repeatedly removing the current maximum biclique from the bipartite graph k times, which follows the framework in a well-studied diversified top-k clique problem [57].

The baseline solution is shown in Algorithm 5. It firstly initiates the result set $\mathcal{D}$ as empty (line 1), and then greedily compute k bicliques to insert into $\mathcal{D}$ (line 2–7), and return $\mathcal{D}$ as the top-k results (line 8). Each time, it invokes MBC^* to compute the maximum biclique R_i in G satisfying the thresholds of τ_U and τ_V (line 3). If R_i is empty, it indicates that no more bicliques satisfying τ_U and τ_V can be found and we can stop searching (line 4–5). Otherwise, we update $\mathcal{D}$ by inserting R_i into it (line 6), and then remove the edges in R_i from G (line 7).

Time complexity We analyze the time cost of Algorithm 5. The time cost is mainly spent on the k times computation of MBC^*, which consists of the graph reduction time and maximum biclique searching time. In MBC^*, we denote the number of subspaces generated for searching result R_j as l_j, where l_j is bounded by $\log(d^U_{\max}(G))$. For result R_j, we use $T_{\text{reduce}}(G)$ to denote the graph reduction time (including one-hop and two-hop graph reduction), and $T_{\text{search}}(G_{i,j})$ to denote the maximum biclique searching time, where $G_{i,j}$ represents the reduced graph in the i-th subspace for R_j. Here, $T_{\text{search}}(G) = O(|V(G)|d^V_{\max}(G)\beta)$ where

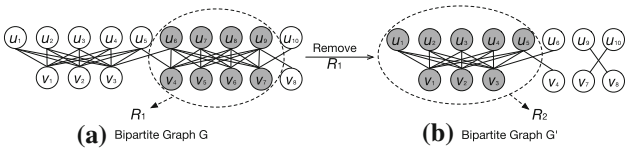


Fig. 7 Search top-2 bicliques in G by TopKBasic

β denotes the number of the maximal bicliques in G as introduced in [59]. Thus, the time cost of Algorithm 5 is $O(\sum_{j=1}^{k}\sum_{i=1}^{l_j}(T_{\text{reduce}}(G)+T_{\text{search}}(G_{i,j}))$.

Example 7 Given a bipartite graph G in Fig. 7a with thresholds $\tau_U = 1$ and $\tau_V = 1$, we adopt Algorithm 5 to find the diversified top-2 bicliques in G:

(1) To find R_1 in G with MBC*, suppose $|C_0| = 10$ and $\tau_V^0 = 5$, we generate two subspaces as follows:

(i) $\tau_U^1 = \left\lfloor \frac{|C_0|}{\tau_V^0} \right\rfloor = 2$, $\tau_V^1 = \left\lfloor \frac{\tau_V^0}{2} \right\rfloor = 2$. We find the maximum biclique C_1^* (marked as gray in Fig. 7a), and we update $|C_1^*| = 16$.

(ii) $\tau_U^2 = \left\lfloor \frac{|C_1^*|}{\tau_V^1} \right\rfloor = 8$, $\tau_V^2 = \left\lfloor \frac{\tau_V^1}{2} \right\rfloor = 1$. We cannot find larger bicliques.

Thus, we obtain $R_1 = C_1^*$ as shown in Fig. 7a. Then, we remove all edges in R_1 from G, and get G' as shown in Fig. 7b. (Here, we omit the vertices with no edges.) (2) To find R_2 in G' with MBC*, suppose $|C_0'| = 6$, and $\tau_V^{0\prime} = 4$, we generate two subspaces as follows:

(i) $\tau_U^{1\prime} = \left\lfloor \frac{|C_0'|}{\tau_V^{0\prime}} \right\rfloor = 1$, $\tau_V^{1\prime} = \left\lfloor \frac{\tau_V^{0\prime}}{2} \right\rfloor = 2$. We find $C_1^{*\prime}$ (marked as gray in Fig. 7b), and update $|C_1^{*\prime}| = 15$.

(ii) $\tau_U^{2\prime} = \left\lfloor \frac{|C_1^{*\prime}|}{\tau_V^{1\prime}} \right\rfloor = 7$, $\tau_V^{2\prime} = \left\lfloor \frac{\tau_V^{1\prime}}{2} \right\rfloor = 1$. We cannot find larger bicliques.

Thus, we obtain $R_2 = C_1^{*\prime}$, as shown in Fig. 7b. It should be noticed that the subspaces generated in G and G' are different. Consequently, for R_1 and R_2, we compute the reduced subgraph by Reduce and the maximum biclique by MBC in each subspace independently.

Finally, we obtain the result set $\mathcal{D} = \{R_1, R_2\}$.

6.2 Advanced diversified top-*k* search

In this subsection, we first analyze the drawbacks of the baseline solution, and then introduce our new diversified top-*k* biclique search approach, based on the idea of deriving the same subspaces for different results to share computation cost among them.

6.2.1 Problem analysis

Drawbacks of TopKBasic The major limitation of TopKBasic is the isolated computation of R_i by MBC*. Recall that in MBC*, we progressively generate subspaces based on the value of the maximum biclique size found so far (line 5–6 in MBC*). We call such subspaces generated for R_i as *a subspace set*. Obviously, for the top-k results, the generated subspace sets are different (e.g., the generated subspace sets of R_1 and R_2 in Example 7). Consequently, both *graph reduction* by Reduce and *maximum biclique search* by MBC in MBC* will be computed independently in each subspace among all the k results, which is costly.

Our idea Intuitively, since the different generated subspace sets lead to the isolated computations of R_i in TopKBasic, we consider to generate the same subspace set for all the k results so as to share the computation among them. Specifically, we fix the subspace set in MBC* as follows: (1) Instead of using the largest biclique size found so far as the lower bound of the optimal solution for generating subspaces, we use a constant c. According to Theorem 1, it is not hard to prove that with c as the lower bound, we can preserve the maximum biclique whose size is larger than c in the derived subspaces. Thus to find the top-k results, we set c as a constant value which is smaller than the size of the k-th result R_k. (2) We fix $\tau_V^0 = d_{\max}^U(G_{\text{ori}})$ where G_{ori} denotes to the original bipartite graph G, and $d_{\max}^U(G_{\text{ori}})$ is guaranteed to be an upper bound of $|V(R)|$ for all the k results. Consequently, with the fixed c and τ_V^0, we can generate the same subspace set for all the k results. We denote such a fixed subspace set as $\mathcal{FS}(G, c)$, or $\mathcal{FS}$ in short if the context is clear.

With the idea of the fixed subspace set $\mathcal{FS}(G, c)$, the following two issues need to be further addressed:

- First, we do not know the size of the k-th result R_k. Although we can set c as a small constant, e.g., $c = \tau_U \times \tau_V$, the τ_U^i and τ_V^i computed based on c in each subspace may be very loose for graph reduction and search space pruning.
- Second, even we can generate the same subspace set for the k results, we still need to remove R_i from bipartite graph G when searching for R_{i+1}, which indicates that the reduced graph and the maximum biclique in each subspace need to be recomputed.

6.2.2 Advanced top-*k* biclique search

To solve the above problems, we first preserve the following three information for each subspace in $\mathcal{FS}(G, c)$:

(1) the thresholds τ_U^i and τ_V^i computed based on c;
(2) the reduced subgraph G_i w.r.t. τ_U^i and τ_V^i;

(3) the maximum biclique C_i^* in G_i that $|U(C_i^*)| \geq \tau_U^i$, $|V(C_i^*)| \geq \tau_V^i$ and $|C_i^*| > c$.

Based on $\mathcal{FS}(G, c)$, we address the two issues as follows:

- To address the first issue, instead of initiating c as a very small constant to cover all the results which leads to loose thresholds, we search for the top-k results by progressively relaxing c. Specifically, we use a lower bound of the size of the top-1 biclique to initiate c. Then, we generate $\mathcal{FS}(G, c)$ and search results in it. Once we cannot find enough results within $\mathcal{FS}(G, c)$, we relax c to c' by multiplying a factor α, where $0 < \alpha < 1$, and regenerate $\mathcal{FS}(G, c')$ to cover more results.
- To address the second issue, instead of recomputing the subgraph by Reduce and maximum biclique by MBC in each subspace when searching for the next result, we apply light-costed subgraph updating and on-demand maximum biclique searching in $\mathcal{FS}(G, c)$. Specifically, as we have maintained the reduced subgraph G_i and the maximum biclique C_i^* in each subspace in $\mathcal{FS}(G, c)$, when we need to remove R_j from G, we can update G_i by simply eliminating the edges in R_j from G_i. Moreover, we only need to recompute C_i^* that has overlaps with R_j. Otherwise, C_i^* remains unchanged even when G_i is updated.

The advanced algorithm The proposed algorithm is shown in *Algorithm* 6. It firstly initiates $\mathcal{D}$ as an empty set and R_0 as empty (line 1). We set the value of c as the size of the initial biclique in G found by InitMBC, which is a lower bound of the size of top-1 result R_1 (line 2). We use $flag$ to indicate whether or not we need to generate $\mathcal{FS}$ with constant c, initialized as **true**, and i to denote the index of the top-k results, initialized as 0 (line 3). Then, we search for the top-k results (line 4–16). We first invoke GenSubSpaces to generate $\mathcal{FS}$ when $flag$ is **true** , and after generation, we set $flag$ as **false** (line 5–6). With $\mathcal{FS}$, we invoke FixedMBC* to search the maximum biclique in each subspace, respectively, and return the one with the largest size as the result R_{i+1} (line 7). If R_{i+1} is empty, it indicates that no bicliques larger than c can be found in $\mathcal{FS}$. Here, we will terminate the computation if $c < \tau_U \times \tau_V$, as in this case, no more bicliques satisfying τ_U and τ_V can be found (line 9–10). Otherwise, we relax c by multiplying a factor α, where $0 < \alpha < 1$, and set $flag$ as **true** to indicate that we need to regenerate $\mathcal{FS}$ (line 11–12). If R_{i+1} is not empty, we add R_{i+1} into $\mathcal{D}$ and update G by deleting edges in R_{i+1} from G (line 13–16). Finally, we return $\mathcal{D}$ as the diversified top-k results (line 17).

Procedure GenSubSpaces generates the fixed subspace set based on c. It follows similar procedures in MBC*, except that in GenSubSpaces, each iteration generates a pair of τ_U^{i+1} and τ_V^{i+1} based on the fixed constant c (line 22–23) and searches for the maximum biclique whose size is larger than c (line 25). Here, we slightly modify MBC that we use a constant c rather than an initial biclique for size pruning, and if no biclique larger than c can be found, we directly return an empty biclique. Besides, we further preserve the reduced subgraph G_i, thresholds τ_U^i and τ_V^i, and maximum biclique C_i^* as the *subspace information* in $\mathcal{FS}$ (line 26), in order to share the computation cost among all results preserved in $\mathcal{FS}$.

Algorithm 6: TopK(G, k, τ_U, τ_V)

Input : Bipartite graph G, integer k, thresholds τ_U and τ_V
Output : The set of diversified top-k results $\mathcal{D}$

1 $\mathcal{D} \leftarrow \emptyset$; $R_0 \leftarrow \emptyset$;
2 $c \leftarrow |\mathsf{InitMBC}(G, \tau_U, \tau_V)|$;
3 $flag \leftarrow$ **true**; $i \leftarrow 0$;
4 **while** $i < k$ **do**
5 **if** $flag =$ **true then**
6 $\mathcal{FS} \leftarrow \mathsf{GenSubSpaces}(G, \tau_U, \tau_V, c)$; $flag \leftarrow$ **false**;
7 $R_{i+1} \leftarrow \mathsf{FixedMBC}^*(\mathcal{FS}, R_i, c)$;
8 **if** R_{i+1} is empty **then**
9 **if** $c < \tau_U \times \tau_V$ **then**
10 break;
11 **else**
12 $c = \alpha \times c$; $flag \leftarrow$ **true**;
13 **else**
14 $\mathcal{D} \leftarrow \mathcal{D} \cup R_{i+1}$;
15 $E(G) \leftarrow E(G) \setminus E(R_{i+1})$;
16 $i \leftarrow i + 1$;
17 **return** $\mathcal{D}$;
18 **Procedure** GenSubSpaces(G, τ_U, τ_V, c)
19 $\tau_V^0 \leftarrow d_{\max}^U(G)$;
20 $i \leftarrow 0$;
21 **while** $\tau_V^i > \tau_V$ **do**
22 $\tau_U^{i+1} \leftarrow \max(\left\lfloor \frac{c}{\tau_V^i} \right\rfloor, \tau_U)$;
23 $\tau_V^{i+1} \leftarrow \max(\left\lfloor \frac{\tau_V^i}{2} \right\rfloor, \tau_V)$;
24 $G_{i+1} \leftarrow \mathsf{Reduce}(G, \tau_U^{i+1}, \tau_V^{i+1})$;
25 $C_{i+1}^* \leftarrow \mathsf{MBC}(G_{i+1}, \tau_U^{i+1}, \tau_V^{i+1}, c)$;
26 $\mathcal{FS} \leftarrow \mathcal{FS} \cup (G_{i+1}, \tau_U^{i+1}, \tau_V^{i+1}, C_{i+1}^*)$;
27 $i \leftarrow i + 1$;
28 **return** $\mathcal{FS}$;
29 **Procedure** FixedMBC*($\mathcal{FS}, R, c$)
30 $C^* \leftarrow \emptyset$;
31 **for** $(G_i, \tau_U^i, \tau_V^i, C_i^*)$ in $\mathcal{FS}$ **do**
32 $E(G_i) \leftarrow E(G_i) \setminus E(R)$;
33 **if** $E(C_i^*) \cap E(R) \neq \emptyset$ **then**
34 $C_i^* \leftarrow \mathsf{MBC}(G_i, \tau_U^i, \tau_V^i, c)$;
35 **if** $|C_i^*| > |C^*|$ **then**
36 $C^* \leftarrow C_i^*$;
37 **return** C^*;

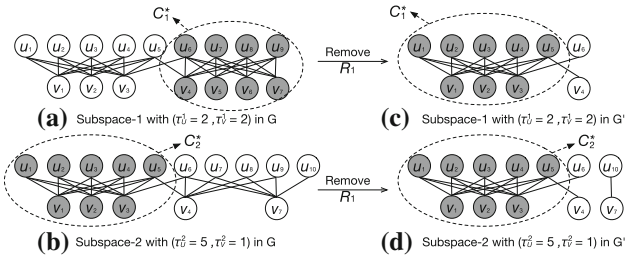


Fig. 8 Search top-2 bicliques in G by TopK

Procedure FixedMBC* searches for the maximum biclique in the fixed subspace set $\mathcal{FS}$. Firstly, we initiate C^* as empty (line 30), and then progressively update it with larger bicliques found in each subspace (line 31–36). For the i-th subspace in $\mathcal{FS}$, we first update the subgraph G_i by eliminating all edges in R from G_i, where R is the last diversified biclique result we found (line 32). Then if C_i^* overlaps with R, which indicates that the maximum biclique in current subspace has changed, we recompute C_i^* by MBC (line 33–34). Otherwise, C_i^* remains unchanged, and there is no need to update it. We update C^* if we find larger biclique (line 35–36), and finally return C^* as the result(line 37).

Time complexity We analyze the time cost of Algorithm 6. The time cost mainly consists of the graph reduction time (in GenSubSpaces) and maximum biclique searching time (in GenSubSpaces and FixedMBC*).

Firstly, for graph reduction, suppose we generate k' subspace sets to cover all the top-k results by invoking GenSubSpaces. Here, k' is bounded by $\log_\alpha(\frac{1}{c_0})$, where c_0 is a lower bound of the size of the top-1 biclique R_1 ($0 < \alpha < 1$). In each subspace set $\mathcal{FS}_m$ ($1 \le m \le k'$), we denote the number of subspaces as l_m, where l_m is bounded by $\log(d^U_{\max}(G))$. Then, the total graph reduction time is $O(\sum_{m=1}^{k'} \sum_{i=1}^{l_m} T_{\text{reduce}}(G))$. Secondly, for maximum biclique search, in GenSubSpaces, we need to compute the maximum biclique in all subspaces in $\mathcal{FS}_m$, while in FixedMBC*, we only need to compute the maximum biclique when needed. Suppose for result R_j, we need to compute $C^*_{1,j}, C^*_{2,j}, ..., C^*_{l'_j,j}$, where $C^*_{i,j}$ denotes the i-th maximum biclique that need to be recomputed on subgraph $G_{i,j}$ to obtain R_j. Here, $l'_j \le l_m$ for R_j preserved in $\mathcal{FS}_m$. Then, the total maximum biclique search time is $O(\sum_{j=1}^{k} \sum_{i=1}^{l'_j} T_{\text{search}}(G_{i,j}))$. Consequently, the time cost of Algorithm 6 is $O(\sum_{m=1}^{k'} \sum_{i=1}^{l_m} T_{\text{reduce}}(G) + \sum_{j=1}^{k} \sum_{i=1}^{l'_j} T_{\text{search}}(G_{i,j}))$, where $k' \le \log_\alpha(\frac{1}{|R_1|})$. Note that in practice, we observe that k' is much smaller than k, and for result R_j preserved in $\mathcal{FS}_m$, l'_j is also much smaller than l_m in most cases.

Example 8 Given a bipartite graph G in Fig. 7a with thresholds $\tau_U = 1$ and $\tau_V = 1$, we adopt Algorithm 6 to find the diversified top-2 bicliques in G as shown in Fig. 8.

Suppose we have $c = 10$ and $\tau_V^0 = 5$. We generate $\mathcal{FS}(G, c)$ consisting of two fixed subspaces:

(i) $\tau_U^1 = \left\lfloor \frac{c}{\tau_V^0} \right\rfloor = 2$, $\tau_V^1 = \left\lfloor \frac{\tau_V^0}{2} \right\rfloor = 2$. The reduced subgraph G_1 is shown in Fig. 8a with the maximum biclique C_1^* marked as gray;

(ii) $\tau_U^2 = \left\lfloor \frac{c}{\tau_V^1} \right\rfloor = 5$, $\tau_V^2 = \left\lfloor \frac{\tau_V^1}{2} \right\rfloor = 1$. The reduced subgraph G_2 is shown in Fig. 8b with the maximum biclique C_2^* marked as gray.

Based on $\mathcal{FS}(G, c)$, we search for the top-2 diversified bicliques as follows:

(1) With the preserved maximum bicliques in subspace set, we obtain $R_1 = C_1^*$ whose size is maximized. Then, we remove all edges in R_1 from G and get G'.

(2) After found R_1, we can update G_1 and G_2 by directly removing edges in R_1 from them, as shown in Fig. 8c, d, respectively (here we omit vertices with no edges). Furthermore, we only need to recompute C_1^* as it overlaps with R_1, and skip C_2^* as it remains unchanged. Then, we obtain $R_2 = C_2^*$. Finally, we obtain the result $\mathcal{D} = \{R_1, R_2\}$. Compared with TopKBasic, benefiting from the fixed subspace set, TopK saves the cost by sharing the computation of graph reduction and maximum biclique search in subspaces among the results preserved in $\mathcal{FS}(G, c)$.

6.3 Optimization strategies

In Algorithm 6, the computation cost mainly consists of two parts: (1) the graph reduction when generating $\mathcal{FS}$ in GenSubSpaces, and (2) the updating of maximum biclique C_i^* in FixedMBC*. To further save the computation cost, we propose the following two optimizations.

Global size pruning In GenSubSpaces, we apply Reduce on G in each subspace to reduce the graph size, which is costly since G is large. However, in $\mathcal{FS}(G, c)$, we search for biclique whose size is larger than c. Based on this, before we apply Reduce on G in each subspace w.r.t. τ_U^i and τ_V^i, we can firstly prune all vertices that cannot be involved in a biclique with size larger than c, so as to share the computation among all subspaces in $\mathcal{FS}(G, c)$. Although we do not know the biclique size before searching, for vertex u, we could use the summarization of the degree for all u's neighbors as an upper bound for the size of biclique that involves u. Following Definiton 4, with the size constraint c, we use $G' \sqsubseteq_c G$ to denote that G' is an MBC-preserved graph of G w.r.t. c. We derive the following lemma:

Lemma 8 *Given a bipartite graph G, and size constraint c, we have:*

(1) $\forall u \in U(G)$: $\sum_{v \in N(u,G)} d(v, G) \leq c \implies G \ominus u \sqsubseteq_c G$;
(2) $\forall v \in V(G)$: $\sum_{u \in N(v,G)} d(u, G) \leq c \implies G \ominus v \sqsubseteq_c G$.

We omit the proof here. Lemma 8 provides a sufficient condition for a vertex to be eliminated s.t. the maximum biclique whose size is larger than c is preserved. Based on the Lemma 1, Lemma 8 can be iteratively applied to reduce the graph size until no vertices can be eliminated.

We can simply modify GenSubSpaces in Algorithm 6 by applying the global size pruning rule on G to get G' first, and then iteratively generate subspaces by applying Reduce on G'.

Lazy candidate refining In FixedMBC*, when searching for R_{j+1} after found R_j, the case of C_i^* overlapping with R_j indicates that C_i^* is not up to date, thus we recompute C_i^* by adopting MBC. However, it is not necessary to update C_i^* immediately if it cannot be R_{j+1}, thus we decide to refine the candidates in a lazy manner. Specifically, in FixedMBC*, when C_i^* that overlaps with R_j is no larger than the optimal biclique C^* found so far, instead of directly updating C_i^* by MBC which is costly, we label it with $lazy = \textbf{true}$, and will not recompute it until it could be the maximum one. To apply the lazy refine strategy, we modify GenSubSpaces by initiating $lazy = \textbf{false}$ for C_i^* in all subspaces. Then, we modify FixedMBC* as follows:

(1) Before updating subspaces in $\mathcal{FS}$ and searching for R_{j+1}, we first traverse all subspaces to get the update-to-date maximum bicliques, i.e., those who do not have overlaps with R_j and have $lazy = \textbf{false}$. Among all these bicliques, we use the size of the largest one as the lower bound of $|R_{j+1}|$, denoted as $lb(R_{j+1})$.
(2) Then in all subspaces: (i) For C_i^* with $lazy = \textbf{true}$, if $|C_i^*| > lb(R_{j+1})$, we update C_i^* by MBC and set $lazy = \textbf{false}$; Otherwise, we skip C_i^*. (ii) For C_i^* with $lazy = \textbf{false}$ but overlaps with R_j, if $|C_i^*| > lb(R_{j+1})$, we update C_i^* by MBC; Otherwise, we set $lazy = \textbf{true}$ for C_i^* and skip it. (iii) For C_i^* with $lazy = \textbf{false}$ and does not overlap with R_j, there is no need to refine it as it is already up-to-date.
(3) Finally, we return C^* as the biclique with the largest size among all up-to-date candidates with $lazy = \textbf{false}$.

7 Performance studies

In this section, we show the performance studies. We first present the experimental results by comparing the proposed maximum biclique search algorithm MBC*, with the following two baseline algorithms:

(1) MBC: MBC is developed based on the algorithm in [59], where the code is obtained from the authors, with the pruning rules in Algorithm 1 added.

(2) MAPEB: MAPEB is developed based on the parameterized algorithm APEB in [14]. Given a bipartite graph G and an integer p, APEB aims to find a biclique C in G with at least p edges (where (G, p) is called a yes-instance) , or report that no such biclique exists. Naturally, we extend APEB with the binary search technique to find the maximum biclique C^*. We denote the lower bound and the upper bound of $|C^*|$ as lb and ub, respectively. The basic idea is that we iteratively set $p = \lfloor \frac{lb+ub}{2} \rfloor$, and adopt APEB to compute if (G, p) is a yes-instance: if it is, we update C^* as the found biclique C and set $lb = |C| + 1$; otherwise, we update $ub = p - 1$. We stop the computation when $lb > ub$, and return C^*. We initialize C^* as $\emptyset$, lb as $\tau_U \times \tau_V$, and ub as the maximum $\mathsf{score}(u)$ (defined in Eq. 2) among all vertices u in G. We also add pruning rules for size constraints of τ_U and τ_V. We call the extended algorithm MAPEB.

We evaluate our algorithms in two aspects: (1) the effectiveness of the graph reduction techniques and optimization strategies used in MBC*, and (2) the efficiency and scalability of maximum biclique search by comparing MBC* with MBC and MAPEB. Then, we show the performance of the diversified top-k biclique search by comparing the proposed algorithm TopK with the baseline algorithm TopKBasic. A case study of anomaly detection on real datasets obtained from Alibaba Group is further described to demonstrate the resultant quality by applying our method. Unless otherwise specified, experiments are conducted with $\tau_U = 3, \tau_V = 3$ by default. All of our experiments are performed on a machine with an Intel Xeon E5-2650 (32 Cores) 2.6GHz CPU and 128GB main memory running Linux.

Datasets We use 18 real datasets selected from different domains with various data properties, including the ones used in existing works. The detailed statistics of the datasets are shown in Table 1. The first 13 datasets are obtained from KONECT[1]. The last five datasets are real datasets obtained from the E-Commerce company Alibaba Group. Here, the AddCart20 and AddCart18 datasets include data of customers adding products into cart in 1 day (sampled from data in 2020) and 10 days (sampled from data in 2018), respectively. The Transaction20 and Transaction18 datasets include data of customers purchasing products in 3 days (sampled from data in 2020) and 15 days (sampled from data in 2018), respectively. Additionally, the LabeledAddCart dataset includes fraudulent transactions labels that we utilize as the ground truth in the case study.

[1] http://konect.cc/.

Table 1 Dataset statistics

Dataset	Category	$\|U\|$	U Type	$\|V\|$	V Type	$\|E\|$	E Type
Writers	Authorship	89,355	Writer	46,213	Work	144,340	Authorship
YouTube	Affiliation	124,325	User	94,238	Group	293,360	Membership
Github	Authorship	56,519	User	120,867	Project	440,237	Membership
BookCrossing	Rating	105,278	User	340,523	Book	1,149,739	Rating
StackOverflow	Rating	545,195	User	96,678	Post	1,301,942	Favorite
Teams	Affiliation	901,130	Athlete	34,461	Team	1,366,466	Membership
ActorMovies	Affiliation	127,823	Movie	383,640	Actor	1,470,404	Appearance
TVTropes	Feature	64,415	Work	87,678	Trope	3,232,134	HasFeature
Wikipedia	Feature	2,036,440	Article	1,853,493	Category	3,795,796	Inclusion
Flickr	Affiliation	499,610	User	395,979	Group	8,545,307	Membership
DBLP	Authorship	1,425,813	Author	4,000,150	Publication	8,649,016	Authorship
LiveJournal	Affiliation	3,201,203	User	7,489,073	Group	112,307,385	Membership
WebTrackers	Hyperlink	27,665,730	Domain	12,756,244	Tracker	140,613,762	Inclusion
LabeledAddCart	MISC	78,582,023	Customer	23,827,661	Product	184,265,522	AddCart
Transaction20	MISC	38,711,236	Customer	18,206,012	Product	229,852,398	Purchasing
AddCart20	MISC	68,016,875	Customer	34,660,953	Product	243,843,277	AddCart
AddCart18	MISC	141,839,807	Customer	65,589,796	Product	1,307,950,593	AddCart
Transaction18	MISC	272,227,190	Customer	75,350,951	Product	1,319,706,942	Purchasing

Table 2 Graph reduction on TVTropes

k	(τ_U^k, τ_V^k)	$\|U\|$	$\|V\|$	$\|E\|$	$\|C_k^*\|$	r_k (%)
0	(3, 3)	64,415	87,678	3,152,266	6,045	97.53
1	(3, 928)	15	6,088	32,991	5,564	1.02
2	(5, 464)	40	5,823	62,913	5,564	1.95
3	(11, 232)	59	2,247	43,602	5,564	1.35
4	(23, 116)	36	78	1,903	5,564	0.06
7	(191, 14)	1,259	115	46,776	5,564	1.45
8	(397, 7)	3,899	59	66,219	5,564	2.05
9	(863, 3)	8,889	27	63,251	6,045	1.96

Table 3 Graph reduction on BookCrossing

k	(τ_U^k, τ_V^k)	$\|U\|$	$\|V\|$	$\|E\|$	$\|C_k^*\|$	r_k (%)
0	(3, 3)	15,330	46,068	599,593	880	52.15
1	(3, 110)	154	9,284	89,550	840	7.79
2	(7, 55)	194	2,020	46,471	880	4.04
3	(16, 27)	236	496	23,155	880	2.01
4	(32, 13)	272	138	10,773	880	0.94
5	(67, 6)	468	70	8,910	880	0.77

7.1 Graph reduction and optimizations

In this subsection, we test the effectiveness and efficiency of the graph reduction techniques and optimization strategies used in the algorithm MBC*.

Effectiveness of graph reduction We test the effectiveness of the proposed one-hop and two-hop graph reduction techniques on datasets of TVTropes and BookCrossing, and show the results in Tables 2 and 3, respectively. We set MAX_ITER in Reduce as 2. Experiments on other datasets have similar outcomes. In Tables 2 and 3, we list τ_U^k, τ_V^k and the number of vertices and edges of the reduced graph in each iteration k in MBC*. We also list the size of C_k^* found in each iteration. We compute the compression ratio r_k as the value of dividing reduced graph size by its original size. In iteration 0, we show the results of graph G_0 reduced by $\tau_U = 3$ and $\tau_V = 3$, as a comparison. We omit the results in the iterations where the reduced graphs are empty. From the results, we can see that in each iteration, we adopt much more strict τ_U^k and τ_V^k constraints rather than τ_U and τ_V. Therefore, by utilizing the graph reduction techniques, we get much smaller reduced graphs, e.g., compression ratio of 0% (omitted in the table) to 2.05% by using τ_U^k and τ_V^k in our progressively bounding framework v.s. 97.53% by using τ_U and τ_V as shown in Table 2. This saves huge search space and accelerates the biclique computation greatly.

Efficiency of graph reduction We conduct experiments on LiveJournal and WebTrackers to compare the performance of the basic algorithms with the optimized versions. We denote the basic version of Algorithm 2 as BASIC, the algorithm with early pruning strategy introduced in Sect. 5.2 as OPT$_1$, and the algorithm with early skipping strategy introduced in

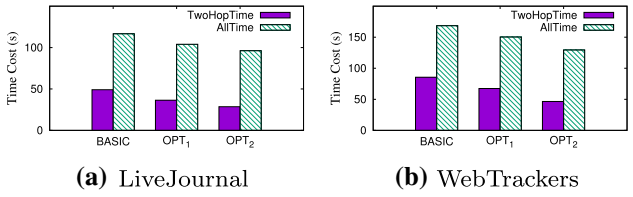


Fig. 9 Optimization strategies

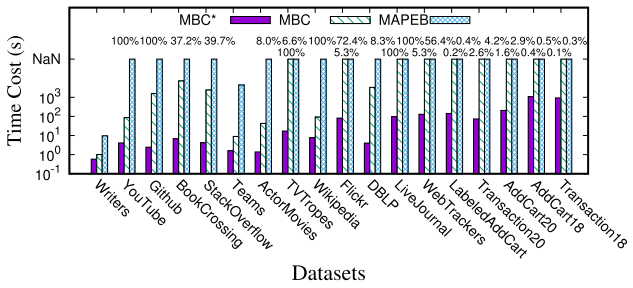


Fig. 10 $C^*_{3,3}$ search on all datasets

Sect. 5.2 as OPT_2 based on OPT_1. The results are shown in Fig. 9, with the two-hop graph reduction time cost denoted as TwoHopTime, and the total time cost denoted as AllTime. We can see that for TwoHopTime in LiveJournal, OPT_2 is about 21.41% faster than OPT_1, and 41.74% faster than BASIC. Consequently, OPT_2 accelerates AllTime by around 17.56% w.r.t. the BASIC version. For TwoHopTime in WebTrackers, OPT_2 is about 30.9% faster than OPT_1, and 45.7% faster than BASIC. Consequently, OPT_2 accelerates AllTime by around 23.2% w.r.t. the BASIC version. When comparing with the baseline algorithm in the following experiments, we apply all the optimization techniques.

7.2 MBC* versus baseline algorithms

In this subsection, we compare the performance of MBC*, MBC and MAPEB on maximum biclique search by: (1) conducting experiments on all datasets; (2) varying τ_U and τ_V thresholds on both small-sized and large-sized graphs; (3) varying graph density; (4) varying graph scale.

In all experiments, we set the maximum processing time as 24 h, and if the methods cannot finish computing, we denote the time cost as NaN. For those experiments that cannot finish within 24 h, we also report the quality ratio above the corresponding bars, which is calculated as:

$$\text{quality ratio} = \frac{\text{the size of current best biclique}}{\text{the size of the maximum biclique}}$$

Note that it is possible that the quality ratio is 100% while the algorithm cannot finish, because the size of the maximum biclique is unknown before the algorithm finishes.

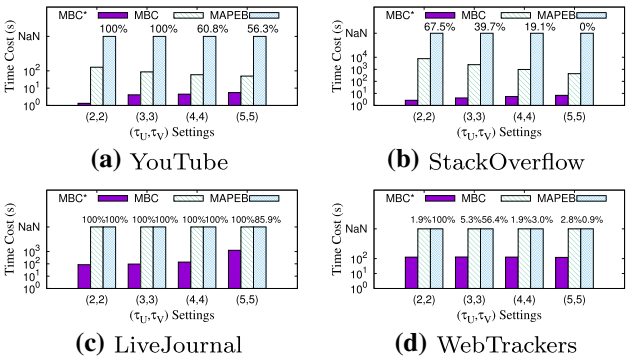
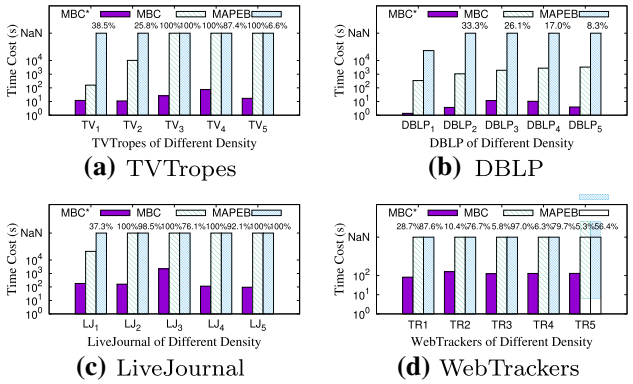


Fig. 11 C^* search by varying τ_U and τ_V

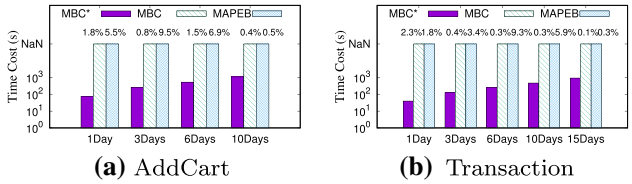
All datasets In this experiment, we test the efficiency of $C^*_{3,3}$ search in all datasets by comparing MBC* with MBC and MAPEB, and report the processing time in Fig. 10. From Fig. 10, we can see that when the size of dataset is relatively small, e.g., around 0.1 million edges in Writers, MBC* and MBC can both find $C^*_{3,3}$ efficiently. As the graph size scales up, e.g., for the graphs with millions of edges such as BookCrossing and StackOverflow, MBC takes hours to compute the results, while MBC* only takes seconds. Furthermore, when the graph size grows up to around 1 billion edges such as AddCart and Transaction, MBC cannot finish computing within 24 h, while MBC* only takes minutes to compute the results. MAPEB, however, fails to finish computing for most cases. Moreover, for most time-out cases, the bicliques found by MBC and MAPEB are far smaller than the maximum bicliques. From the results shown in Fig. 10, we can see that MBC* is much more efficient and scalable than both MBC and MAPEB on all datasets.

Varying τ_U and τ_V thresholds We vary τ_U and τ_V thresholds to compute C^* and illustrate the performance of MBC*, MBC and MAPEB in Fig. 11. Figure 11 shows that MBC can process small graphs (YouTube and StackOverflow) but fails in processing large graphs (LiveJournal and WebTrackers). For small graphs, when τ_U and τ_V get larger, the time cost of MBC decreases. This is because as τ_U and τ_V get larger, MBC can filter more search branches. For large graphs, MBC cannot finish computing within 24 h, since the search space is huge and MBC is stuck in local search. MAPEB cannot finish computing for all cases. The main reason is that MAPEB is developed based on APEB [14], which mainly benefits from the early termination as soon as the yes-instance is found. However, to find the maximum biclique, we will encounter no-instances in the binary search process in MAPEB. For the no-instance case (G, p), for each vertex $u \in U(G)$ ($v \in V(G)$ resp.), APEB has to enumerate all the combinations (with size constraints of $\geq \tau_V$ ($\geq \tau_U$ resp.) and $\leq \lceil \sqrt{p} \rceil$) of u's (v's resp.) neighbors and induce biclique for each combination correspondingly, which is very costly. In comparison, MBC*



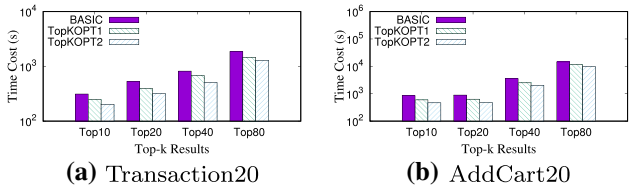
(a) TVTropes (b) DBLP (c) LiveJournal (d) WebTrackers

Fig. 12 $C^*_{3,3}$ search by varying graph density



(a) AddCart (b) Transaction

Fig. 13 $C^*_{3,3}$ search by varying graph scale



(a) Transaction20 (b) AddCart20

Fig. 14 Optimization strategies in TopK

is orders of magnitude faster than both MBC and MAPEB on all settings. For most cases, when τ_U and τ_V get larger, the time cost of MBC* slightly increases. This is because in most real cases, as τ_U and τ_V get larger, $|C^*|$ becomes smaller. Thus, MBC* generates relatively looser τ_U^k and τ_V^k constraints, which results in larger reduced graph. Specifically, in WebTrackers, the processing time is steady. This is because for all τ_U and τ_V settings in this experiment, $|C^*|$ in WebTrackers is relatively large, and consequently τ_U^k and τ_V^k are quite strict. In general, the high efficiency of MBC* mainly benefits from the effective progressive bounding framework with graph reduction techniques, which saves enormous search space in biclique search.

Varying graph density In this experiment, we test the effect of graph density on the performance, and demonstrate the results in Fig. 12. We prepare graphs with different density by sampling edges in the original graph. For example, we sample 20%, 40%, 60%, 80% and 100% edges in TVTropes, and denote these (sub)graphs as TV_1, TV_2, TV_3, TV_4 and TV_5 in ascending order of density. Figure 12 shows that as the graphs grow denser, MBC takes longer time to find the maximum bicliques, or cannot finish computing within 24 h. Although MAPEB may output larger bicliques than MBC sometimes (e.g., on dataset of WebTrackers), since it may find yes-instances efficiently with some appropriate p during the binary search, it cannot finish computing for most cases due to the inefficiency of the no-instances in the binary search. In contrast, MBC* is orders of magnitude faster than both MBC and MAPEB on all settings. It is worth noting that for dense graphs, MBC* also finds maximum bicliques efficiently. For example, in Fig. 12c, as the graphs grow denser from LJ_3 to LJ_5, the processing time of MBC* decreases. The reason is that MBC* can find larger C_k^* in denser graphs. This helps improve the τ_U^k and τ_V^k thresholds and lead to small reduced graphs (or even empty) in the progressive bounding framework. Therefore, MBC* finds maximum biclique efficiently on both sparse and dense graphs.

Varying graph scale The effects of graph size on the performance show scalability. We prepare datasets by obtaining 1, 3, 6 and 10 days data of AddCart18, and 1, 3, 6, 10 and 15 days data of Transaction18. We list the statistics in Table 4, and report the results in Fig. 13. In Fig. 13, we can see that both MBC and MAPEB cannot finish computing within 24 h on all datasets and the reported bicliques are much smaller than the maximum bicliques. In contrast, the processing time of MBC* increases steadily as the graph scales up. For graphs of AddCart10d and Transaction15d, which both consist of about 1.3 billion edges, MBC* costs 18 min and 15 min to compute the results respectively, which is quite efficient. To the best of our knowledge, no existing solutions can find maximum bicliques in bipartite graphs at this scale.

7.3 TopK versus TopKBasic

In this subsection, we test the efficiency of TopK and TopKBasic on diversified top-k biclique search. We first test the efficiency of the proposed optimizations of global size pruning and lazy candidate refining in TopK. Then, we compare TopK with TopKBasic by: (1) varying the result number k; (2) varying τ_U and τ_V thresholds; and (3) varying graph density. In this subsection, we set $k = 80$, $\tau_U = 3$ and $\tau_V = 3$ by default unless otherwise specified. Besides, in TopK, we relax the lower bound c by multiplying factor α to include more results, where $\mathcal{FS}$ with smaller c can preserve more results, while $\mathcal{FS}$ with larger c can generate tighter bounds in subspaces. In compromise, we set $\alpha = 0.7$ in this subsection. Moreover, we set the maximum processing time as 24 h, and if the computation is not finished, we denote the time cost as NaN.

Efficiency of optimizations We conduct experiments on Transaction20 and AddCart20 to compare the basic TopK algorithm with the optimized versions. We denote the basic version of Algorithm 6 as BASIC, the algorithm with global

Table 4 Statistics of AddCart and Transaction

Dataset	$\|U\|$	$\|V\|$	$\|E\|$
AddCart1d	36,610,265	18,840,419	112,796,688
AddCart3d	78,574,410	35,834,266	362,528,389
AddCart6d	107,870,369	48,056,268	768,628,469
AddCart10d	141,839,807	65,589,796	1,307,950,593
Transaction1d	57,324,865	14,381,171	99,906,746
Transaction3d	133,563,771	30,702,475	305,137, 702
Transaction6d	166,496,732	45,016,333	490,500, 877
Transaction10d	231,377,734	59,688,447	872,112, 829
Transaction15d	272,227,190	75,350,951	1,319,706, 942

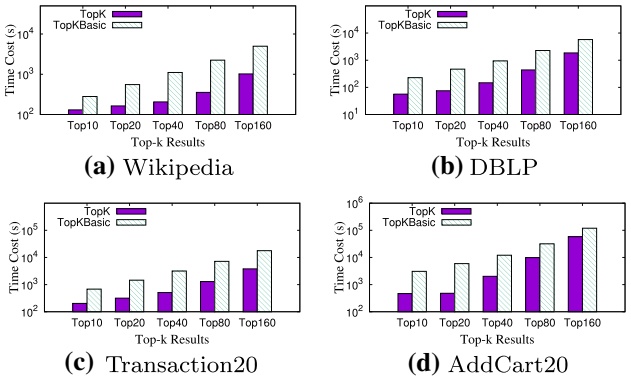


Fig. 15 Top-k search by varying k

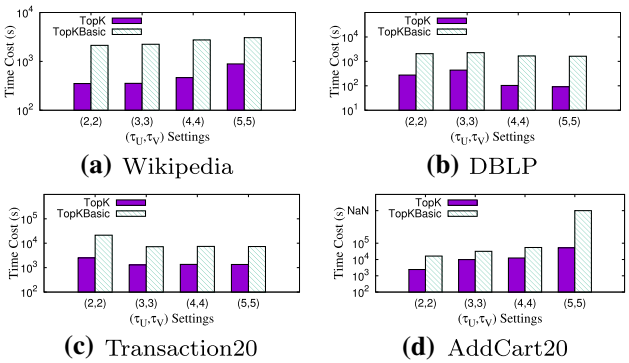


Fig. 16 Top-k search by varying τ_U and τ_V

size pruning strategy introduced in Sect. 6.3 as TopKOPT$_1$, and the algorithm with lazy candidate refining strategy introduced in Sect. 6.3 as TopKOPT$_2$ based on TopKOPT$_1$. We show the time cost to compute the top-10, 20, 40, and 80 bicliques on Transaction20 and AddCart20 respectively in Fig. 14. From the result, we can see that on Transaction20, TopKOPT$_1$ is 21.27% faster than BASIC on average, which mainly benefits from the computation sharing among subspaces in $\mathcal{FS}$ by pruning vertices with size constraint first. Furthermore, TopKOPT$_2$ is 18.37% faster than TopKOPT$_1$ on average, which mainly saves the computation cost in unnecessary maximum biclique updating in subspaces. Consequently, TopKOPT$_2$ accelerates the computation by 35.81% on average w.r.t. the BASIC version on Transaction20. Similarly, on AddCart20, TopKOPT$_1$ is 27.73% faster than BASIC, and TopKOPT$_2$ further improves TopKOPT$_1$ by 19.92% on average. In total, TopKOPT$_2$ accelerates the computation by 42.03% on average on Addcart20.

In the following experiments, when comparing TopK with the baseline algorithm TopKBasic, we apply all the optimization techniques.

Varying results number k In this experiment, we test the efficiency by varying the results number k and report the processing time of TopK and TopKBasic in Fig. 15. We set k as 10, 20, 40, 80 and 160, respectively. For both TopK and TopKBasic, when k increases, the time cost also increases. For small graphs of Wikipedia and DBLP, we can see that TopK achieves several times better performance than TopKBasic on all k settings. For large graphs of Transaction20 and AddCart20, TopK also outperforms TopKBasic by several times, and an order of magnitude for top-20 biclique searching on AddCart20. Besides, from the figure, we can see that as k becomes larger, it takes longer time for both TopK and TopKBasic to find the results. This is because as k increases, the sizes of the result bicliques tend to be smaller. Consequently, the subspaces of later results in top-k are generated with relatively looser τ_U^i and τ_V^i constraints, which leads to larger reduced subgraphs and longer biclique searching time. In general, TopK outperforms TopKBasic by several times to an order of magnitude for all k settings.

Varying τ_U and τ_V In this experiment, we test the efficiency by varying the thresholds of τ_U and τ_V. The results are reported in Fig. 16. On Wikipedia and AddCart20, when τ_U and τ_V get larger, the time cost of both TopK and TopKBasic increases. The reason is that the average size of the top-k results becomes much smaller on Wikipedia and AddCart20 as τ_U and τ_V get larger. This leads to relatively looser τ_U^i and τ_V^i constraints and thus larger reduced subgraphs in subspaces, which takes longer time for biclique searching. On

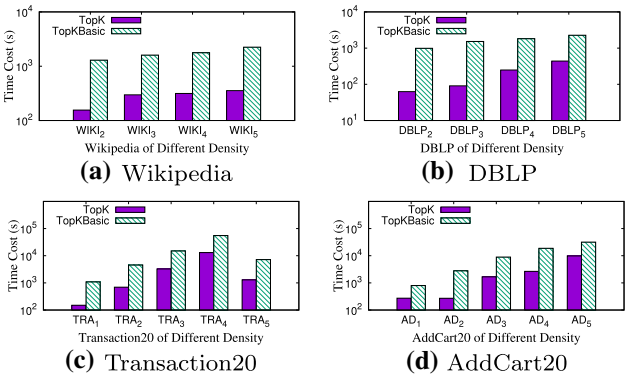


Fig. 17 Top-k search by varying graph density

DBLP and Transaction20, the sizes of the top-k results are relatively large on all τ_U and τ_V settings, and thus the performance is not sensitive to the τ_U and τ_V settings but to the specific generated τ_U^i and τ_V^i in subspaces. As TopKBasic needs to compute the k results one by one, while TopK can preserve more results in $\mathcal{FS}$ by slightly relax the τ_U^i and τ_V^i constraints in subspaces, the experimental results show that TopK benefits a lot from the computation sharing, and outperforms TopKBasic by several times to an order of magnitude on all τ_U and τ_V settings.

Varying graph density In this experiment, we show the effect of graph density on the performance and report the results in Fig. 17. We prepare graphs with different density by sampling edges in the original graphs, including small graphs of DBLP and Wikipedia, and large graphs of Transaction20 and AddCart20. For example, we sample 20%, 40%, 60%, 80% and 100% edges in Transaction20, denoted as TRA_1, TRA_2, TRA_3, TRA_4 and TRA_5, respectively. Note that we eliminate the results on WIKI_1 and DBLP_1, since we cannot obtain enough top-80 results on them. Figure 17 shows that as the graphs grow denser, TopKBasic takes longer time to find the top-k results in most cases, except that in TRA_5, the time cost decreases. The main reason is that the sizes of result bicliques in TRA_5 are relatively large, and consequently the subspaces are generated with more strict τ_U^i and τ_V^i constraints. The time cost of TopK has similar tendency with TopKBasic but increases slower as graphs grow denser (except TRA_5 where time cost decreases for the same reason), and TopK is from times to an order of magnitude faster than TopKBasic on all graphs. Therefore, TopK finds the diversified top-k bicliques efficiently on both sparse and dense graphs.

7.4 Case study

Our proposed algorithm has been deployed in Alibaba Group to detect fraudulent transactions. E-business owners at Taobao and Tmall (two E-commerce platforms of Alibaba Group) may pay some agents in black market to promote the rankings of their online shops. Considering the costs of fake transactions and maintenance of a large amount of user accounts, these agents usually need to organize a group of users to "purchase" a set of products at the same time for cost effectiveness. This will lead to some bicliques (i.e., click farms) in the bipartite graph consisting of users, products and purchase transactions. As the maximum biclique alone cannot cover all fraudulent transactions, we apply the diversified top-k biclique search method as follows.

TopK We adopt Algorithm 6 to compute the diversified top-k bicliques (i.e., suspicious click farms) in the bipartite graph. Note that TopK improves the recall rate of fraudulent transaction by 50% according to the feedback of the risk management team from Alibaba Group.

To further demonstrate the effectiveness and efficiency of TopK, we also evaluate the following two baseline approaches on a real dataset LabeledAddCart obtained from Alibaba Group, which includes the labels of ground-truth fraudulent transactions.

(1) EnumK We adopt EnumK, whose logic is the same with MBC but without the size pruning rule (in line 5 and 13 in Algorithm 1), to enumerate all maximal bicliques satisfying the thresholds τ_U and τ_V, and each maximal biclique represents a click farm. However, it is not possible to find all maximal bicliques and then select the top-k among them due to the huge number of maximal bicliques, thus we evaluate the result of the first-k output maximal bicliques.

(2) Reduce Given appropriate values of thresholds τ_U and τ_V, Reduce outputs the reduced bipartite graph, where the edges represent suspicious fraudulent transactions. Although Reduce cannot output bicliques, it can reduce the candidate size.

We define the precision and recall rate as follows:

$$\text{precision} = \frac{\text{number of found fraudulent transactions}}{\text{number of output edges of the method}}$$

$$\text{recall} = \frac{\text{number of found fraudulent transactions}}{\text{number of ground-truth fraudulent transactions}}$$

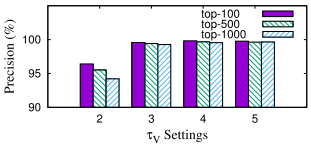


Fig. 18 Precision of TopK

TopK result evaluation In this experiment, we vary τ_V from 2 to 5 (with $\tau_U = 1$) to test the precision of top-k diversified bicliques found by TopK on LabeledAddCart, and show the results in Fig. 18. The figure shows that the precision is over 95% in most cases except top-1000 when $\tau_V = 2$. This is because coincidences are more likely to happen when τ_V is

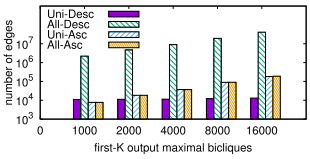


Fig. 19 Quality of EnumK

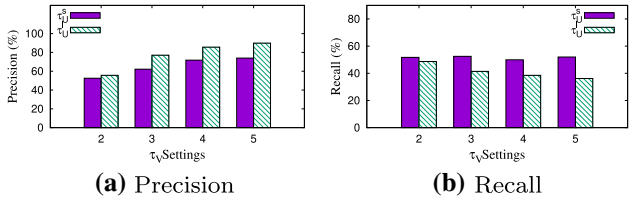


Fig. 20 Precision and recall rate of Reduce

small. When $\tau_V > 2$, the precision is even larger than 99%. In general, TopK outputs fraudulent transactions with high precision, and the found biclique can be served as the evidence when taking disciplinary measures. In real application in Alibaba Group, TopK not only returns fraudulent transactions with high precision, but also improves the recall rate by 50% w.r.t. to existing solutions.

EnumK result evaluation We conduct experiments of EnumK on LabeledAddCart and show the results in Fig. 19. We set $\tau_U = 1$ and $\tau_V = 2$, and the results with other settings are similar. Given the fact that EnumK cannot finish maximal biclique enumeration within 24 h, we record two statistics of the first-k output maximal bicliques: (1) the total number of output edges, denoted as *All*, and (2) the number of unique output edges, denoted as *Uni*. Besides, the enumeration process easily becomes stuck in local search, so the search order has great influence on the result of first-k bicliques. Thus, we adopt two search orders in EnumK, i.e., we iteratively add $v \in V$ into biclique in descending order (denoted as *Desc*) or ascending order (denoted as *Asc*) of the number of v's neighbors in U. This is because, intuitively, we may enumerate the maximal bicliques in the dense region or sparse region of the bipartite graph respectively. From Fig. 19, we can see that for *Desc* order, when the output biclique number increases, the total number of output edges increases as well. However, the number of unique edges barely grows, which indicates that EnumK enumerates many redundant maximal bicliques with very limited effective information when searching in dense region of the graph. In comparison, for *Asc* order, both total output edges and unique edges increases. However, the average size of the first-16000 maximal bicliques is only 12, which is too small to be used in anomaly detection application, with the precision of only 33.23% compared with the ground-truth. The computation cost of EnumK is also high, and the algorithm outputs huge amounts of maximal bicliques (over 10^9 bicliques in 24 h). In conclusion, maximal biclique enumeration is not suitable to this case study for anomaly detection on large-scale graphs.

Reduce result evaluation Given specific τ_U and τ_V values, we can detect fraudulent transactions with Reduce. In this experiment, we vary τ_V from 2 to 5, and for each τ_V^k, we set two corresponding τ_U^k values, i.e., the small value τ_U^{sk} for loose condition, and the large value τ_U^{lk} for strict condition. All τ_U values are suggested by the experts of anomaly detection in Alibaba. Due to the confidential nature, we omit the exact values. For simplicity, we use τ_U^s and τ_U^l to represent the loose and strict constraints for all τ_V^k.

We evaluate the performance in terms of precision and recall rate, and present the results in Fig. 20. In Fig. 20a, the precision of Reduce improves when τ_V grows larger, since the more common products a group bought together, the more suspicious the transactions are. Similarly, larger τ_U also leads to higher precision with fixed τ_V. However, the precision does not meet the requirement of at least 95% (from Alibaba). In Fig. 20b, the recall rate is relatively high especially for loose constraints τ_U^s, due to the fact that we only take advantages of the graph topological structure. However, we gain the high recall rate at the cost of low precision and large amount of output edges (over 10^7 edges for all settings). Besides, the result quality depends heavily on the given τ_U and τ_V thresholds, which cannot be easily adapted to different datasets manually. Therefore, Reduce is not suitable for anomaly detection in this case study.

8 Related work

In this section, we review the related work, including maximum biclique search and its variants, maximal biclique enumeration and diversified top-k search.

Maximum biclique search and its variants The maximum biclique problem has become increasingly popular in recent years [14,42,43]. Reference [43] proposes an integer programming methodology to find the maximum biclique in general graphs. However, it is not applicable for large-scale graphs. Reference [42] develops a Monte Carlo algorithm for extracting a list of maximal bicliques, which contains a maximum biclique with fixed probability. Reference [14] studies the parameterized maximum biclique problem in bipartite graphs that reports if there exists a biclique with at least p edges, where p is a given integer parameter. Besides, there are two variants of the maximum biclique problem, i.e., the maximum vertex biclique and the maximum balanced biclique. The former one aims to find the biclique C^* that $|U(C^*)| + |V(C^*)|$ is maximized. This problem can be solved in polynomial time by a minimum cut algorithm [33]. The latter one aims to find the biclique C^* with maxi-

mum cardinality that $|U(C^*)| = |V(C^*)|$. The most popular approaches are heuristic algorithms, including [2,44,55] that solve the problem by converting it into a maximum balanced independent set problem on the complement bipartite graph with node deletion strategies, and [60] that combines tabu search and graph reduction to find the maximum balanced biclique on the original bipartite graph. References [53,56] propose local search framework to find good solutions within reasonable time. Refs. [32,61] introduce exact algorithms to find the maximum balanced biclique by following the branch-and-bound framework.

Maximal biclique enumeration The maximal biclique enumeration problem is widely studied. A biclique is said to be maximal if it is not contained in any larger bicliques. Reference [3] proposes a consensus approach, which starts with a collection of simple bicliques, and then expands the bicliques as a sequence of transformations on the biclique collections. References [36,41] find maximal bicliques $C = (U, V, U \times V)$ by exhaustively enumerating U as subsets of one vertex partition, obtaining V as their common neighbors in the other vertex partition, and then checking the maximality of C. In [59], the authors propose algorithm iMBEA, which combines backtracking with branch-and-bound framework to filter out the branches that cannot lead to maximal bicliques. Reference [15,29] reduce the problem to the maximal clique enumeration problem by transferring the bipartite graph into a general graph. Reference [21] proves that maximal biclique is in correspondence with frequent closed itemset. The maximal biclique enumeration can be reduced then to the well-studied frequent closed itemsets mining problem [13,27,50,52]. References [35,37] propose parallel methods to enumerate maximal bicliques in large graphs.

Diversified top-k search The diversified top-k search problem has been extensively studied, which aims to find top-k results that are not only most relevant to a query but also diversified. In the literature, most existing solutions focus on finding diversified top-k results for a specific query. For example, Lin et al. study the k most representative skyline problem [22]. References [1,6] focus on the diversified top-k document retrieval. Reference [62] studies the diversified keyword query recommendation. Reference [12] focuses on the diversified top-k graph pattern matching. Reference [24] studies the problem of top-k shortest paths with diversity. Zhang et al. study the diversified top-l (k, r)-core [58]. Yuan et al. and Wu et al. study the diversified top-k clique search problem [54,57]. Nevertheless, the techniques developed for diversified top-k clique search are not suitable for our diversified top-k biclique search problem. Some other works study the general framework for diversified top-k search. For example, [10,39,40,51] study the general diversified top-k results problem. References [8,34] study top-k result diversification on a dynamic environment. The complexity of query result diversification is analyzed in [9]. Nevertheless, the diversity in the above frameworks is considered based on the pair-wise dissimilarity of the query results, which cannot be applied directly on the diversified top-k biclique search problem studied in this paper.

9 Conclusion

Maximum biclique search in a bipartite graph is a fundamental problem with a wide spectrum of applications. Existing solutions are not scalable for handling large bipartite graphs because the search has to consider the size of both sides of the biclique. In this paper, instead of solving the problem directly on the original bipartite graph, we propose a progressive bounding framework which aims to solve the problem on several much smaller bipartite graphs. We prove that only logarithmic rounds are needed to guarantee the algorithm correctness, and in each round, we show how to significantly reduce the bipartite graph size by considering the properties of the one-hop and two-hop neighbors for each vertex. Based on the maximum biclique search method, we further propose an efficient algorithm to find the diversified top-k bicliques, which is also desirable in many applications. By taking advantage of the progressive bounding framework, we consider to derive the same subspaces for different results by slightly relaxing the constraints in each subspace, so as to share the computation cost among these results. We further propose two optimizations to accelerate the computation by pruning search space and lazy refining candidates. We conducted experiments on real datasets from different application domains, and two of the datasets contain billions of edges. The experimental results demonstrate that our approach is efficient and scalable to handle large bipartite graphs. It is reported that 50% improvement on recall can be achieved after applying our method in Alibaba Group to identify the fraudulent transactions.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

References

1. Agrawal, R., Gollapudi, S., Halverson, A., Ieong, S.: Diversifying search results. In: Baeza-Yates, R., Boldi, P., Ribeiro-Neto, B.A., Cambazoglu, B.B. (eds.) Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9–11, 2009, pp. 5–14. ACM (2009)
2. Al-Yamani, A.A., Ramsundar, S., Pradhan, D.K.: A defect tolerance scheme for nanotechnology circuits. IEEE Trans. Circuits Syst. **54**(11), 2402–2409 (2007)
3. Alexe, G., Alexe, S., Crama, Y., Foldes, S., Hammer, P.L., Simeone, B.: Consensus algorithms for the generation of all maximal bicliques. Discrete Appl. Math. **145**(1), 11–21 (2004)
4. Allahbakhsh, M., Ignjatovic, A., Benatallah, B., Bertino, E., Foo, N., et al.: Collusion detection in online rating systems. In: Asia-Pacific Web Conference, pp. 196–207. Springer (2013)
5. Ambühl, C., Mastrolilli, M., Svensson, O.: Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. SIAM J. Comput. **40**(2), 567–596 (2011)
6. Angel, A., Koudas, N.: Efficient diversity-aware search. In: Sellis, T.K., Miller, R.J., Kementsietsidis, A., Velegrakis, Y. (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011, pp. 781–792. ACM (2011)
7. Beutel, A., Xu, W., Guruswami, V., Palow, C., Faloutsos, C.: Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In: 22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13–17, 2013, pp. 119–130 (2013)
8. Borodin, A., Jain, A., Lee, H.C., Ye, Y.: Max-sum diversification, monotone submodular functions, and dynamic updates. ACM Trans. Algorithms **13**(3), 41:1-41:25 (2017)
9. Deng, T., Fan, W.: On the complexity of query result diversification. Proc. VLDB Endow. **6**(8), 577–588 (2013)
10. Drosou, M., Pitoura, E.: Disc diversity: result diversification based on dissimilarity and coverage. Proc. VLDB Endow. **6**(1), 13–24 (2012)
11. Eppstein, D.: Arboricity and bipartite subgraph listing algorithms. Inf. Process. Lett. **51**(4), 207–211 (1994)
12. Fan, W., Wang, X., Wu, Y.: Diversified top-k graph pattern matching. Proc. VLDB Endow. **6**(13), 1510–1521 (2013)
13. Fang, G., Wu, Y., Li, M., Chen, J.: An efficient algorithm for mining frequent closed itemsets. Informatica (Slovenia) **39**(1), 87–98 (2015)
14. Feng, Q., Li, S., Zhou, Z., Wang, J.: Parameterized algorithms for edge biclique and related problems. Theor. Comput. Sci. **734**, 105–118 (2017)
15. Gely, A., Nourine, L., Sadi, B.: Enumeration aspects of maximal cliques and bicliques. Discrete Appl. Math. **157**(7), 1447–1459 (2009)
16. Kershenbaum, A., Cutillo, A., Darabos, C., Murray, K., Schiaffino, R., Moore, J.H.: Bicliques in graphs with correlated edges: From artificial to biological networks. In: European Conference on the Applications of Evolutionary Computation, pp. 138–155. Springer (2016)
17. Konc, J., Janezic, D.: An improved branch and bound algorithm for the maximum clique problem. Commun. Math. Comput. Chem. **58**, 569–590 (2007)
18. Langston, M.A., Chesler, E.J., Zhang, Y.: 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, p. 473 (2008)
19. Li, C.-M., Fang, Z., Jiang, H., Xu, K.: Incremental upper bound for the maximum clique problem. INFORMS J. Comput. **30**(1), 137–153 (2017)
20. Li, C.M., Quan, Z.: An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. AAAI **10**, 128–133 (2010)
21. Li, J., Li, H., Soh, D., Wong, L.: A correspondence between maximal complete bipartite subgraphs and closed patterns. In: European Conference on Principles of Data Mining and Knowledge Discovery, pp. 146–156. Springer (2005)
22. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: the k most representative skyline operator. In: Chirkova, R., Dogac, A, Özsu, M.T., Sellis, T.K. (eds.) Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15–20, 2007, pp. 86–95. IEEE Computer Society (2007)
23. Liu, G., Sim, K., Li, J.: Efficient mining of large maximal bicliques. In: International Conference on Data Warehousing and Knowledge Discovery, pp. 437–448. Springer (2006)
24. Liu, H., Jin, C., Yang, B., Zhou, A.: Finding top-k shortest paths with diversity. In: 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16–19, 2018, pp. 1761–1762. IEEE Computer Society (2018)
25. Liu, J., Wang, W.: Op-cluster: clustering by tendency in high dimensional space. In: Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19–22 December 2003, Melbourne, Florida, USA, pp. 187–194 (2003)
26. Lu, C., Yu, J.X., Wei, H., Zhang, Y.: Finding the maximum clique in massive graphs. PVLDB **10**(11), 1538–1549 (2017)
27. Lucchese, C., Orlando, S., Perego, R.: Fast and memory efficient mining of frequent closed itemsets. IEEE Trans. Knowl. Data Eng. **18**(1), 21–36 (2006)
28. Lyu, B., Qin, L., Lin, X., Zhang, Y., Qian, Z., Zhou, J.: Maximum biclique search at billion scale. Proc. VLDB Endow. **13**(9), 1359–1372 (2020)
29. Makino, K., Uno, T.: New algorithms for enumerating all maximal cliques. In: Scandinavian Workshop on Algorithm Theory, pp. 260–272. Springer (2004)
30. Manurangsi, P.: Inapproximability of maximum biclique problems, minimum k-cut and densest at-least-k-subgraph from the small set expansion hypothesis. Algorithms **11**(1), 10 (2018)
31. Maslov, E., Batsyn, M., Pardalos, P.M.: Speeding up branch and bound algorithms for solving the maximum clique problem. J. Global Optim. **59**(1), 1–21 (2014)
32. McCreesh, C., Prosser, P.: An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem. In: International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, pp. 226–234. Springer (2014)
33. Michael, R.G., David, S.J.: Computers and Intractability: A Guide to the Theory of np-Completeness, pp. 90–91. WH Free. Co., San Francisco (1979)
34. Minack, E., Siberski, W., Nejdl, W.: Incremental diversification for very large sets: a streaming-based approach. In: Ma, W., Nie, J., Baeza-Yates, R., Chua, T., Croft, W.B. (eds.) Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25–29, 2011, pp. 585–594. ACM (2011)
35. Mukherjee, A.P., Tirthapura, S.: Enumerating maximal bicliques from a large graph using mapreduce. IEEE Trans. Serv. Comput. **10**(5), 771–784 (2017)
36. Mushlin, R.A., Kershenbaum, A., Gallagher, S.T., Rebbeck, T.R.: A graph-theoretical approach for pattern discovery in epidemiological research. IBM Syst. J. **46**(1), 135–149 (2007)

37. Nataraj, R., Selvan, S.: Parallel mining of large maximal bicliques using order preserving generators. Int. J. Comput. **8**(3), 105–113 (2014)
38. Peeters, R.: The maximum edge biclique problem is np-complete. Discrete Appl. Math. **131**(3), 651–654 (2003)
39. Qin, L., Yu, J.X., Chang, L.: Diversifying top-k results. Proc. VLDB Endow. **5**(11), 1124–1135 (2012)
40. Ranu, S., Hoang, M.X., Singh, A.K.: Answering top-k representative queries on graph databases. In: Dyreson, C.E., Li, F., Özsu, M.T. (eds.) International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22–27, 2014, pp. 1163–1174. ACM (2014)
41. Sanderson, M.J., Driskell, A.C., Ree, R.H., Eulenstein, O., Langley, S.: Obtaining maximal concatenated phylogenetic data sets from large sequence databases. Mol. Biol. Evolut. **20**(7), 1036–1042 (2003)
42. Shaham, E., Yu, H., Li, X.: 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, Miami, Florida, USA, May 5–7, 2016, pp. 315–323 (2016)
43. Shahinpour, S., Shirvani, S., Ertem, Z., Butenko, S.: Scale reduction techniques for computing maximum induced bicliques. Algorithms **10**(4), 113 (2017)
44. Tahoori, M.B.: Application-independent defect tolerance of reconfigurable nanoarchitectures. ACM J. Emerg. Technol. Comput. Syst. **2**(3), 197–218 (2006)
45. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically significant biclusters in gene expression data. In: Proceedings of the Tenth International Conference on Intelligent Systems for Molecular Biology, August 3–7, 2002, Edmonton, Alberta, Canada, pp. 136–144 (2002)
46. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Global Optim. **37**(1), 95–111 (2007)
47. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Discrete Mathematics and Theoretical Computer Science, pp. 278–289. Springer (2003)
48. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: International Workshop on Algorithms and Computation, pp. 191–203. Springer (2010)
49. Tomita, E., Yoshida, K., Hatta, T., Nagao, A., Ito, H., Wakatsuki, M.: A much faster branch-and-bound algorithm for finding a maximum clique. In: International Workshop on Frontiers in Algorithmics, pp. 215–226. Springer (2016)
50. Tong, Y., Chen, L., Ding, B.: Discovering threshold-based frequent closed itemsets over probabilistic data. In: IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1–5 April, 2012, pp. 270–281 (2012)
51. Vieira, M.R., Razente, H.L., Barioni, M.C.N., Hadjieleftheriou, M., Srivastava, D., Trania, C., Tsotras, V.J.: On query result diversification. In: Abiteboul, S., Böhm, K., Koch, C., Tan, K. (eds.) Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11–16, 2011, Hannover, Germany, pp. 1163–1174. IEEE Computer Society (2011)
52. Wang, J., Han, J., Pei, J.: Closet+: searching for the best strategies for mining frequent closed itemsets. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 236–245. ACM (2003)
53. Wang, Y., Cai, S., Yin, M.: New heuristic approaches for maximum balanced biclique problem. Inf. Sci. **432**, 362–375 (2018)
54. Wu, J., Li, C., Jiang, L., Zhou, J., Yin, M.: Local search for diversified top-k clique search problem. Comput. Oper. Res. **116**, 104867 (2020)
55. Yuan, B., Li, B.: A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar. JETC **10**(3), 25:1-25:19 (2014)
56. Yuan, B., Li, B., Chen, H., Yao, X.: A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem. IEEE Trans. Cybern. **45**(5), 1040–1053 (2015)
57. Yuan, L., Qin, L., Lin, X., Chang, L., Zhang, W.: Diversified top-k clique search. VLDB J. **25**(2), 171–196 (2016)
58. Zhang, F., Lin, X., Zhang, Y., Qin, L., Zhang, W.: Efficient community discovery with user engagement and similarity. VLDB J. **28**(6), 987–1012 (2019)
59. Zhang, Y., Phillips, C.A., Rogers, G.L., Baker, E.J., Chesler, E.J., Langston, M.A.: On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. BMC Bioinform. **15**, 110 (2014)
60. Zhou, Y., Hao, J.-K.: Combining tabu search and graph reduction to solve the maximum balanced biclique problem. arXiv preprint arXiv:1705.07339 (2017)
61. Zhou, Y., Rossi, A., Hao, J.-K.: Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs. Eur. J. Oper. Res. **269**(3), 834–843 (2018)
62. Zhu, X., Guo, J., Cheng, X., Du, P., Shen, H.: A unified framework for recommending diverse and relevant queries. In: Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M.P., Bertino, E., Kumar, R. (eds.) Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28–April 1, 2011, pp. 37–46. ACM (2011)