Maximum Biclique Search at Billion Scale

Bingqing Lyu
Alibaba Group
bingqing.lbq@alibaba-inc.com

Lu Qin
The University of Technology Sydney
lu.qin@uts.edu.au

Xuemin Lin
The University of New South Wales
lxue@cse.unsw.edu.au

Ying Zhang
The University of Technology Sydney
ying.zhang@uts.edu.au

Zhengping Qian
Alibaba Group
zhengping.qzp@alibaba-inc.com

Jingren Zhou
Alibaba Group
jingren.zhou@alibaba-inc.com

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. 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.

PVLDB Reference Format:
Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. Maximum Biclique Search at Billion Scale. *PVLDB*, 13(9): 1359-1372, 2020.
DOI: https://doi.org/10.14778/3397230.3397234

1. INTRODUCTION

Proceedings of the VLDB Endowment, Vol. 13, No. 9
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3397230.3397234

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 clique in general graph, biclique is a fundamental structure in 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 [3, 5]. 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 modelled as bicliques in a bipartite graph. Similarly, in web services, biclique 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 [32, 16, 11, 43, 9]. 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 [15]. 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 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., find 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 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 hours running of maximal biclique enumeration algorithm on a e-commerce bipartite graph obtained from Alibaba). Thus, we have to consider the top k diversified/representative bicliques. Inspired by the well-studied top k diversified clique problem (e.g., [42]), 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 [27]. In the literature, there are many solutions to solve another related NP-hard problem: the maximum clique search problem in a general graph [34, 10, 33, 35, 36, 13, 21, 12, 17]. 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 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 [27, 43] 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.

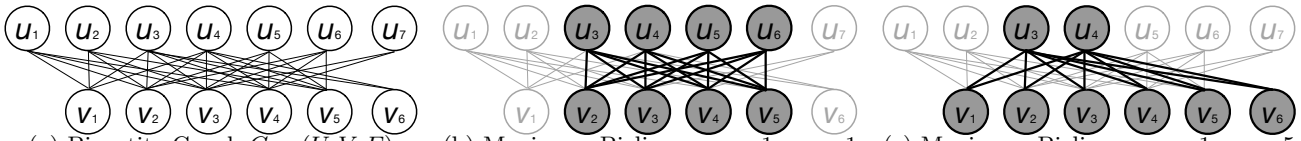
Our Solution. Based on the above discussion, existing coloring and core decomposition based approaches cannot yield effective pruning in maximum biclique search. Our paper aims for a new way to solve the problem. Our main idea is as follows: instead of finding the upper bounds for pruning, we try to 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 (τ_U^0, τ_V^0), (τ_U^1, τ_V^1), ... 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.

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.
- *Extensive Performance Studies on Billion-Scale Bipartite Graphs.* We conduct extensive performance studies using 16 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 minutes. 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 problem and shows its hardness. Section 3 intro-



(a) Bipartite Graph $G = (U, V, E)$ (b) Maximum Biclique, $\tau_U = 1, \tau_V = 1$ (c) Maximum Biclique, $\tau_U = 1, \tau_V = 5$

Figure 1: An Example of a Bipartite Graph and its Maximum Biclique

duces the baseline solution based on the branch-and-bound framework. In Section 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 Section 6, we evaluate our proposed algorithms using extensive experiments. We review the related work in Section 7 and conclude the paper in Section 8.

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 2.1: (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 2.1: Fig. 1 (a) 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. 1 (b), 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. 1 (c), where $U(C_2) = \{u_3, u_4\}$ and $V(C_2) = \{v_1, v_2, ..., v_6\}$. □

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* ← C;
2  BranchBound(U(G), ∅, V(G), ∅);
3  return C*;
4  Procedure BranchBound(U, V, C_V, X_V)
5  if |V| ≥ τ_V and |U| × |V| > |C*| then
6      C* ← (U, V, U × V);
7  while C_V ≠ ∅ do
8      v* ← C_V.pop();
9      U' ← {u ∈ U|(u, v*) ∈ E(G)};
10     V' ← V ∪ {v*} ∪ {v ∈ C_V|U' ⊆ N(v, G)};
11     C'_V ← {v ∈ C_V \ V' | |N(v, G) ∩ U'| ≥ τ_U};
12     X'_V ← {v ∈ X_V | |N(v, G) ∩ U'| ≥ τ_U};
13     if |U'| ≥ τ_U and |V'| + |C'_V| ≥ τ_V and |U'| × (|V'| +
       |C'_V|) > |C*| and ∄ v ∈ X_V s.t. U' ⊆ N(v, G) then
14         BranchBound(U', V', C'_V, X'_V);
15     X_V ← X_V ∪ {v*};
```

NP-hardness and Inapproximability. As shown in [27], the maximum biclique problem is NP-hard, and as proved in [4] and [20], 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 [43] 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 [43], 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.,

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

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 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

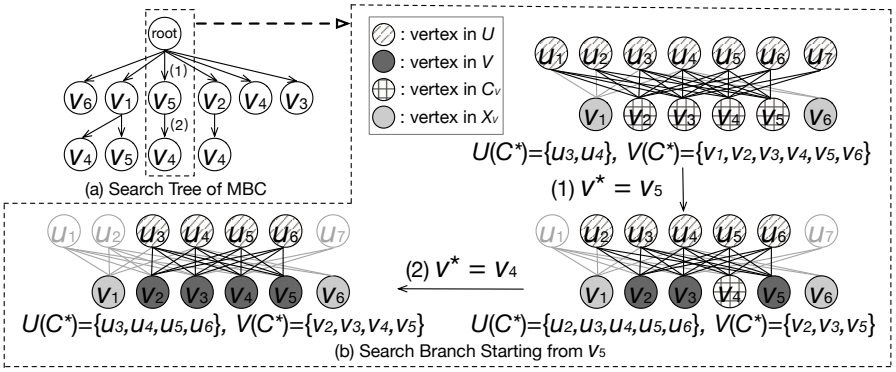


Figure 2: An Example of MBC Searching

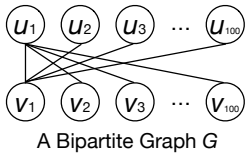


Figure 3: Drawbacks of MBC

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). 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 3.1: Given the bipartite graph G in Fig. 1(a) and thresholds $\tau_U = 1$ and $\tau_V = 1$, we show the search tree of MBC in Fig. 2(a). The vertices in V are processed in non-descending order of degree [43], and each tree node represents v^* selected in the branch. We illustrate the details in search branch from v_5 in Fig. 2(b). 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 4.1: Fig. 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 neithor 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_U^1, τ_V^1), (τ_U^2, τ_V^2), ..., (τ_U^k, τ_V^k). We will guarantee that for any possible biclique C with $U(C) \times V(C) \geq lb(C^*)$, there exists a pair (τ_U^i, τ_V^i) for $1 \leq i \leq k$ s.t. $\tau_U^i \leq |U(C)|$ and $\tau_V^i \leq |V(C)|$. Then, for each (τ_U^i, τ_V^i) for $1 \leq i \leq k$, we compute a biclique C_i^* with maximum size s.t. $|U(C_i^*)| \geq \tau_U^i$ and $|V(C_i^*)| \geq \tau_V^i$. Among the computed bicliques, the biclique with the maximum size is the answer for the original problem.

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).

Algorithm 2: MBC$^*(G, \tau_U, \tau_V)$

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 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 4.1: (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 4.2: (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 4.1: ***(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.

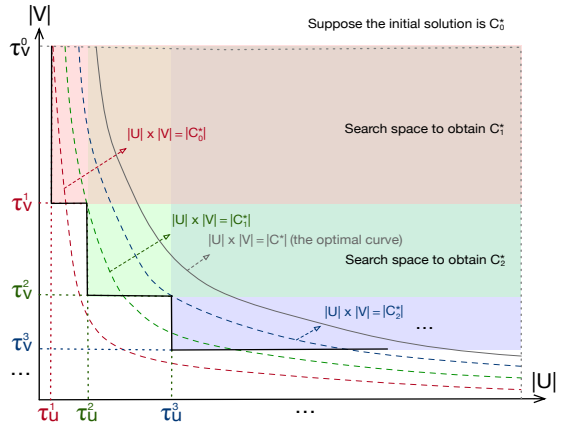


Figure 4: Illustration of Algorithm Rationale

- *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 4.1 holds. □

Theorem 4.1 shows that all 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 biclique C in G, we can guarantee that $|U(C)| \leq d_{max}^V(G)$ and $|V(C)| \leq d_{max}^U(G)$. Therefore, Algorithm 2 obtains the optimal solution.

The rationale of the progressive bound framework is illustrated 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_U^1, τ_V^1), (τ_U^2, τ_V^2), and (τ_U^3, τ_V^3), 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 $|V| \geq \tau_V^3$ is totally covered by the three search spaces. This illustrates the correctness of the progressive bounding framework.

Example 4.2: Given the bipartite graph G in Fig. 1(a) 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. 1(c), that we have $|C_0^*| = 12$ and $\tau_V^0 = 6$. Then we search the optimal solution progressively:
(1) $\tau_U^1 = 2$, $\tau_V^1 = 3$. 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 $\tau_V^1 = 3$. 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) $\tau_U^2 = 5$, $\tau_V^2 = 1$. Since we cannot find any larger biclique on reduced graph G_2, $|C_2^*| = 16$.
As shown above, we progressivly use multiple strict τ_U^k and τ_V^k 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_U^i and τ_V^i 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 5.1: (MBC-Preserved Graph) Given a bipartite graph G, and thresholds τ_U^i and τ_V^i, a bipartite graph G' is called a MBC-preserved graph w.r.t. τ_U^i and τ_V^i, if $U(G') \subseteq U(G)$, $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$ and $|C^*_{\tau_U^i, \tau_V^i}(G')| = |C^*_{\tau_U^i, \tau_V^i}(G)|$. In other words, the maximum biclique for G is preserved in G'. We use $G' \sqsubseteq_{\tau_U^i, \tau_V^i} G$ to denote that G' is an MBC-preserved graph of G. □

We can easily derive the following lemma:

Lemma 5.1: ***(Transitive Property)*** *If* $G_1 \sqsubseteq_{\tau_U^i, \tau_V^i} G_2$ *and* $G_2 \sqsubseteq_{\tau_U^i, \tau_V^i} G_3$, *we have* $G_1 \sqsubseteq_{\tau_U^i, \tau_V^i} G_3$. □

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 5.2: *Given a bipartite graph* G, *thresholds* τ_U^i *and* τ_V^i, *we have:*
(1) $\forall u \in U(G)$: $d(u, G) < \tau_V^i \implies G \ominus u \sqsubseteq_{\tau_U^i, \tau_V^i} G$;
(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 5.2 provides a sufficient condition for a vertex to be eliminated s.t. the maximum biclique is preserved. Based

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;
```

on the Lemma 5.1, Lemma 5.2 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 5.2. 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 5.2. The algorithm terminates until no such vertices can be found in G_i. The following lemma shows the time complexity of Algorithm 3.

Lemma 5.3: *Algorithm 3 requires $O(|G|)$ time.* □

The proof of Lemma 5.3 is omitted due to space limit.

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 5.2: (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. 5(b), 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 5.4: *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) $\forall v \in V(G) : d_{\tau_U^i}(v, G) < \tau_V^i \implies G \ominus v \sqsubseteq_{\tau_U^i,\tau_V^i} G$. □

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 ← G;
2  G_i ← Reduce2H(G_i, U(G_i), τ_U^i, τ_V^i);
3  G_i ← Reduce2H(G_i, V(G_i), τ_V^i, τ_U^i);
4  return G_i;
5  Procedure Reduce2H(G_i, U, τ_U^i, τ_V^i)
6  for each u ∈ U do
7      S ← ∅;
8      for each v ∈ N(u, G_i) do
9          for each u' ∈ N(v, G_i) do
10             if S.find(u') = ∅ then
11                 S ← S ∪ {(u', 1)};
12             else
13                 o ← S.find(u');
14                 o.cnt ← o.cnt + 1;
15     c ← |{o ∈ S | o.cnt ≥ τ_V^i}|;
16     if c < τ_U^i then
17         G_i ← G_i ⊖ u;
18 return G_i;
```

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:

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

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 5.4 and the transitive property shown in Lemma 5.1, we are ready to design the two-hop graph reduction algorithm. The pseudocode of the algorithm is shown in Algorithm 4. Since Lemma 5.4 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 5.4 (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

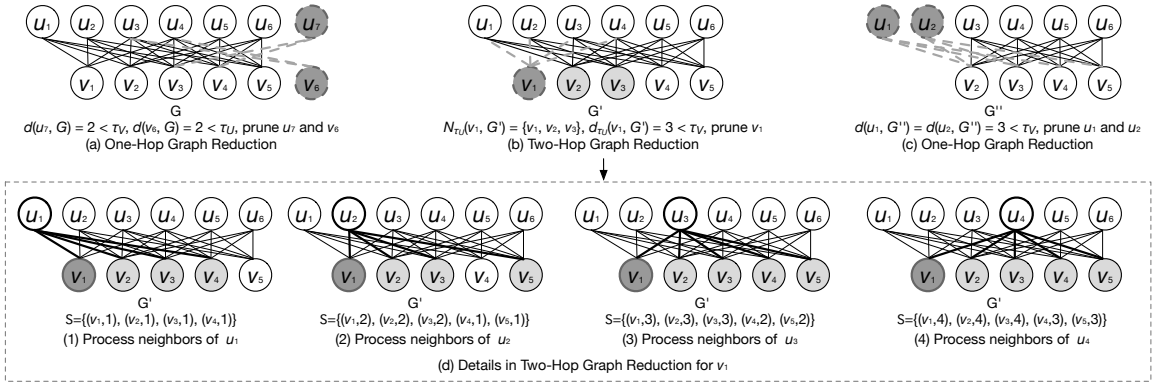


Figure 5: An Example of Graph Reduction with $\tau_U = 4$, $\tau_V = 4$

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 5.4 (line 16-17).

Lemma 5.5: *Algorithm 4 requires* $O(\sum_{u \in U(G)} d(u,G)^2 + \sum_{v \in V(G)} d(v,G)^2)$ *time.* □

The proof of Lemma 5.5 is omitted due to space limit.

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.

(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 5.6: *For any vertices u, u' and threshold τ, we have:*

$$u' \in N_\tau(u, G) \iff u \in N_\tau(u', G) \qquad \square$$

Based on Lemma 5.6, 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 5.4 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 5.4 has stronger pruning power than the one-hop reduction rule in Lemma 5.2.

Lemma 5.7: *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 Lemma 5.3 and Lemma 5.5, 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 5.1: We show the example of the complete graph reduction process in Fig. 5. Given the bipartite graph G in Fig. 1(a) and thresholds $\tau_U = 4$, $\tau_V = 4$ and MAX_ITER $= 2$, we first apply Reduce1Hop in Fig. 5(a). 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. 5(b) with the details shown in Fig. 5(d). 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 3 vertices with $cnt = 4$, i.e., $c = d_{\tau_U}(v_1, G') = 3$. According to Lemma 5.4, since $d_{\tau_U}(v_1, G') < \tau_V$, we prune v_1. After that, we further apply Reduce1Hop in Fig. 5(c), and prune vertices u_1 and u_2. By applying Reduce, we save huge search space in biclique search. □

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
AddCart	MISC	141,839,807	Customer	65,589,796	Product	1,307,950,593	AddCart
Transaction	MISC	272,227,190	Customer	75,350,951	Product	1,319,706,942	Purchasing

6. PERFORMANCE STUDIES

Below, we present our experimental results by comparing the proposed maximum biclique search algorithm MBC*, with the baseline algorithm MBC. MBC is developed based on the algorithm in [43], where the code is obtained from the authors, with the pruning rules in Algorithm 1 added. 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. A case study of anomaly detection on real datasets obtained from Alibaba Group is described to demonstrate the resultant quality by applying our method. Unless otherwise specified, experiments are conducted with $\tau_U = 3$ and $\tau_V = 3$ 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 16 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 3 datasets are real datasets obtained from the E-Commerce company Alibaba Group. Here, the AddCart dataset includes data of customers adding products into cart in 10 days, and the Transaction dataset includes data of customers purchasing products in 15 days. Additionally, the LabeledAddCart dataset includes fraudulent transactions labels that we utilize as the ground truth in the case study.

6.1 Graph Reduction and Optimizations

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

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 Table 2 and Table 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.

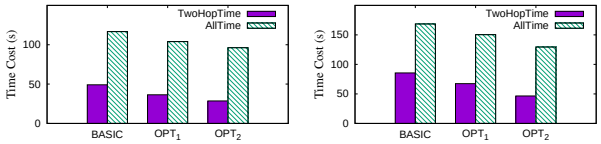
[1]http://konect.uni-koblenz.de/

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



(a) LiveJournal (b) WebTrackers

Figure 6: Optimization Strategies

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 Section 5.2 as OPT_1, and the algorithm with early skipping strategy introduced in Section 5.2 as OPT_2 based on OPT_1. The results are shown in Fig. 6, 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.

6.2 MBC* vs MBC

In this subsection, we compare the performance of MBC* and MBC 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 hours, and if the methods cannot finish computing, we denote the time cost as NaN. For those experiments that cannot finish within 24 hours, 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.

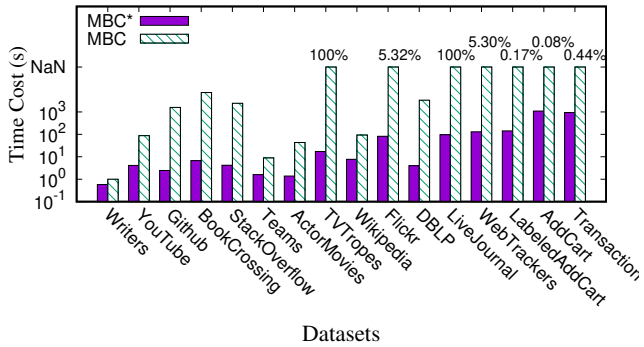
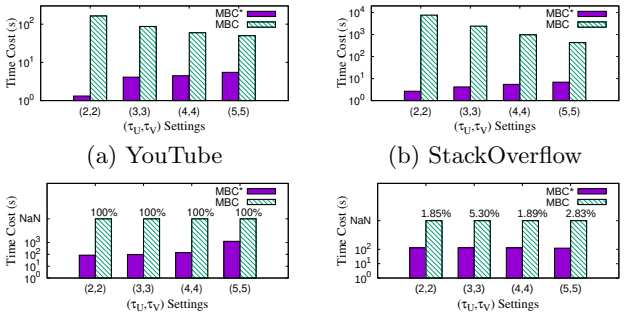


Figure 7: $C^*_{3,3}$ Search on All Datasets

All Datasets. In this experiment, we test the efficiency of $C^*_{3,3}$ search in all datasets by comparing MBC* with MBC, and report the processing time in Fig. 7. From Fig. 7, 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 hours, while MBC* only takes minutes to compute the results. For most cases where MBC cannot finish computing, the bicliques it finds within 24 hours are far smaller than the maximum bicliques. From the results shown in Fig. 7, we can see that MBC* is much more efficient and scalable than MBC on all datasets.

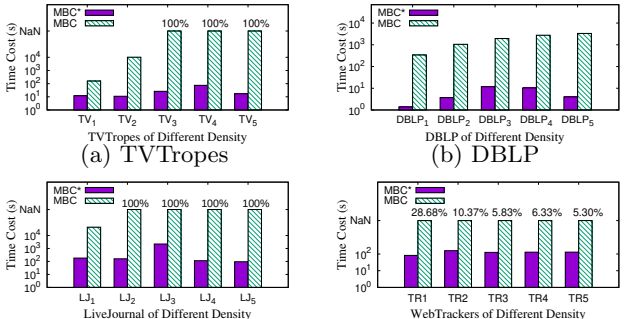
Varying τ_U and τ_V thresholds. We vary τ_U and τ_V thresholds to compute C^* and illustrate the performance of MBC* and MBC in Fig. 8. Fig. 8 shows that MBC can process small graphs (YouTube and StackOverflow) but fails in processing large graphs (LiveJournal and WebTrackers). For small graphs, when 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 hours, since the search space is huge and MBC is stuck in local search. In comparison, MBC* is orders of magnitude faster than MBC 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.



(a) YouTube (b) StackOverflow

(c) LiveJournal (d) WebTrackers

Figure 8: C^* Search by Varying τ_U and τ_V



(a) TVTropes (b) DBLP

(c) LiveJournal (d) WebTrackers

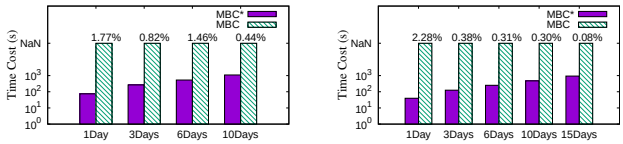
Figure 9: $C^*_{3,3}$ Search by Varying Graph Density

Varying Graph Density. In this experiment, we test the effect of graph density on the performance, and demonstrate the results in Fig. 9. 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. Fig. 9 shows that as the graphs grow denser, MBC takes longer time to find the maximum bicliques, or cannot finish computing within 24 hours. In contrast, MBC* is orders of magnitude faster than MBC on all settings. It is worth noting that for dense graphs, MBC* also finds maximum bicliques efficiently. For example, in Fig. 9(c), 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.

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



(a) AddCart (b) Transaction

Figure 10: $C^*_{3,3}$ Search by Varying Graph Scale

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 AddCart, and 1, 3, 6, 10 and 15 days data of Transaction. We list the statistics in Table 4, and report the results in Fig. 10. In Fig. 10, we can see that MBC cannot finish computing within 24 hours 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 graph of AddCart10d and Transaction15d, which both consist of about 1.3 billion edges, MBC* costs 18 minutes and 15 minutes 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.

6.3 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 use the top-K diversified biclique method as follows.

topKMax. We repeatedly compute the maximum biclique using MBC* and then delete it from the bipartite graph, and further apply MBC* on the remaining graph, until we obtain K bicliques (i.e., suspicious click farms). We call this approach topKMax. Note that topKMax 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 topKMax, 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) enumKMax. We adopt enumKMax, 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:

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

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

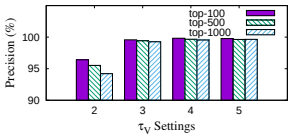


Figure 11: Precision of topKMax

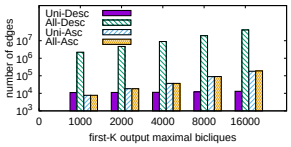
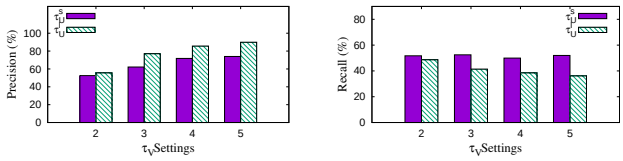


Figure 12: Quality of enumKMax

topKMax 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 topKMax on LabeledAddCart, and show the results in Fig. 11. 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 small. When $\tau_V > 2$, the precision is even larger than 99%. In general, topKMax 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, topKMax not only returns fraudulent transactions with high precision, but also improves the recall rate by 50% w.r.t. to exisiting solutions.

enumKMax Result Evaluation. We conduct experiments of enumKMax on LabeledAddCart and show the results in Fig. 12. We set $\tau_U = 1$ and $\tau_V = 2$, and the results with other settings are similar. Given the fact that enumKMax cannot finish maximal biclique enumeratation within 24 hours, 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 enumKMax, 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. 12, 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 enumKMax enumerates many redundant maximal bicliques with very limited effective information when searching in dense region of the graph. In comparision, 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 enumKMax is also high, and the algorithm outputs huge amounts of maximal bicliques (over 10^9 bicliques in 24 hours). In conclusion, maximal biclique enumeration is not suitable to this case study for anomaly detection on large-scale graphs.



(a) Precision (b) Recall

Figure 13: Precision and Recall Rate of Reduce

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. 13. In Fig. 13(a), the precision of Reduce improves when τ_V grows larger, since the more common products a group bought together, the more suspecious 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. 13(b), 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.

7. RELATED WORK

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

Maximum Biclique Search and its Variants. The maximum biclique problem has become increasingly popular in recent years [30, 29, 7]. [30] proposes an integer programming methodology to find the maximum biclique in general graphs. However, it is not applicable for large scale graphs. [29] develops a Monte Carlo algorithm for extracting a list of maximal bicliques, which contains a maximum biclique with fixed probability. [7] studies the parameterized maximum biclique problem in bipartite graphs, that reports if there exists a biclique with at least k edges, where k 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 [23]. The latter one aims to find the biclique C^* with maximum cardinality that $|U(C^*)| = |V(C^*)|$. The most popular apporoach is heuristic algorithms, including [1, 40, 31] that solve the problem by converting it into a maximum balanced independent set problem on the complement bipartite graph with node deletion strategies, and [44] that combines tabu search and graph reduction to find the maximum balanced biclique on the original bipartite graph. [41, 39] propose local search framework to find good solutions within reasonable time. [22, 45] 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. [2] 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. [28, 25] 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 [43], the authors propose algorithm iMBEA, which combines backtracking with branch-and-bound framework to filter out the branches that cannot lead to maximal bicliques. [19, 8] reduce the problem to the maximal clique enumeration problem by transferring the bipartite graph into a general graph. [14] 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 [6, 37, 38, 18]. [26, 24] propose parallel methods to enumerate maximal bicliques in large graphs.

8. 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. 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.

Acknowledgments. Lu Qin is supported by ARC DP1601-01513. Ying Zhang is supported by ARC DP180103096 and FT170100128. Xuemin Lin is supported by 2019B15151200-48, 2018YFB1003504, NSFC61232006, ARC DP180103096, ARC DP200101338, and ARC DP170101628.

9. REFERENCES

[1] A. A. Al-Yamani, S. Ramsundar, and D. K. Pradhan. A defect tolerance scheme for nanotechnology circuits. *IEEE Trans. on Circuits and Systems*, 54-I(11):2402–2409, 2007.

[2] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics*, 145(1):11–21, 2004.

[3] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, E. Bertino, N. Foo, et al. Collusion detection in online rating systems. In *Asia-Pacific Web Conference*, pages 196–207. Springer, 2013.

[4] C. Ambühl, M. Mastrolilli, and O. Svensson. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM J. Comput.*, 40(2):567–596, 2011.

[5] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. 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*, pages 119–130, 2013.

[6] G. Fang, Y. Wu, M. Li, and J. Chen. An efficient algorithm for mining frequent closed itemsets. *Informatica (Slovenia)*, 39(1), 2015.

[7] Q. Feng, S. Li, Z. Zhou, and J. Wang. Parameterized algorithms for edge biclique and related problems. *Theoretical Computer Science*, 2017.

[8] A. Gely, L. Nourine, and B. Sadi. Enumeration aspects of maximal cliques and bicliques. *Discrete applied mathematics*, 157(7):1447–1459, 2009.

[9] A. Kershenbaum, A. Cutillo, C. Darabos, K. Murray, R. Schiaffino, and J. H. Moore. Bicliques in graphs with correlated edges: From artificial to biological networks. In *European Conference on the Applications of Evolutionary Computation*, pages 138–155. Springer, 2016.

[10] J. Konc and D. Janezic. An improved branch and bound algorithm for the maximum clique problem. *proteins*, 4(5), 2007.

[11] M. A. Langston, E. J. Chesler, and Y. Zhang. 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)*, volume 00, page 473, 07 2008.

[12] C.-M. Li, Z. Fang, H. Jiang, and K. Xu. Incremental upper bound for the maximum clique problem. *INFORMS Journal on Computing*, 30(1):137–153, 2017.

[13] C. M. Li and Z. Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI*, volume 10, pages 128–133, 2010.

[14] J. Li, H. Li, D. Soh, and L. Wong. A correspondence between maximal complete bipartite subgraphs and closed patterns. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 146–156. Springer, 2005.

[15] G. Liu, K. Sim, and J. Li. Efficient mining of large maximal bicliques. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 437–448. Springer, 2006.

[16] J. Liu and W. Wang. 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*, pages 187–194, 2003.

[17] C. Lu, J. X. Yu, H. Wei, and Y. Zhang. Finding the maximum clique in massive graphs. *PVLDB*, 10(11):1538–1549, 2017.

[18] C. Lucchese, S. Orlando, and R. Perego. Fast and memory efficient mining of frequent closed itemsets. *IEEE Trans. Knowl. Data Eng.*, 18(1):21–36, 2006.

[19] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Scandinavian Workshop on Algorithm Theory*, pages 260–272. Springer, 2004.

[20] P. Manurangsi. 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.

[21] E. Maslov, M. Batsyn, and P. M. Pardalos. Speeding up branch and bound algorithms for solving the maximum clique problem. *Journal of Global Optimization*, 59(1):1–21, 2014.

[22] C. McCreesh and P. Prosser. 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*, pages 226–234. Springer, 2014.

[23] R. G. Michael and S. J. David. Computers and intractability: a guide to the theory of np-completeness. *WH Free. Co., San Fr*, pages 90–91, 1979.

[24] A. P. Mukherjee and S. Tirthapura. Enumerating maximal bicliques from a large graph using mapreduce. *IEEE Transactions on Services Computing*, 10(5):771–784, 2017.

[25] R. A. Mushlin, A. Kershenbaum, S. T. Gallagher, and T. R. Rebbeck. A graph-theoretical approach for pattern discovery in epidemiological research. *IBM systems journal*, 46(1):135–149, 2007.

[26] R. Nataraj and S. Selvan. Parallel mining of large maximal bicliques using order preserving generators. *International Journal of Computing*, 8(3):105–113, 2014.

[27] R. Peeters. The maximum edge biclique problem is np-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003.

[28] M. J. Sanderson, A. C. Driskell, R. H. Ree, O. Eulenstein, and S. Langley. Obtaining maximal concatenated phylogenetic data sets from large sequence databases. *Molecular biology and evolution*, 20(7):1036–1042, 2003.

[29] E. Shaham, H. Yu, and X. Li. 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*, pages 315–323, 2016.

[30] S. Shahinpour, S. Shirvani, Z. Ertem, and S. Butenko. Scale reduction techniques for computing maximum induced bicliques. *Algorithms*, 10(4):113, 2017.

[31] M. B. Tahoori. Application-independent defect tolerance of reconfigurable nanoarchitectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2(3):197–218, 2006.

[32] A. Tanay, R. Sharan, and R. Shamir. 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*, pages 136–144, 2002.

[33] E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global optimization*, 37(1):95–111, 2007.

[34] E. Tomita and T. Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete mathematics and theoretical computer science*, pages 278–289. Springer, 2003.

[35] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *International Workshop on Algorithms and Computation*, pages 191–203. Springer, 2010.

[36] E. Tomita, K. Yoshida, T. Hatta, A. Nagao, H. Ito, and M. Wakatsuki. A much faster branch-and-bound algorithm for finding a maximum clique. In *International Workshop on Frontiers in Algorithmics*, pages 215–226. Springer, 2016.

[37] Y. Tong, L. Chen, and B. Ding. 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*, pages 270–281, 2012.

[38] J. Wang, J. Han, and J. Pei. 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*, pages 236–245. ACM, 2003.

[39] Y. Wang, S. Cai, and M. Yin. New heuristic approaches for maximum balanced biclique problem. *Information Sciences*, 432:362–375, 2018.

[40] B. Yuan and B. Li. A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar. *JETC*, 10(3):25:1–25:19, 2014.

[41] B. Yuan, B. Li, H. Chen, and X. Yao. A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem. *IEEE Trans. Cybernetics*, 45(5):1040–1053, 2015.

[42] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. Diversified top-k clique search. *VLDB J.*, 25(2):171–196, 2016.

[43] Y. Zhang, C. A. Phillips, G. L. Rogers, E. J. Baker, E. J. Chesler, and M. A. Langston. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinformatics*, 15:110, 2014.

[44] Y. Zhou and J.-K. Hao. Combining tabu search and graph reduction to solve the maximum balanced biclique problem. *arXiv preprint arXiv:1705.07339*, 2017.

[45] Y. Zhou, A. Rossi, and J.-K. Hao. Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs. *European Journal of Operational Research*, 269(3):834–843, 2018.