AMBEA: Aggressive Maximal Biclique Enumeration in Large Bipartite Graph Computing

Zhe Pan, *Member, IEEE*, Xu Li, Shuibing He, Xuechen Zhang, Rui Wang, Yunjun Gao, *Senior Member, IEEE*, Gang Chen, *Member, IEEE*, and Xian-He Sun, *Life Fellow, IEEE*

***Abstract*—Maximal biclique enumeration (MBE) in bipartite graphs is a fundamental problem in data mining with widespread applications. Many recent works solve this problem based on the set-enumeration (SE) tree, which sequentially traverses vertices to generate the enumeration tree nodes representing distinct bicliques, then checks whether these bicliques are maximal or not. However, existing MBE algorithms only expand bicliques with untraversed vertices to ensure distinction, which often necessitate extensive node checks to eliminate non-maximal bicliques, resulting in significant computational overhead during the enumeration process. To address this issue, we propose an aggressive set-enumeration (ASE) tree that aggressively expands all bicliques to their maximal form, thus avoiding costly node checks on non-maximal bicliques. This aggressive enumeration may produce multiple duplicate maximal bicliques, but we efficiently eliminate these duplicates by leveraging the connection between parent and child nodes and conducting low-cost node checking. Additionally, we introduce an aggressive merge-based pruning (AMP) approach that aggressively merges vertices sharing the same local neighbors. This helps prune numerous duplicate node generations caused by subsets of merged vertices. We integrate the AMP approach into the ASE tree, and present the Aggressive Maximal Biclique Enumeration Algorithm (AMBEA). Experimental results show that AMBEA is 1.15× to 5.32× faster than its closest competitor and exhibits better scalability and parallelization capabilities on larger bipartite graphs.**

***Index Terms*—Graph computing, bipartite graph, maximal biclique enumeration.**

Received 27 February 2024; revised 5 July 2024; accepted 5 August 2024. Date of publication 12 August 2024; date of current version 8 November 2024. This work was supported in part by the National Science Foundation of China under Grant 62172361, in part by the Major Projects of Zhejiang Province under Grant LD24F020012, in part by the National Key Research and Development Program of China under Grant 2023YFB4502100 and Grant 2021ZD0110700, and in part by the US National Science Foundation under CNS 2216108. Recommended for acceptance by M. Kandemir. *(Corresponding author: Shuibing He.)*

Zhe Pan, Xu Li, Shuibing He, Rui Wang, Yunjun Gao, and Gang Chen are with the State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou 310027, China, and also with Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, Hangzhou 310051, China (e-mail: panzhe@zju.edu.cn; fhxu@zju.edu.cn; heshuibing@zju.edu.cn; rwang21@zju.edu.cn; gaoyj@zju.edu.cn; cg@zju.edu.cn).

Xuechen Zhang is with the School of Engineering and Computer Science, Washington State University Vancouver, Vancouver, WA 98686 USA (e-mail: xuechen.zhang@wsu.edu).

Xian-He Sun is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 USA (e-mail: sun@iit.edu).

Digital Object Identifier 10.1109/TC.2024.3441864

I. Introduction

BIPARTITE graphs are widely used in various domains to represent relationships between two distinct sets of entities [1]. In a bipartite graph, vertices are divided into two disjoint sets and edges only connect vertices from different sets. A biclique is a complete bipartite subgraph that contains all possible edges connecting two disjoint sets of vertices. A maximal biclique is the biclique that cannot be further extended to include additional vertices. The main focus of this paper is to develop an efficient algorithm for enumerating all maximal bicliques (MBE) in large bipartite graphs.

MBE has emerged as a crucial tool in a wide range of applications, including anomaly detection in e-commerce networks [2], [3], [4], [5], [6], gene expression analysis in expression datasets [7], [8], [9], identification of overlapping communities detection [10], GNN information aggregation [11], and social recommendation in social networks [12] as well. For instance, in an e-commerce network like eBay or Alibaba, bipartite graphs are employed to represent the relationship between customers and their purchases. It is suspicious for a large group of customers to purchase a set of products because there is a high probability that online sellers ask crowdsourcing platforms to click farm for their products to increase exposure [2], [3], [4], [5]. Such suspicious transactions can be well modeled by maximal bicliques, as shown in Fig. 1. If there exists a fast MBE algorithm capable of identifying all the maximal bicliques in the network, it becomes possible to detect suspicious groups to a significant extent. Consequently, the research on MBE has attracted great attention from both academia and industry in recent years [7], [13], [14], [15], [16], [17], [18], [19], [20], [21].

Enumerating all maximal bicliques in large bipartite graphs is known to be a computationally challenging task [22]. Recent studies have employed the set-enumeration (SE) tree approach to address this challenge [23]. The SE tree systematically traverses all vertex sets in a bipartite graph, generating enumeration tree nodes that represent all possible bicliques. Subsequently, MBE algorithms determine whether these bicliques are maximal or non-maximal. However, this approach requires extensive node checks to eliminate non-maximal bicliques. Consequently, this process incurs a significant computational overhead. To mitigate this issue, several state-of-the-art MBE algorithms have proposed various techniques to reduce the enumeration space, such as node pruning and vertex ordering [7], [16], [18]. Despite these optimization efforts, the node

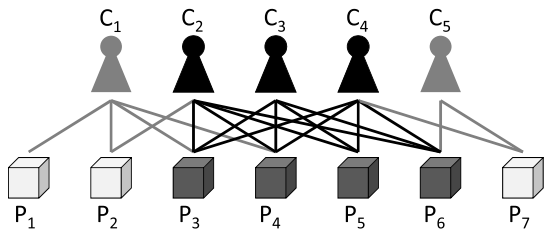


Fig. 1. An example of maximal biclique in the e-commerce network. Customers C_2, C_3, C_4 purchase products P_3, P_4, P_5, P_6 together. ($\{C_2, C_3, C_4\}$, $\{P_3, P_4, P_5, P_6\}$) forms a maximal biclique.

checking for non-maximal bicliques remains the performance bottleneck. For instance, experiments conducted on the Github dataset [18] demonstrate that even the latest MBE algorithm, ooMBEA, requires $25.69\times$ more node checks for non-maximal bicliques than those for maximal bicliques.

Our research has identified two main reasons for the performance issue. First, *the existing SE trees for MBE have structural limitations.* They traverse vertices sequentially to generate new nodes. To ensure that each node represents a distinct biclique, each node only expands its corresponding biclique with untraversed vertices, excluding those that have been traversed. Consequently, the biclique may be non-maximal due to the exclusion of these vertices. For example, node z in Fig. 1 produces a non-maximal biclique because it cannot expand its biclique with v_2 since its ancestor node x has traversed v_2 to generate node y. Second, *the mainstream pruning approaches work passively.* They start from a special vertex v^* and prune nodes with non-maximal bicliques based on the relationship between v^* and other vertices [7], [16], [18]. Therefore, the pruning efficiency heavily relies on the selection of v^* and whether the node generated by v^* produces a maximal biclique.

To address the aforementioned structural limitation, first, we propose an aggressive set-enumeration (ASE) tree for MBE. Unlike the existing SE trees, the ASE tree allows for the aggressive expansion of bicliques in new nodes using all available vertices. As a result, all bicliques within the ASE tree are guaranteed to be maximal, albeit with the possibility of duplicate bicliques. Then the ASE tree leverages the connection between parent and child nodes to eliminate these duplicates. We also prove the correctness of the ASE tree. Notably, the ASE tree for MBE offers two significant advantages. (1) It enables efficient node pruning through low-cost node checking. And (2) the ASE tree tends to exhibit a more balanced structure compared to other trees, making it highly suitable for parallelization.

Second, to enhance the pruning efficiency of the existing passive pruning approaches, we further introduce an aggressive merge-based pruning (AMP) approach. We observe that vertices with the same local neighbors always appear together in maximal bicliques across all MBE algorithms. This insight allows us to safely merge these homogeneous vertices in advance, pruning duplicate nodes that would be generated by subsets of these merged vertices. The challenge lies in identifying vertices with the same local neighbors. Directly comparing the local neighbors of each vertex pair within each node leads to high computation overhead. To overcome this challenge, the AMP approach initially treats all candidate vertices as a unified entity and partitions them based on their local neighbors. This partitioning step is performed at once, enabling efficient merging of vertices with the same local neighbors during node generation.

Third, we design the Aggressive Maximal Biclique Enumeration Algorithm (AMBEA) by integrating the AMP approach into the ASE tree. To evaluate the performance of AMBEA, we conducted experiments using 12 representative real-world datasets and 5 synthetic datasets of varying scales. The experimental results demonstrate that AMBEA outperforms its closest competitor by $1.15\times$ to $5.32\times$, while also reducing the number of unproductive nodes by $2.37\times$ to $8.98\times$. Moreover, AMBEA exhibits enhanced scalability and parallelization capabilities, making it well-suited for handling large bipartite graphs.

Our principal contributions are summarized as follows.

- We propose an innovative aggressive set-enumeration (ASE) tree, which constantly expands bicliques to their maximal form using all available vertices, and efficiently eliminates duplicate maximal bicliques by utilizing the parent-child relationship between nodes. This ASE tree not only enables efficient elimination of unproductive nodes but also enhances parallelization by creating a more balanced enumeration tree (Section III).
- We introduce an efficient aggressive merge-based pruning approach (AMP), which efficiently merges vertices with the same local neighbors, resulting in the pruning of duplicate nodes that would otherwise be generated by subsets of these merged vertices. The merge-based pruning technique can be universally applied to other MBE algorithms, making it a practical and versatile tool in various scenarios (Section IV).
- We integrate the AMP approach into the ASE tree and propose an efficient MBE algorithm AMBEA (Section V). Extensive experiments demonstrate the superior performance of AMBEA compared to the state-of-the-art algorithms in terms of running time, pruning efficiency, scalability, and parallelization (Section VI).

II. Background

In this section, we formulate the MBE problem, review recent popular SE-tree-based MBE algorithms, and analyze the limitations associated with these algorithms.

A. Problem Formulation

Let $G(U, V, E)$ denote an undirected bipartite graph, where U and V are two sets of disjoint vertices in G, and the edge set E consists of edges in G with $E \subseteq U \times V$. We denote a vertex as u or v, where $u \in U$ and $v \in V$. $N(v)$ denotes the neighbors of a vertex v in G. $d(v)$ denotes the degree, i.e., number of neighbors, of a vertex v, and $d(V)$ denotes the maximum degree of vertices in V. $\Gamma(X)$ represents the common neighbors of the vertices in X, i.e., $\Gamma(X) = \bigcap_{v \in X} N(v)$. $\Upsilon(X)$ represents the union neighbors of the vertices in X, i.e., $\Upsilon(X) = \bigcup_{v \in X} N(v)$.

In a bipartite graph $G(U, V, E)$, a *biclique* $B(L, R, E')$ is a complete bipartite subgraph of G that is induced by two vertex

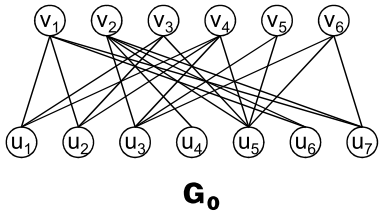


Fig. 2. Input bipartite graph G_0.

sets L and R, satisfying $L \subseteq U, R \subseteq V$, and $E' = L \times R \subseteq E$. In short, we denote the biclique B using the set pair (L, R). A *maximal biclique* is a biclique (L, R) for which there is no larger biclique (L^*, R^*) such that $(L \cup R)$ is a proper subset of $(L^* \cup R^*)$. The main objective of this paper is to *enumerate all maximal bicliques in a bipartite graph*.

B. SE-Tree-Based MBE Algorithms

To systematically enumerate all maximal bicliques, the set-enumeration (SE) [23] approach is commonly used and forms the basis for some of the most efficient maximal biclique enumerating (MBE) algorithms [7], [15], [16], [18], [21]. The SE tree defines a tree *node structure* that represents a distinct biclique and performs *node generation* for enumerating all possible bicliques. Afterward, it examines each biclique through *node checking* to determine if it is a maximal biclique. In the following, we formulate the SE tree from the above three perspectives: (1) *How to define the node structure in the SE tree?* (2) *How to generate new nodes?* (3) *How are nodes identified as maximal bicliques or not?*

Node structure: Each tree node is represented as a 3-tuple (L, R, C). In a bipartite graph $G(U, V, E)$, L is a subset of U, while R and C are two disjoint subsets of V. The biclique of the current node is formed by L, R, and the fully connected edges between them. C consists of candidate vertices that can potentially be added to expand R in the subsequent node generation process.

Node generation: The tree traversal starts with a root node (L, R, C), where L is initialized as U, R is empty, and C is set to V. We perform the node generation process from the current node (L, R, C), and sequentially traverse each candidate vertex v' in C to generate a new child node (L', R', C'). In the new child node, L' consists of vertices that belong to both L and $N(v')$, i.e., $L' = L \cap N(v')$, representing the common neighbors of $R \cup \{v'\}$. R' adds the untraversed vertices in C (including v') that have edge connections with all vertices in L' to R, i.e., $R' = R \cup (\Gamma(L') \cap C)$. C' contains the remaining vertices in C that connect with some but not all vertices in L', i.e., $C' = C \cap (\Upsilon(L') \setminus \Gamma(L'))$. After traversing v', it is removed from the current node, specifically $C = C \setminus \{v'\}$, and we proceed to select another candidate vertex from C to generate another child node. This process continues until C becomes empty.

Node checking: The node (L', R', C') is valid and outputs a maximal biclique if and only if $\Gamma(L') = R'$.

Algorithm 1: SE-tree-based MBE Algorithm

```
Input: Bipartite graph G(U, V, E)
Output: All maximal bicliques
1  biclique_search_se(U, ∅, V);
2  procedure biclique_search_se(L, R, C):
3      foreach v' ∈ C do
4          L' ← L ∩ N(v'); R' ← R; C' ← ∅;
5          foreach v_c ∈ C do
6              if L' ∩ N(v_c) = L' then
7                  R' ← R' ∪ {v_c};
8              else if L' ∩ N(v_c) ≠ ∅ then
9                  C' ← C' ∪ {v_c};
10         if Γ(L') = R' then
11             Output(L', R') as a maximal biclique;
12             biclique_search_se(L', R', C')
13         C ← C \ {v'};
```

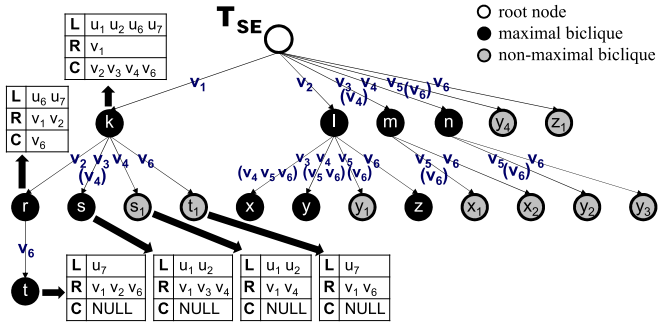


Fig. 3. The basic SE tree T_{SE} on the bipartite graph G_0.

We summarize the SE-tree-based MBE algorithms in Algorithm 1. The algorithm starts with the root node $(U, \emptyset, V)$ and recursively calls the procedure `biclique_search_se` (line #1). When processing a current node (L, R, C), the procedure sequentially traverses each vertex v' in C (lines #3, 13) to generate new child node (L', R', C') (lines #4-9) followed by a node checking (line #10). If the biclique (L', R') is maximal, the procedure reports the maximal biclique and proceeds to explore new nodes from (L', R', C') (line #12).

Example 1: Fig. 3 depicts an enumeration tree T_{SE} on a bipartite graph G_0 using Algorithm 1.[1] The enumeration process commences from the root node, where we recursively explore subspaces by sequentially traversing candidate vertices in a depth-first manner. Initially, we generate node k by traversing v_1. Node k passes the node checking since $\Gamma(L_k) = \{v_1\} = R_k$. Next, node k generates node r by traversing v_2, resulting in $L_r = N(v_2) \cap L_k = \{u_3, u_4, u_5, u_6, u_7\} \cap \{u_1, u_2, u_6, u_7\} = \{u_6, u_7\}$ and $R_r = R_k \cup (C_k \cap \Gamma(L_r)) =$

[1]To facilitate comparison, all MBE algorithms operate on the same bipartite graph G_0. In the enumeration tree, nodes with identical letters in their identifiers share the same set L. Only nodes without subscripts in their identifiers output maximal bicliques. For example, nodes s, s_1, and s_2 have the same set L, but only node s outputs a maximal biclique. We use subscripts on the vertex set, such as L_s denoting set L of node s.

$\{v_1\} \cup (\{v_2, v_3, v_4, v_5\} \cap \{v_1, v_2\}) = \{v_1, v_2\}$. C_r solely contains v_6 because v_3, v_4, and v_5 do not connect with any vertex in L_r. Continuing this process, we generate node s with biclique $(\{u_1, u_2\}, \{v_1, v_3, v_4\})$ and node s_1 with biclique $(\{u_1, u_2\}, \{v_1, v_4\})$. However, node s_1 corresponds to a non-maximal biclique because $\Gamma(L_{s_1}) = \{v_1, v_3, v_4\} \neq R_{s_1} = \{v_1, v_4\}$. Compared to node s, node s_1 generates a non-maximal biclique because node k fails to expand set R of node s_1 with v_3 since v_3 has been removed after generating node s. Thus, we remove node s_1 after the node checking phase. The similar generation of other nodes is exemplified in Fig. 3.

The SE-tree-based MBE algorithms have a time complexity of $O(|V|d(V)\beta)$, where node generation takes $O(|V|d(V))$ time because it involves a maximum of $|V|$ set intersections (each taking $O(d(V))$ time), node checking takes $O(|E|) = O(|V|d_{avg}(V))$ time because we can obtain $O(\Gamma(L'))$ by accessing each edge once in bipartite graph, and β represents the total biclique count, including non-maximal instances removed by node checking. The basic SE tree leads to an extensive enumeration space as the power set of the vertex set V, i.e., $\beta = O(2^{|V|})$, causing the performance bottleneck. Various recent optimizations have been proposed to reduce the enumeration space and expedite the MBE process, which can be categorized into the following groups:

Pivot-based pruning approaches. To reduce the need for costly node checks, some node pruning techniques are proposed to proactively remove non-maximal bicliques before generating the corresponding tree nodes. PMBE [16] and ooMBEA [18] employed the pivot-based pruning approaches. A *"pivot"* refers to a vertex that aids in node pruning based on the neighborhood containment relationship. Specifically, given a node (L, R, C) and a pivot v^* in C, nodes induced by vertices $v' \in C$ can be safely pruned if $N(v') \cap L \subset N(v^*)$ (excluding v^*), as their corresponding bicliques can be further expanded using v^*. Initially, PMBE leveraged the pivot technique based on the global containment relationship but overlooked local containment relationships. To address this limitation, ooMBEA proposed the batch-pivots technique by performing a 2-hop depth-first search (DFS) on each vertex in C. However, it should be noted that this additional DFS operation incurs a time complexity of $O(|E|)$ per vertex, equivalent to other node checking approaches.

Passive merge-based pruning approach. iMBEA [7] conducted proactive node pruning through vertex merging. Specifically, when a node (L, R, C) traverses a vertex v^* to generate a node (L', R', C'), vertices are merged only if (L', R') is maximal. This process merges vertices v' in C such that $N(v') \cap L = L'$, resulting in the pruning of nodes induced by those vertices. The rationale behind this merging is that the nodes generated by the merged vertices will always produce non-maximal bicliques, as they can be further expanded using the traversed vertex v^*. However, this approach has limitations as it only passively performs the vertex merging when the node induced by v^* passes the maximality checking process.

Vertex ordering approaches. SE-tree-based MBE algorithms typically employ a specific traversal order for candidate vertices (line #3 in Algorithm 1) to help prune the search space [12]. While iMBEA [7] and FMBE [15] enforce the vertex order at each node, PMBE [16] and ooMBEA [18] establish a global vertex order. Specifically, for a current node (L, R, C), iMBEA and FMBE sort vertices $v' \in C$ based on $|N(v') \cap L|$ in increasing order. On the other hand, PMBE and ooMBEA presort all vertices in V before enumeration, utilizing their newly proposed orders known as rev-topological order and unilateral order, respectively. The choice of vertex order is flexible and its effectiveness depends on how well it works together with the algorithm.

Node checking facilitation. Some studies [7], [18] include an extra set Q, which stores traversed vertices, at each node to facilitate node checking. A node generates a non-maximal biclique if there exists a vertex v_q in Q such that L is a subset of $N(v_q)$ because this biclique can be further expanded by vertex v_q. However, set Q introduces inefficiency as it requires extra memory and computation. Therefore, we omit the use of the set Q and determine if a node produces a non-maximal biclique by checking if there exists a vertex v^* in $N(u_l)$, where u_l is a vertex in L, such that L is a subset of $N(v^*)$ and v^* is not in R or C.

C. Performance Issue in Existing Algorithms

Despite optimization efforts, SE-tree-based MBE algorithms struggle with performance due to node checks for eliminating non-maximal bicliques, with two main limitations:

(1) **Existing SE trees for MBE have structural limitations.** SE-tree-based MBE algorithms typically involve two steps. First, the SE tree is used to generate all possible bicliques without repetition. In this step, nodes in existing SE trees are restricted to expanding their set R only with untraversed vertices, excluding those that have already been traversed. This limitation can result in numerous non-maximal bicliques since some traversed vertices are omitted. Second, non-maximal bicliques are removed to output only the maximal ones. However, the node generation and checking for these extensive non-maximal bicliques often become the performance bottleneck. For example, our experiments show that even the recent MBE algorithm ooMBEA [18], despite its optimizations, needs to check and remove 25.69× more nodes with non-maximal bicliques compared to maximal bicliques on the Github dataset [24]. Therefore, reducing the generation of unproductive nodes is crucial to alleviate the computational burden of the node checking process.

(2) **Mainstream pruning approaches work passively.** We conduct a comprehensive review of current pruning approaches for MBE and identified their passive nature. To our observation, both existing pivot-based and merge-based pruning approaches rely on specific vertices to initiate the pruning process, rather than directly pruning nodes. Specifically, pivot-based approaches involve selecting specific vertices, known as pivots, and incur additional computational overhead for node pruning using these pivots. On the other hand, the passive merge-based approach only conducts vertex merging if the node induced by the traversing vertex v^* passes the maximality checking, and it only merges vertices that share the same *local neighbors* with

v^*, where the local neighbors of a vertex $v_c \in C$ in a node (L, R, C) are represented by the intersection between $N(v_c)$ and set L, denoted as $N_L(v_c)$. Hence, these passive pruning approaches have limitations as their pruning efficiency heavily depends on specific vertices.

III. Aggressive Set-Enumeration Tree

We introduce an aggressive set-enumeration (ASE) tree for MBE to tackle the structural limitation in existing SE trees. Then, we prove its correctness and analyze its benefit.

A. Proposed ASE Tree for MBE

Main idea: Existing SE-tree-based MBE algorithms have a limitation in that they only generate tree nodes (bicliques) with untraversed vertices to ensure distinction, but often require extensive node checks to eliminate non-maximal bicliques. To overcome this limitation, we propose a novel **aggressive set-enumeration (ASE) tree**, which aggressively expand all bicliques to their maximal form during the *node generation* process, by allowing the inclusion of all vertices, to avoid checking their maximality. While this aggressive expansion may result in duplicate bicliques, we address this during the *node checking* process by using a low-cost method to eliminate duplicate instances and output the distinct maximal bicliques.

Prior definition: Before delving into the details of the ASE tree, we define some important terms and notations:

Definition 1: $\vec{\mathbf{v}}$. $\vec{v}$ refers to the traversed candidate vertex on the edge from the parent node to the child node in the enumeration tree. For instance, in Fig. 3, $\vec{v}$ of node k is v_1.

Definition 2: $\mathbf{X}_{\mathbf{v}}^{+}, \mathbf{X}_{\mathbf{v}}^{-}$. Given a predetermined order, set X can be divided into two parts: X_v^+ contains vertices with higher orders (larger IDs) than v, and X_v^- contains the other vertices, i.e., vertices with lower and equal orders than v.

Node generation: The ASE tree traversal starts from a root node $(U, \emptyset, V)$ and sequentially traverses each candidate vertices v' in C from the current node (L, R, C) to generate a new node (L', R', C'). In the new node, L' represents the common neighbors of $R \cup \{v'\}$, i.e., $L' = L \cap N(v')$. The difference now is that R' contains all common neighbors of vertices in L', i.e., $R' = \Gamma(L')$. C' consists of vertices in C with higher orders than v' and connect with any vertices in L', i.e., $C' = C_{v'}^+ \cap (\Upsilon(L') \setminus \Gamma(L'))$.

Node checking: The node checking mechanism comprises two main components:

(1) *Basic node checking for duplication elimination:* To identify and eliminate all duplicate maximal bicliques generated by the above node generation process, we use the basic node checking based on mapping that assigns a unique *target vertex* v'_T and a unique *target parent node* (L_T, R_T) to each maximal biclique (L', R'). We only output maximal bicliques on the nodes generated from these unique target vertices and target parent nodes, and eliminate other duplicates.

The mapping details are as follows:

$$v'_T = \min\{v \in R' \mid \Gamma(R'^{-}_{v}) = L'\} \tag{1}$$

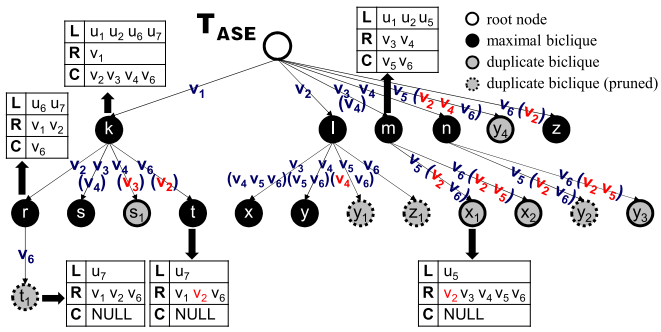


Fig. 4. The ASE tree T_{ASE} on the bipartite graph G_0. Unlike T_{SE}, traversed vertices for aggressive node generation are highlighted in red.

$$\begin{cases} v_T = \min\{v \in R' \mid \Gamma(R'^{-}_{v}) \cap N(v'_T) = L'\} \\ L_T = \Gamma(R'^{-}_{v_T}) \\ R_T = \Gamma(L_T) \end{cases} \tag{2}$$

In this mapping, v'_T denotes the smallest vertex v in R' for which L' exactly matches the common neighbors of all vertices in R' up to and including v according to Equation 1. After determining v'_T, if $N(v'_T)$ is identical to L', (L_T, R_T) refers to the root node. Otherwise, according to Equation 2 we can obtain an intermediate vertex v_T, which is the smallest vertex v in R' for which L' exactly matches the intersection between $\Gamma(R'^{-}_{v})$ and $N(v'_T)$. Then, we can derive L_T as $\Gamma(R'^{-}_{v})$, and R_T as the common neighbors of vertices in L_T.

Based on this mapping, the node (L', R', C') is valid and outputs a maximal biclique if two conditions holds:

- O1: The current node's $\vec{v}$ is the target vertex v'_T.
- O2: The parent node corresponds to the target parent maximal biclique (L_T, R_T).

(2) *Low-cost node checking for further node pruning.* To speed up the examination of duplicates in the ASE tree, we further introduce a low-cost node checking that only examines the local neighbors of candidate vertices. This approach allows us to safely prune the node generated by candidate vertex $v_c$if local neighbors of v_c at the current node remain identical to those at the parent node, denoted by $L_{parent} \cap N(v_c) = L_{current} \cap N(v_c)$, because both the parent and current nodes would generate identical bicliques by traversing vertex v_c based on the node generation rule. According to Equation 2, the mapping mechanism selects the minimum v_T with maximum $|L_T|$, so the current node cannot correspond to the biclique (L_T, R_T) since $|L_{current}| < |L_{parent}|$. This low-cost node checking only requires set intersections in $O(d(V))$, whereas other approaches require at least $O(|E|)$ for computing $\Gamma(L')$ or $\Gamma(R'^{-}_{v_T})$.

Example 2: Fig. 4 depicts an enumeration tree T_{ASE} on the bipartite graph G_0. The main difference between T_{ASE} and T_{SE} in Fig. 3 lies in the node generation process. Specifically, in T_{ASE}, each node can be further expanded with traversed vertices, as highlighted in the figure. For instance, when node k in T_{SE} traverses v_6, it generates node t_1 with a non-maximal biclique because T_{SE} cannot expand R_{t_1} using v_2, as v_2 has already been traversed to generate node r. In contrast, in T_{ASE}, node k in T_{ASE} traverses v_6 to generate node t with a maximal

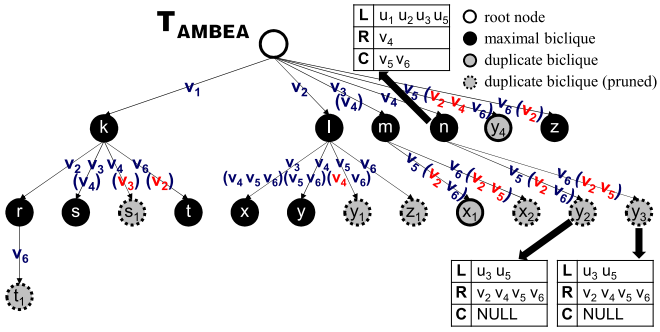


Fig. 5. The enumeration tree T_{AMBEA} on the bipartite graph G_0 based on AMBEA.

biclique because T_{ASE} can expand R_t with v_2. Therefore, all bicliques in T_{ASE} are maximal but duplicate bicliques may exist, such as node t and node t_1.

T_{ASE} employs an aggressive node checking rule to eliminate duplicate bicliques. To illustrate the conditions for node checking, we will use nodes x_1 and t_1 as examples. For node x_1, according to Equation 1, we sequentially add vertices from R_{x_1} to an empty set X until $\Gamma(X)$ equals L_{x_1}. By doing so, we determine that v'_T of node x_1 is v_3, as $\Gamma(\{v_2, v_3\}) = \{u_5\} = L_{x_1}$. Consequently, node x_1 does not satisfy condition O1 since its $\vec{v}$ is v_5, not v_3. For node t_1, we obtain its v'_T as v_6 in the same way. Similarly, using Equation 2, we deduce that its v_T is v_1 since $\Gamma(\{v_1\}) \cap N(v_6) = \{u_7\} = L_{t_1}$. Then, we find that its $L_T = \Gamma(\{v_1\}) = \{u_1, u_2, u_6, u_7\}$. Consequently, node t_1 does not satisfy condition O2 since set L of its parent node r (i.e., $L_r = \{u_6, u_7\}$) is not identical to its L_T.

Compared to T_{SE}, T_{ASE} outputs the same set of maximal bicliques. Moreover, T_{ASE} prunes more nodes than T_{SE} through a low-cost node checking, as seen with nodes t_1, y_1, z_1, and y_2. For instance, node r proactively prunes node t_1 since $L_k \cap N(v_6) = \{u_7\} = L_r \cap N(v_6)$.

In terms of time complexity, each node takes $O(|V|d(V))$ for both node generation and node checking, as it involves $O(|V|)$ set intersections, each taking $O(d(V))$ time. Finally, ASE-tree-based MBE algorithms run in $O(|V|d(V)\beta)$, which is equivalent to the time complexity of Algorithm 1.

B. Correctness of the ASE-Tree-Based MBE Algorithm

Below, we provide a proof of the correctness of our ASE-tree-based MBE algorithm by establishing Theorem 1.

Theorem 1: The ASE tree precisely outputs all maximal bicliques while eliminating non-maximal or duplicate bicliques.

Proof: The ASE tree avoids outputting non-maximal bicliques due to its node generation rule, which states that $L' = L \cap N(v') = \Gamma(R \cup \{v'\})$ and $R' = \Gamma(L')$. This guarantees that (L', R') always meets the definition of a maximal biclique.

Furthermore, the node checking rule ensures that each maximal biclique is output without duplicates. Specifically, the mapping mechanism ensures that each maximal biclique (L', R') has a unique target vertex v'_T and a unique target parent maximal biclique (L_T, R_T). For a given node corresponding to (L_T, R_T), we know that v'_T is always included in its candidate set C_T because v'_T is always greater than v_T and connects with some but not all vertices in L_T. Thus, if target parent node (L_T, R_T, C_T) exists, it can generate the node with (L', R') by traversing v'_T. By following this process, the corresponding node of each maximal biclique can recursively obtain its unique target parent node until reaching the root node. Thus, the ASE tree enumerates all maximal bicliques exactly once, eliminating duplicates. □

C. Characteristics of the ASE Tree

Notably, the ASE tree for MBE has two advantageous properties. (1) *It enables aggressive node pruning through low-cost node checking.* When no pruning techniques are considered, both T_{SE} and T_{ASE} trees always involve the same number of nodes. For instance, both T_{SE} in Fig. 3 and T_{ASE} in Fig. 4 involve 10 valid maximal bicliques and 9 invalid bicliques. This is because both trees ensure that v'_T in Equation 1 is always $\vec{v}$ of a valid node (L', R', C'), and candidate set C' is always $(\Upsilon(L') \setminus \Gamma(L'))^+_{v'_T}$. Thus, compared to the basic SE tree, the ASE tree proactively prunes invalid nodes through low-cost node checking, reducing the number of nodes that need to undergo the costly node checking process. (2) *The ASE tree is generally more balanced than others, making it more suitable for parallelization.* For nodes that output the same maximal biclique (L', R') in different trees, the parent node in the ASE tree always has the minimum $\vec{v}$, i.e., v_T in Equation 2. As a result, the depth of such nodes is always the smallest, leading to a more balanced ASE tree.

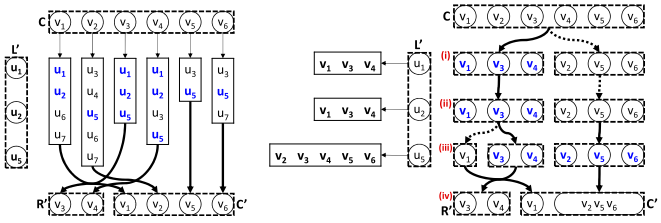
IV. Aggressive Merge-Based Pruning

We further propose a novel aggressive merge-based pruning (AMP) approach to tackle the passive pruning limitation in existing SE trees for MBE. This section first introduces this AMP approach and then analyzes its benefit.

A. Proposed AMP Approach for MBE

Inspired by the neighborhood-based tree pruning approach in [7], we notice that vertices with the same local neighbors (refer to Section II-C) always appear together in maximal bicliques. Therefore, we can aggressively merge these vertices with the same local neighbors into a single group, and prune all nodes induced by the subset of these merged vertices. We emphasize that this aggressive merge-based pruning is quite different from the existing passive merge-based pruning approach. The passive approach only merges vertices when one of the merging vertices is chosen as the traversing vertex for generating the child node, thus limiting pruning efficiency which is determined by the choice of the traversing vertex. *In contrast, our AMP approach aggressively merges these vertices during node generation, thereby enabling complete pruning of all nodes induced by these homogeneous vertices.*

However, it is challenging to identify and merge all vertices that have the same local neighbors within each node due to the potential high merging overhead. To accomplish this, one naive approach is to compute and store the local



(a) Baseline approach. (b) AMP approach.

Fig. 6. Node generation processes for node m in Fig. 4.

neighbors of candidate vertices and compare them pairwise. However, this straightforward method has a time complexity of $O(|V|^2 d(V))$, as it involves $O(|V|^2)$ pairwise comparisons, each taking $O(d(V))$ time. This complexity exceeds the time complexity of conducting node generation and node checking on each node, which runs in $O(|V|d(V))$.

To minimize the merging overhead, we optimize the node generation process. Unlike existing MBE algorithms that compute the local neighbors of candidate vertices sequentially to generate nodes, the AMP approach treats candidate vertices as a unified entity and partitions them based on their local neighbors. Initially, all candidates in C are considered as a single group. Then, vertices u_l in L' are sequentially selected for group partition. By checking whether the vertex in a group is connected to the incoming vertex u_l, each group can be split into two subgroups. Consequently, candidate vertices with the same local neighbors are always placed in the same group. Finally, the vertices within each group are merged in an aggressive manner.

Example 3: Following Example 2, Fig. 6 illustrates two node generation processes for node m in Fig. 4. To facilitate comparison, we assume that C contains all vertices in V. The neighbors of each vertex are represented by solid rectangles, with an incoming edge indicating the corresponding vertex. Local neighbors are highlighted.

Fig. 6(a) shows that the baseline approach sequentially computes local neighbors for each candidate vertex. On the other hand, Fig. 6(b) demonstrates the AMP approach, which partitions candidate vertices into multiple groups using the vertices in L' sequentially. At step (i), C is divided into two groups since v_1, v_3, and v_4 connect with u_1, while v_2, v_5, and v_6 don't. At step (iii), group $\{v_1, v_3, v_4\}$ is further divided into two groups since v_3 and v_4 connect with u_5, while v_1 doesn't. At step (iv), we aggressively merge v_2, v_5, and v_6 together since they have the same local neighbors (i.e., $\{u_5\}$).

As a result, the AMP approach successfully prunes node x_2 generated by v_6 because it finds that candidate vertices v_5 and v_6 have the same local neighbors at node m. In contrast, the passive merge-based pruning approach fails to prune node x_2 because node x_1 generated by v_5 does not pass the node checking, thus not triggering the pruning condition. The AMP approach can further prune nodes s_1 and y_3 similarly.

Algorithm 2 outlines the main procedure of the AMP approach. The algorithm utilizes a structural array called $GroupArray$ to maintain the group information dynamically. Each element in $GroupArray$ represents a group and contains

Algorithm 2: Main procedure of the AMP approach

Data: Bipartite graph $G(U, V, E)$
Input: Original candidate set C, new set L'
Output: New candidate set C'

```
1  struct GroupInfo:
2      Integer incoming_v;
3      Integer next_gid;
4      Integer ln_size;
5  procedure partition_merge (C, L'):
6      Assign a new attribute gid = 0 on each vertex in C;
7      GroupArray.append(GroupInfo (∞, ∞, 0));
8      foreach u_l ∈ L' do
9          foreach v_c ∈ C ∩ N(u_l) do
10             cid ← v_c.gid;
11             if GroupArray[cid].incoming_v ≠ u_l then
12                 new_gid ← GroupArray.size();
13                 GroupArray[cid].incoming_v ← u_l;
14                 GroupArray[cid].next_gid ← new_gid;
15                 GroupArray.append(GroupInfo
                     (∞, ∞, GroupArray[cid].ln_size + 1));
16             v_c.gid ← GroupArray[cid].next_gid;
17     C' ← merge vertices in C with the same gid, excluding
         groups whose ln_size is 0 or |L'|;
18     return C'
```

three attributes: $incoming\_v$, $next\_gid$, and ln_size (lines #1-4). $incoming\_v$ and $next\_gid$ attributes are used to dispatch current vertex to an appropriate group, while ln_size attribute indicates the local neighborhood size of any vertex within the group. Additionally, each candidate vertex in C is assigned a unique identifier, gid, which allows the vertex to retrieve its group information through its group index (line #6). Initially, $GroupArray$ consists of only one group, which includes all candidate vertices (line #7). Then, the algorithm iteratively partitions the candidate vertices using each vertex u_l in L' (line #8). For vertices with the same gid, the algorithm dispatches vertices connected to u_l into a new group (lines #9-16). A new group is created when encountering the first vertex with a specific gid. Upon creating a new group, the algorithm increments the local neighborhood size by 1. To optimize memory usage, we suggest recycling groups with no elements, which can be achieved using a stack structure. Finally, the algorithm merges vertices that have the same local neighbors since they always share the same gid (line #17). The time complexity of the `partition_merge` procedure is $O(|E|)$ as it only requires accessing each edge in E once.

B. Characteristics of the AMP Approach

Note that the AMP approach offers two key contributes: (1) *It enables node pruning through aggressive vertex merging.* The AMP approach is the first pruning approach capable of efficiently merging all homogeneous vertices with identical local neighbors. Unlike other pruning methods, it proactively prunes nodes without relying on specific vertices. It is applicable to all MBE algorithms and improves performance across the board. (2) *It facilitates efficient processing by reducing the time*

Algorithm 3: Aggressive MBE Algorithm (AMBEA)

Input: Bipartite graph $G(U, V, E)$
Output: All maximal bicliques
1 Sort vertices in V based on their neighborhood sizes in increasing order;
2 **foreach** $v \in V$ **do**
3 `biclique_search_ambea` $(U, \emptyset, V, v)$;
4 **procedure** `biclique_search_ambea` (L, R, C, v'):
5 $L' \leftarrow L \cap N(v')$; $\overline{R'} \leftarrow \Gamma(L')$;
6 $C' \leftarrow$ `partition_merge` $(C^{+}_{v'}, L')$;
7 $\overline{L} \leftarrow \emptyset'$; $\overline{R} \leftarrow R'^{-}_{v'} \setminus (R \cup C)$;
8 **foreach** $u_l \in L \setminus L'$ **do**
9 **if** $(R'^{-}_{v'} \setminus \{v'\}) \subseteq N(u_l)$ **then**
10 $\overline{L} \leftarrow \overline{L} \cup \{u_l\}$;
11 **if** $\overline{L} \neq \emptyset \wedge \overline{R} = \emptyset$ **then**
12 **Output**(L', R') as a maximal biclique;
13 **foreach** $v'_c \in C'$ **do**
14 **if** $N(v'_c) \cap \overline{L} \neq \emptyset$ **then**
15 `biclique_search_ambea` (L', R', C', v'_c)

complexity for each enumeration node. Compared to existing approaches, which require $O(|V|d(V))$ for processing each node, the AMP approach can be easily applied to them and reduce the time complexity for node processing to $O(|E|) = O(|V|d_{avg}(V))$, which is lower than $O(|V|d(V))$.

V. Aggressive Maximal Biclique Enumeration Algorithm

In this section, we present an enhanced algorithm for MBE called the aggressive maximal biclique enumeration algorithm (AMBEA) by combining the aforementioned ASE tree and AMP. We provide a comprehensive explanation of the design of AMBEA, including algorithm analysis, and develop a parallel version of the algorithm to achieve further acceleration.

A. Proposed AMBEA

To explore a highly efficient MBE algorithm, we develop the AMBEA as illustrated in Algorithm 3. Given a bipartite graph $G(U, V, E)$, AMBEA initially sorts vertices in V based on their neighborhood sizes in increasing order (line #1), since this order has been proven to be the most effective for AMBEA, as shown in Section VI-C. Next, this algorithm sequentially traverses each v in V and recursively calls the `biclique_search_ambea` procedure (lines #2-3). This procedure incorporates the AMP approach for node generation (lines #5,6) and maintains $\overline{L}$ and $\overline{R}$ for node checking in the ASE tree (lines #7-10, 14). Specifically, $\overline{L}$ contains all vertices in $L \setminus L'$ that connect with all vertices smaller than v' in R', given by $\overline{L} = \Gamma(R'^{-}_{v'} \setminus \{v'\}) \setminus L'$. If $\overline{L} \neq \emptyset$ (line #11), it implies $\Gamma(R'^{-}_{v'} \setminus \{v'\}) \neq L'(i.e., \Gamma(R'^{-}_{v'}))$, ensuring that v' satisfies Equation 1 and condition O1 is met. $\overline{L}$ is used for low-cost node checking (line #14) since $N(v'_c) \cap \overline{L} = \emptyset$ implies the local neighbors of v'_c have not changed. $\overline{R}$ contains all vertices smaller than v' in R' that connect with all vertices in L'. Let $\vec{v}$ of node (L, R, C) be v^*. Because local neighbors of v' change at node (L, R, C) (line #14), we know $\Gamma(R^{-}_{v^*} \setminus \{v^*\}) \cap N(v') \neq \Gamma(R^{-}_{v^*}) \cap N(v')$. If $\overline{R} = \emptyset$ (line #11), it implies $R'^{-}_{v'} \subseteq R \cup C$. Then we know $R'^{-}_{v^*} \subseteq R^{-}_{v^*}$, since $v^* < v'$ and all vertices in C are greater than v^*. We conclude $R'^{-}_{v^*} = R^{-}_{v^*}$ because $R' \supset R$. Hence, $\Gamma(R'^{-}_{v^*}) = \Gamma(R^{-}_{v^*}) = L$, which implies that (L, R) satisfies Equation 2 and condition O2 is met.

Example 4: Fig. 5 illustrates the enumeration tree T_{AMBEA} generated on the bipartite graph G_0 using Algorithm 3. In comparison to T_{ASE} shown in Fig. 4, T_{AMBEA} incorporates additional pruning of nodes, namely s_1, x_2, and y_3 utilizing the AMP approach. For instance, node n prunes node y_2 through low-cost node checking (line #14), as $N(v_5) \cap L_{root} = \{u_3, u_5\} = N(v_5) \cap L_n$. Then node n prunes node y_3 through the AMP approach (line #6) since v_5 and v_6 at node n have identical local neighbors, i.e., $\{u_3, u_5\}$. As a result, AMBEA is able to prune all unproductive child nodes of node n. In contrast, existing passive pruning approach cannot prune all unproductive child nodes because they have to check the node generated by the special v^* at least. By comparing T_{AMBEA} with T_{SE} in Fig. 3, AMBEA successfully prunes 7 out of 9 unproductive nodes, thus achieving superior pruning efficiency through the combination of the ASE tree and the AMP approach.

Time complexity. The time complexity of AMBEA consists of two parts: the vertex ordering time (line #1) and the generation time of the enumeration tree (lines #2,3). The vertex ordering time can be bounded in $O(|V|log(|V|))$. To ensure a fair comparison, we use β to represent the total number of nodes in the enumeration tree, including both nodes that output maximal bicliques and unproductive nodes that are pruned by node checking. We know AMBEA generates the enumeration tree in $O(|E|\beta)$ because AMBEA processes a total of β nodes and all the computations for $L', R', C', \overline{L}, \overline{R}$ in each node can be bounded by $O(|E|)$. Therefore, the time complexity of AMBEA is $O(|E|\beta + |V|log(|V|)) = O(|E|\beta)$ since the overhead for vertex ordering is much smaller than the overhead for generating the enumeration tree.

AMBEA achieves the minimum time complexity of $O(|E|)$ for processing each node, which is equal to ooMBEA as demonstrated in Section II-B. This time complexity is superior to other algorithms that require at least $O(|V|d(V))$. In practice, the value of β for AMBEA can be significantly smaller than that of other competitors due to its high pruning efficiency. The experimental results in Section VI-B demonstrate that AMBEA outperforms other algorithms because it can prune a larger number of unproductive nodes through low-cost node checking in the ASE tree (Section III) and the AMP approach (Section IV).

Space complexity. The space complexity of AMBEA consists of two components: the input bipartite graph and the memory required to maintain the enumeration tree. The space complexity of the input bipartite graph is $O(|E|)$. As AMBEA generates the enumeration tree in a DFS manner, it needs to store one node at each level of the enumeration tree for backtracking. The number of levels, namely the height of the enumeration tree, can be bounded by $O(d(V))$. Each node requires $O(|L| + |R| + |C| + |\overline{L}| + |\overline{R}|) = O(d(V) + |V|) = O(|V|)$ space. Consequently, the space complexity

TABLE I
DATASET STATISTICS

Datasets	Type	$\|U(G)\|$	$\|V(G)\|$	$\|E(G)\|$	$d(U)$	$d(V)$	Maximal Bicliques
Unicode (UL)	Country-Hosts-Language	614	254	1,255	141	69	460
UCforum (UF)	User-Post-Forum	899	522	7,089	99	126	16,261
MovieLens (Mti)	Tag-Assignment-Movie	16,528	7,601	71,154	640	146	140,266
Teams (TM)	Athlete-Membership-Team	901,130	34,461	1,366,466	17	2,671	517,943
ActorMovies (AM)	Movie-Appearance-Actor	383,640	127,823	1,470,404	646	294	1,075,444
Wikipedia (WC)	Article-Inclusion-Category	1,853,493	182,947	3,795,796	54	11,593	1,677,522
YouTube (YG)	User-Membership-Group	94,238	30,087	293,360	1,035	7,591	1,826,587
StackOverflow (SO)	User-Favorite-Post	545,195	96,680	1,301,942	4,917	6,119	3,320,824
DBLP (Pa)	Author-Authorship-Publication	5,624,219	1,953,085	12,282,059	287	1,386	4,899,032
IMDB (IM)	Movie-Appearance-Actor	896,302	303,617	3,782,463	1,590	1,334	5,160,061
BookCrossing (BX)	User-Rating-Book	340,523	105,278	1,149,739	2,502	13,601	54,458,953
Github (GH)	User-Membership-Project	120,867	56,519	440,237	3,675	884	55,346,398
LJ5	User-Membership-Group	1,837,928	853,994	5,610,437	25	52,775	2,181,295
LJ10	User-Membership-Group	2,301,031	1,421,088	11,227,130	49	105,454	7,430,705
LJ15	User-Membership-Group	2,548,619	1,912,139	16,843,216	63	158,562	22,727,251
LJ20	User-Membership-Group	2,704,651	2,357,485	22,456,757	79	211,506	61,836,924
LJ25	User-Membership-Group	2,812,080	2,771,510	28,068,423	88	263,711	151,468,807

of AMBEA is $O(|E| + |V|d(V))$, which matches the state-of-the-art methods. Since the ASE tree tends to generate a balanced enumeration tree with a small height, as mentioned in Section III-A, AMBEA can benefit from memory savings.

B. Parallel Version of AMBEA

In order to further improve the efficiency, we have developed a parallel version of AMBEA called ParAMBEA. Drawing inspiration from [15], ParAMBEA executes multiple AMBEA procedures concurrently and parallelizes operations within each AMBEA procedure. Following the steps outlined in Algorithm 3, ParAMBEA assigns a thread to each procedure corresponding to each vertex $v \in V$ (lines #2,3). Additionally, whenever there are available threads, ParAMBEA dynamically creates subprocedures (lines #13-15). To fully utilize parallelism, ParAMBEA parallelizes all the for loops within the `biclique_search_ambea` procedure (e.g., lines #8-10, 13-15) by utilizing loop unrolling techniques. Since the ASE tree is inclined to generate balanced enumeration trees, as mentioned in Section III-A, this characteristic greatly helps in load balancing for ParAMBEA.

VI. EVALUATION

In this section, we conduct extensive experiments to evaluate performance of the proposed techniques.

A. Experimental Setup

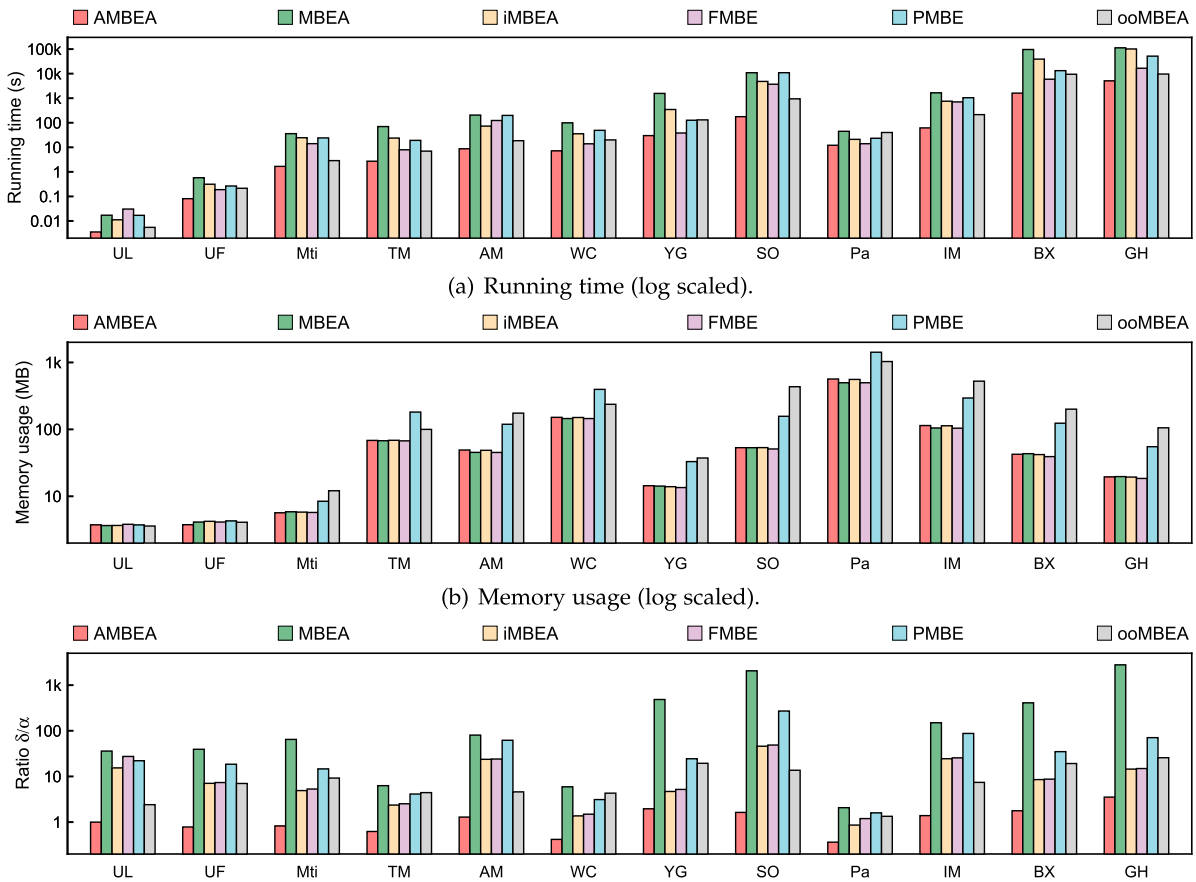
Platform. All experiments are conducted on a machine equipped with four Intel Xeon(R) Gold 5318Y 2.10GHz CPUs (24 cores per CPU) and 128GB of main memory. The operating system used is Linux kernel-5.4.0. Unless stated otherwise, the algorithms are executed using a single core.

Competitors. In our experiments, we compare our serial algorithm, AMBEA, with five state-of-the-art serial MBE algorithms: MBEA [7], iMBEA [7], FMBE [15], PMBE [16], and ooMBEA [18]. We also compare our parallel algorithm, ParAMBEA, with the cutting-edge parallel MBE algorithms, ParMBE [15] and GMBE [21]. Our code is available at https://github.com/ISCS-ZJU/AMBEA. We obtain all competitor implementations in the repository [25]. Additionally, we implemented several other variants to assess the different techniques proposed in this paper. We detail these variants in the corresponding experiments.

Datasets. We use both real-world and synthetic datasets to validate the proposed techniques. Synthetic datasets are generated by sampling 5%, 10%, 15%, 20%, and 25% of the edges from the large LiveJournal dataset ($|U|$=7,489,073, $|V|$=3,201,203, $|E|$=112,307,385). These generated datasets are named as LJ5, LJ10, LJ15, LJ20, and LJ25, respectively, based on the percentage of edges sampled from the original dataset. We collect all real-world datasets, including the LiveJournal dataset, from the KONECT repository [24] covering various domains. Since the bipartite graph consists of two symmetrical vertex sets, we denote the set with fewer vertices as V, where $|U| > |V|$. We arrange real-world datasets in increasing order based on their maximal biclique counts, as the running time of the MBE problem is mainly influenced by the number of maximal bicliques present in the dataset. The dataset statistics are presented in Table I.

B. Overall Evaluation

We evaluate each MBE algorithm on all 12 real-world datasets using three metrics: running time, memory consumption, and pruning efficiency on unproductive nodes. To ensure fairness, we run each algorithm in a separate process. We measure the execution time of the process, excluding the time taken for graph loading from the disk, and monitor the maximum memory usage of the process. To gauge the pruning efficiency, we record the number of node checks performed by each MBE algorithm as β. Given that the count of nodes outputting maximal bicliques remains fixed (denoted as α), we can determine the number of unproductive nodes as $\delta = \beta - \alpha$. Therefore,



(a) Running time (log scaled).

(b) Memory usage (log scaled).

(c) Ratio of δ/α (log scaled). Since the number of maximal bicliques (α) in each dataset remains fixed, a smaller ratio δ/α indicates a higher proportion of unproductive nodes (δ) being pruned.

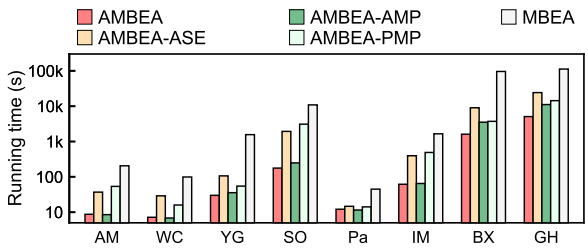
Fig. 7. Overall evaluation.

pruning efficiency is measured by the ratio δ/α, where a smaller ratio indicates higher pruning efficiency.

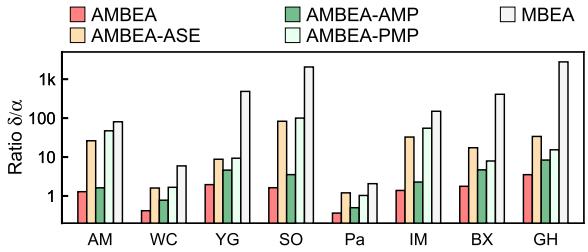
Fig. 7(a) compares the running time of AMBEA with mainstream MBE algorithms on real-world datasets. It is evident that the performance of existing MBE algorithms exhibits significant variations with changes in datasets. For instance, on datasets like StackOverflow, IMDB, and Github, ooMBEA outperforms MBEA, iMBEA, FMBE, and PMBE. However, on datasets with large $d(V)/d(U)$ like YouTube, DBLP, and BookCrossing, FMBE proves to be 1.59×-3.43× faster than ooMBEA. Fortunately, our proposed AMBEA consistently achieves the minimum running time across all test datasets due to its aggressive pruning of unproductive nodes. Notably, AMBEA surpasses its closest competitor by 1.15×-5.32×. Specifically, on the BookCrossing dataset, AMBEA completes in 1,609 seconds, while all other competitors require over 5,934 seconds.

Fig. 7(b) illustrates the memory usage of MBE algorithms. The experimental results show that AMBEA, MBEA, iMBEA, and FMBE exhibit comparable memory requirements across all datasets. In contrast, PMBE and ooMBEA noticeably require more memory compared to the other algorithms. This disparity arises due to the fact that AMBEA, MBEA, iMBEA, and FMBE only require memory to store the input bipartite graph and a set of nodes for backtracking purposes. However, PMBE necessitates additional memory to store the CDAG index structure, as mentioned in [16], and ooMBEA partitions the input bipartite graph into multiple subgraphs, thereby requiring additional memory to accommodate these subgraphs. Specifically, on the Github dataset, PMBE requires 55 MB of memory, and ooMBEA requires 105 MB of memory, while the other algorithms require no more than 20 MB of memory.

Fig. 7(c) compares the pruning efficiency of MBE algorithms. The experiment results show that existing algorithms have a major performance issue: they spend considerable time checking and eliminating unproductive nodes, as discussed in Section II-B. Specifically, on the Github dataset, all existing MBE algorithms require 15.58× more node checks than the number of maximal bicliques, resulting in a substantial computational overhead. Fortunately, AMBEA effectively addresses this issue by reducing the ratio δ/α by 2.37×-8.98× compared to the next-best competitors. This efficient pruning is the key reason why AMBEA is faster than the other algorithms.



(a) Running time (log scaled).



(b) Ratio of δ/α (log scaled).

Fig. 8. Effect of the ASE tree and the AMP approach. Variants AMBEA-ASE and AMBEA-AMP enable the ASE tree and the AMP, respectively, on MBEA. Both AMBEA-ASE and AMBEA-AMP outperform MBEA because their δ/α is far smaller than that of MBEA. AMBEA has the minimum running time in most cases since it has the minimum δ/α.

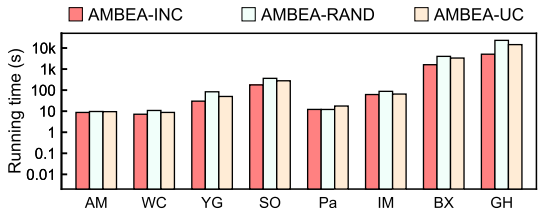


Fig. 9. Effect of vertex ordering.

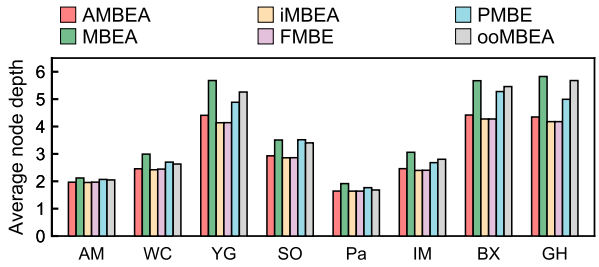


Fig. 10. Comparison of average node depth among MBE algorithms.

C. Breakdown Evaluations

We evaluate the effects of the ASE tree, the AMP approach, and the vertex ordering on 8 datasets with a larger number of maximal bicliques.

Effect of the ASE tree. To study the effect of the ASE tree proposed in Section III, we design a variant AMBEA-ASE that only enables the ASE tree on MBEA. This is because MBEA generates the enumeration tree according to Algorithm 1 without any optimizations. As shown in Fig. 8, AMBEA-ASE consistently outperforms MBEA by aggressively pruning numerous nodes that do not output maximal bicliques through the low-cost node checking. In particular, AMBEA-ASE achieves a speedup of 14.7× compared to MBEA on the YouTube dataset, as it prunes 98.2% of unproductive nodes in MBEA.

Effect of the AMP approach. To study the effect of the AMP approach proposed in Section IV, we design two variants for comparison: AMBEA-AMP, which enables only the AMP approach on MBEA, and AMBEA-PMP, which enables only the passive merge-based pruning (PMP) approach in Section II-B. As shown in Fig. 8, both AMBEA-AMP and AMBEA-PMP consistently outperform MBEA, as they both prune numerous unproductive nodes through vertex merging. Moreover, AMBEA-AMP shows superior pruning efficiency compared to AMBEA-PMP. In particular, on the StackOverflow dataset, AMBEA-AMP achieves a speedup of 12.5× compared to AMBEA-PMP, as it is able to further prune 96.5% of unproductive nodes that AMBEA-PMP could not eliminate. This is due to the fact that AMBEA-AMP aggressively merges vertices with the same local neighbors, whereas AMBEA-PMP can only do so if the corresponding node passes the node checking process.

Effect of vertex ordering. To study the effect of vertex ordering as mentioned in Section V-A, we design three variants AMBEA-INC, AMBEA-RAND, and AMBEA-UC that sort vertices in V in increasing order based on degree, random order, and the unilateral order proposed by the latest MBE algorithm ooMBEA [18], respectively. Fig. 9 shows that AMBEA-INC outperforms both AMBEA-RAND and AMBEA-UC on most datasets, particularly those with a large number of maximal bicliques. For instance, on the Github dataset, AMBEA-INC only requires 5,085 seconds, while AMBEA-RAND and AMBEA-UC require 23,067 seconds and 14,500 seconds, respectively. Thus, AMBEA always sorts vertices in V in increasing order based on their degree.

D. Sensitivity Analysis

Impact of different algorithms on the balance of their enumeration trees. To explore the impact of different algorithms on the balance of the enumeration tree, we measure the depth of each tree node and calculate the average node depth for nodes that output maximal bicliques across various MBE algorithms, as shown in Fig. 10. A lower average node depth indicates a more balanced enumeration tree. We observe that AMBEA always generates a more balanced enumeration tree compared to MBEA, as evidenced by the smaller average node depth. The AMP approach does not affect the average node depth since it does not impact the generation of nodes that output maximal bicliques. PMBE and ooMBEA tend to generate imbalanced enumeration trees. This can be attributed to their use of the pivot-based pruning approach, where a candidate vertex v' is forced to generate a node with a maximal biclique in the next level if its local neighbors are a subset of those of a pivot vertex v^*, i.e., $N_L(v') \subseteq N_L(v^*)$. iMBEA and FMBE tend to generate balanced enumeration trees by reordering vertices at each node. However, this reordering process incurs significant overhead at each node. Overall, the choice of algorithm has a

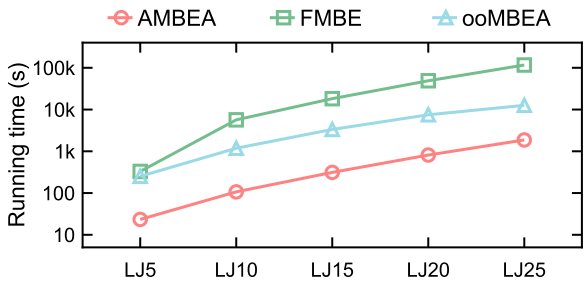


Fig. 11. Evaluation on scalability (log scaled).

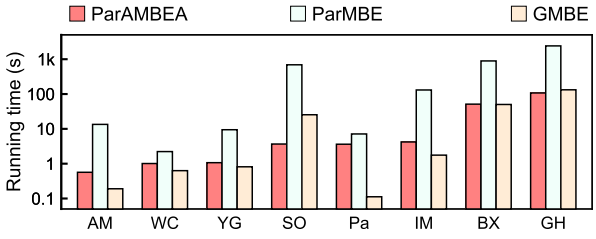


Fig. 12. Overall evaluation on parallelization (log scaled). ParAMBEA and ParMBE run on a 96-core CPU machine. GMBE runs on an A100 GPU.

noticeable impact on the balance of enumeration trees, with AMBEA achieving a better balance compared to other MBE algorithms.

Evaluation on scalability. To evaluate the scalability of AMBEA, we conduct experiments using synthetic datasets of varying scales. For comparison, we focus on AMBEA in relation to FMBE and ooMBEA, as these two algorithms have demonstrated strong performance on large datasets, as depicted in Fig. 7(a). As illustrated in Fig. 11, the results demonstrate that AMBEA is significantly faster than FMBE by $47.2\times - 141.7\times$, and outperforms ooMBEA by $6.7\times - 11.1\times$ on synthetic datasets. FMBE consistently lags behind ooMBEA because all synthetic datasets are generated from the LiveJournal dataset, resulting in $d(V)$ exceeding 50,000. Specifically, on the LJ25 dataset, AMBEA completes in 1,870 seconds, while FMBE and ooMBEA require 116,582 seconds and 12,614 seconds, respectively. Experimental results indicate the scalability of AMBEA when applied to large datasets.

Evaluation on parallelization. To evaluate the parallel performance of AMBEA, we compare its parallel version, ParAMBEA, with cutting-edge parallel algorithms, including CPU-based ParMBE and emerging GPU-based GMBE. In Fig. 12, we test ParAMBEA and ParMBE using 96 threads on a 96-core CPU machine, while GMBE runs on an A100 GPU [26] using the default configuration in [21]. Through highly efficient pruning, ParAMBEA achieves a speedup of $2.0\times$ to $198.6\times$ compared to ParMBE. Despite running on a 96-core CPU machine, ParAMBEA performs comparably to GMBE, which leverages the advanced A100 GPU with thousands of lightweight cores. Notably, on the StackOverflow dataset, ParAMBEA is $6.9\times$ faster than GMBE due to the successful workload balancing of the ASE tree. Additionally, we test ParAMBEA and ParMBE on four time-consuming datasets

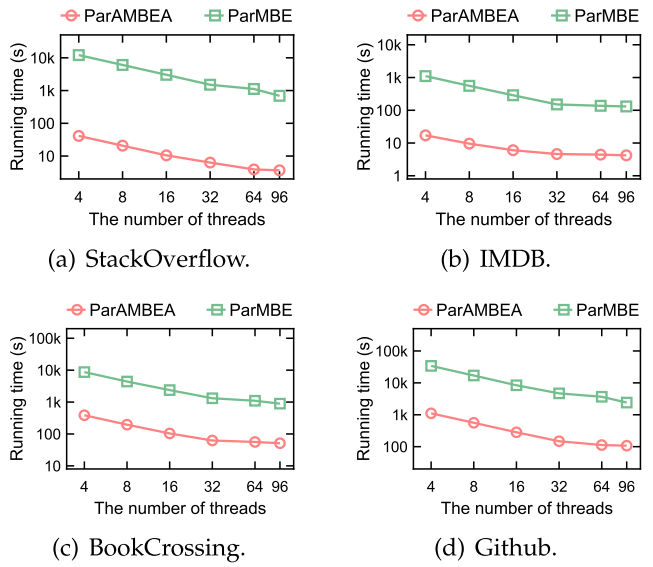


Fig. 13. Evaluations on parallelization with varying thread counts (log scaled).

using varying numbers of threads (4 to 96). As shown in Fig. 13, ParAMBEA achieves significant speedups of $17.4\times$ to $293.8\times$ compared to ParMBE, demonstrating nearly linear scaling with the increasing number of threads.

VII. Related Work

Maximal biclique enumeration. Early methods employed a brute-force BFS approach to solve the MBE problem [13], [27]. Recent studies have utilized the SE tree to enhance the enumeration process in a DFS manner, incorporating optimization techniques such as pruning and vertex ordering [7], [15], [16], [18]. Parallel MBE algorithms [15], [21] are also proposed for speed-up the enumeration. However, these SE-tree based methods still require extensive node checks for non-maximal bicliques, causing performance bottlenecks. Our proposed AMBEA addressed this issue by leveraging ASE and AMP techniques. Additionally, Gely et al. [28] reduced MBE to the maximal clique enumeration problem but introduced extensive computational overhead by adding additional edges to the bipartite graph. Li et al. [29] reduced MBE to the frequent closed itemset mining problem, but their method is not suitable for large bipartite graphs due to high memory requirements for storing the graph in binary format.

Customized biclique enumeration. Emerging studies have explored the enumeration of customized bicliques in bipartite graphs. Yang et al. [11], [30] and Ye et al. [31] addressed the enumeration of all (p, q)-bicliques. Yao et al. [32] introduced the concept of vertex similarity and enumerated maximal similar bicliques, where vertices exhibit similarity. Yin et al. [33] extended the problem by assigning attributes to vertices and enumerating fair bicliques, where vertices with different attributes are considered roughly equal. Chen et al. [34] focused on biclique percolation communities, which are unions of maximal bicliques. Wang et al. [35] enumerated maximal τ-bicliques on uncertain bipartite graphs, where each edge has an existence

probability. However, these approaches are inadequate for solving the MBE problem, as they only conduct node checks for their-defined bicliques, rather than maximal bicliques.

Maximum biclique finding. Other research has focused on finding one or several special bicliques (L, R) among all maximal bicliques. Lyu et al. [3], [4] and Wang et al. [5] addressed the problem of finding the Maximum Edge Biclique (MEB), which maximizes $|L| \times |R|$. Chen et al. [36] focused on finding the Maximum Balanced Biclique (MBB) with $|L|$ strictly equal to $|R|$ and the maximum number of vertices. Moreover, other works have explored finding maximum bicliques in signed [20], [37] or weighted bipartite graphs [38], [39]. Nonetheless, these methods have limitations in effectively accelerating MBE because they selectively prune the search space that does not lead to their target bicliques, but these pruned search spaces may still be part of the MBE solution space.

VIII. Conclusion

In this paper, we proposed the ASE tree, which aggressively expands bicliques with all vertices and eliminates potential duplicates through the connection between parent and child nodes. The ASE tree effectively prunes unproductive nodes using a low-cost node checking mechanism and maintains a balanced enumeration tree. To further enhance pruning efficiency, we introduced the AMP approach, which aggressively merges vertices with the same local neighbors. This is achieved by optimizing the node generation process. We then presented AMBEA, which applies the AMP approach to the ASE tree, along with its parallel version, ParAMBEA. Through extensive experiments, we demonstrated the effectiveness of AMBEA and all the proposed techniques. The experimental results showed that AMBEA can accelerate maximal biclique enumeration, while also exhibiting better scalability on large bipartite graphs.

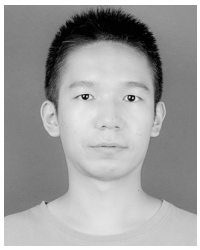
Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and feedbacks. Special thanks are due to Yang Wang, Kai Zeng, and Jingdong Li for their insightful discussions.

References

[1] C. Maier and D. Simovici, "Bipartite graphs and recommendation systems," *J. Adv. Inf. Technol.*, vol. 13, no. 3, pp. 249–258, Jun. 2022.
[2] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, E. Bertino, N. Foo, "Collusion detection in online rating systems," in *Proc. Web Technol. Appl. (APWeb),* Berlin, Heidelberg: Springer, 2013, pp. 196–207.
[3] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum biclique search at billion scale," *Proc. VLDB Endowment*, vol. 13, no. 9, pp. 1359–1372, 2020.
[4] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum and top-k diversified biclique search at scale," *VLDB J.*, vol. 31, no. 6, pp. 1365–1389, 2022.
[5] K. Wang, W. Zhang, X. Lin, L. Qin, and A. Zhou, "Efficient personalized maximum biclique search," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, 2022, pp. 498–511.
[6] Z. Li et al., "What happens behind the scene? Towards fraud community detection in e-commerce from online to offline," in *Proc. Companion Proc. Web Conf. (WWW'21 Companion)*, 2021, pp. 105–113.
[7] 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 Bioinformat.*, vol. 15, no. 1, 2014, Art. no. 110.
[8] T. Siswantining, A. Bustamam, O. Swasti, and H. S. Al-Ash, "Analysis and prediction of protein interactions between HIV-1 protein and human protein using LCM-MBC algorithm combined with association rule mining," *Commun. Math. Biol. Neurosci. (CMBN)*, p. 19, 2021.
[9] Y. Liu, "Computational methods for identifying microRNA-gene regulatory modules," in *Handbook of Statistical Bioinformatics,* Berlin, Heidelberg: Springer, 2022, pp. 187–208.
[10] T. Alzahrani and K. J. Horadam, "Finding maximal bicliques in bipartite networks using node similarity," *Appl. Netw. Sci.*, vol. 4, no. 1, pp. 21: 1–21:25, 2019.
[11] J. Yang, Y. Peng, and W. Zhang, "(p, q)-biclique counting and enumeration for large sparse bipartite graphs," *Proc. VLDB Endowment*, vol. 15, no. 2, pp. 141–153, 2021.
[12] G. Liu, K. Sim, and J. Li, "Efficient mining of large maximal bicliques," in *Proc. Data Warehousing Knowl. Discovery (DaWaK)*, 2006, pp. 437–448.
[13] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone, "Consensus algorithms for the generation of all maximal bicliques," *Discrete Appl. Math.*, vol. 145, no. 1, pp. 11–21, 2004.
[14] Y. He, R. Li, and R. Mao, "An optimized MBE algorithm on sparse bipartite graphs," in *Proc. Int. Conf. Smart Comput. Commun,* Cham, Switzerland: Springer, 2018, pp. 206–216.
[15] A. Das and S. Tirthapura, "Shared-memory parallel maximal biclique enumeration," in *Proc. IEEE 26th Int. Conf. High Perform. Comput., Data, Analytics (HiPC)*, 2019, pp. 34–43.
[16] A. Abidi, R. Zhou, L. Chen, and C. Liu, "Pivot-based maximal biclique enumeration," in *Proc. 29th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2020, pp. 3558–3564.
[17] C. Qin, M. Liao, Y. Liang, and C. Zheng, "Efficient algorithm for maximal biclique enumeration on bipartite graphs," in *Proc. Adv. Natural Comput., Fuzzy Syst. Knowl. Discovery,* Cham, Switzerland: Springer, 2020, pp. 3–13.
[18] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient maximal biclique enumeration for large sparse bipartite graphs," *Proc. VLDB Endowment*, vol. 15, no. 8, pp. 1559–1571, 2022.
[19] Z. Ma, Y. Liu, Y. Hu, J. Yang, C. Liu, and H. Dai, "Efficient maintenance for maximal bicliques in bipartite graph streams," *World Wide Web*, vol. 25, no. 2, pp. 857–877, 2022.
[20] R. Sun, Y. Wu, C. Chen, X. Wang, W. Zhang, and X. Lin, "Maximal balanced signed biclique enumeration in signed bipartite graphs," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, 2022, pp. 1887–1899.
[21] Z. Pan, S. He, X. Li, X. Zhang, R. Wang, and G. Chen, "Efficient maximal biclique enumeration on GPUs," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, 2023, pp. 1–13.
[22] D. Eppstein, "Arboricity and bipartite subgraph listing algorithms," *Inf. Process. Lett.*, vol. 51, no. 4, pp. 207–211, 1994.
[23] R. Rymon, "Search through systematic set enumeration," in *Proc. 3rd Int. Conf. Princ. Knowl. Representation Reasoning (KR)*, 1992, pp. 539–550.
[24] J. Kunegis, "KONECT: The Koblenz network collection," in *Proc. 22nd Int. Conf. World Wide Web (WWW'13 Companion)*, 2013, pp. 1343–1350.
[25] Z. Pan, "GMBE source code," 2023. Accessed: Oct. 14, 2024. [Online]. Available: https://github.com/ISCS-ZJU/GMBE
[26] "NVIDIA a100 tensor core GPU," 2023. Accessed: Oct. 14, 2024. [Online]. Available: https://www.nvidia.com/en-gb/data-center/a100/
[27] M. J. Sanderson, A. C. Driskell, R. H. Ree, O. Eulenstein, and S. Langley, "Obtaining maximal concatenated phylogenetic data sets from large sequence databases," *Mol. Biol. Evol.*, vol. 20, no. 7, pp. 1036–1042, 2003.
[28] A. Gély, L. Nourine, and B. Sadi, "Enumeration aspects of maximal cliques and bicliques," *Discrete Appl. Math.*, vol. 157, no. 7, pp. 1447–1459, 2009.
[29] J. Li, G. Liu, H. Li, and L. Wong, "Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 12, pp. 1625–1637, Dec. 2007.
[30] J. Yang, Y. Peng, D. Ouyang, W. Zhang, X. Lin, and X. Zhao, "(p, q)-biclique counting and enumeration for large sparse bipartite graphs," *VLDB J.*, vol. 32, pp. 1137–1161, 2023.
[31] X. Ye, R.-H. Li, Q. Dai, H. Qin, and G. Wang, "Efficient biclique counting in large bipartite graphs," *Proc. ACM Manage. Data*, vol. 1, no. 1, pp. 1–26, 2023.

[32] K. Yao, L. Chang, and J. X. Yu, "Identifying similar-bicliques in bipartite graphs," *Proc. VLDB Endowment*, vol. 15, no. 11, pp. 3085–3097, 2022.
[33] Z. Yin, Q. Zhang, W. Zhang, R.-H. Li, and G. Wang, "Fairness-aware maximal biclique enumeration on bipartite graphs," in *Proc. IEEE 39th Int. Conf. Data Eng. (ICDE)*, 2023, pp. 1665–1677.
[34] Z. Chen, Y. Zhao, L. Yuan, X. Lin, and K. Wang, "Index-based biclique percolation communities search on bipartite graphs," in *Proc. IEEE 39th Int. Conf. Data Eng. (ICDE)*, 2023, pp. 2699–2712.
[35] J. Wang, J. Yang, Z. Ma, C. Zhang, S. Yang, and W. Zhang, "Efficient maximal biclique enumeration on large uncertain bipartite graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12634–12648, Dec. 2023.
[36] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient exact algorithms for maximum balanced biclique search in bipartite graphs," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, 2021, pp. 248–260.
[37] R. Sun, C. Chen, X. Wang, W. Zhang, Y. Zhang, and X. Lin, "Efficient maximum signed biclique identification," in *Proc. IEEE 39th Int. Conf. Data Eng. (ICDE)*, 2023, pp. 1313–1325.
[38] Y. Zhao, Z. Chen, C. Chen, X. Wang, X. Lin, and W. Zhang, "Finding the maximum k-balanced biclique on weighted bipartite graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 8, pp. 7994–8007, Aug. 2023.
[39] J. Wang, J. Yang, C. Zhang, and X. Lin, "Efficient maximum edge-weighted biclique search on large bipartite graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 8, pp. 7921–7934, Aug. 2023.



Zhe Pan (Member, IEEE) received the B.S. and Ph.D. degrees in computer science and technology from Zhejiang University, Hangzhou, China. He is currently a Postdoctoral Researcher with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include computer architecture, graph computing, and hardware-software co-design.

Xu Li received the B.S. degree in intelligent science and technology from Xiamen University, Xiamen, China, in 2021, and the M.S. degree in electronic information from Zhejiang University, Hangzhou, China, in 2024. His research interests include graph computing, parallel algorithms, and hardware-aware software optimization.

Shuibing He received the Ph.D. degree in computer science and technology from Huazhong University of Science and Technology, in 2009. Currently, he is a ZJU100 Young Professor with the College of Computer Science and Technology, Zhejiang University, China. His research interests include intelligent computing, high-performance computing, graph computing, and memory and storage systems. He is a member of the ACM.

Xuechen Zhang received the M.S. and Ph.D. degrees in computer engineering from Wayne State University. Currently, he is an Associate Professor with the School of Engineering and Computer Science, Washington State University Vancouver. His research interests include the areas of file and storage systems, operating systems, and high-performance computing. He is a member of the ACM.

Rui Wang received the Ph.D. degree in computer science from the University of Science and Technology of China (USTC), in 2021. Currently, she is a ZJU100 Research Fellow with the School of Software Engineering, Zhejiang University (ZJU). She worked as a Postdoctoral Fellow with the School of Computer Science and Technology, ZJU from 2021 to 2023. Her research interests include graph computing and graph storage.

Yunjun Gao (Senior Member, IEEE) received the Ph.D. degree in computer science from Zhejiang University, China, in 2008. Currently, he is a Professor with the College of Computer Science, Zhejiang University, China. His research interests include spatial and spatio-temporal databases, metric and incomplete or uncertain data management, and spatio-textual data processing. He is a member of the ACM and a senior member of the CCF.

Gang Chen (Member, IEEE) received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China. Currently, he is a Professor with the College of Computer Science, Zhejiang University. He has led successful research projects focused on building China's own database management systems. His research interests encompass relational database systems, large-scale data management technologies, and intelligent computing.

Xian-He Sun (Life Fellow, IEEE) received the B.S. degree in mathematics from Beijing Normal University, China, in 1982, and the M.S. and Ph.D. degrees in computer science from Michigan State University, in 1987 and 1990, respectively. He is a Distinguished Professor with the Department of Computer Science, Illinois Institute of Technology (IIT), Chicago. His research interests include parallel and distributed processing, memory and I/O systems.