Efficient Personalized Maximum Biclique Search

Kai Wang[†], Wenjie Zhang[†], Xuemin Lin[†], Lu Qin[⋆], Alexander Zhou[‡]
[†]University of New South Wales, [⋆]University of Technology Sydney, [‡]Hong Kong University of Science and Technology
kai.wang@unsw.edu.au, {zhangw, lxue}@cse.unsw.edu.au, lu.qin@uts.edu.au, atzhou@cse.ust.hk

***Abstract*—Bipartite graphs are naturally used to model relationships between two different types of entities. On bipartite graphs, maximum biclique search is a fundamental problem that aims to find the complete bipartite subgraph (biclique) with the maximum number of edges and is widely adopted for many applications such as anomaly detection in E-commerce and social network analysis. However, maximum biclique search only identifies the biclique whose size is globally maximum, whereas fast microscopic (personalized) analysis is needed in many real-world scenarios. For instance, when a suspected user is identified in an E-commerce network (e.g., a user-product network), it is important to quickly find the anomalous group containing the user and send the group of users for further human expert investigation. To fill this research gap, for the first time, we study the efficient personalized maximum biclique search problem, which aims to find the maximum biclique containing a specific query vertex in real-time. Apart from online computation algorithms, we explore index-based approaches and propose the PMBC-Index. With the PMBC-Index, the query algorithm is up to five orders of magnitude faster than the baseline algorithms. Furthermore, effective pruning strategies and parallelization techniques are devised to support efficient index construction. Extensive experiments on 10 real-world graphs validate both the effectiveness and the efficiency of our proposed techniques.**

I. Introduction

Bipartite graphs are naturally used to model relationships between two different types of entities, such as user-product [1], author-paper [2] and user-page [3]. On bipartite graphs, a maximum biclique [4] is the complete bipartite subgraph (biclique) with the maximum number of edges. It has been demonstrated to be useful in many applications such as anomaly detection [3], [5], gene expression analysis [6]–[8] and social recommendation [5], [7], [9]. Consequently, given a bipartite graph $G(V{=}(U, L), E)$, maximum biclique search (i.e., finding the maximum biclique C^* in G) is a popular research problem [5], [10]–[13]. Notably, it has been studied by the Alibaba Group and the techniques has been utilized on their anomaly detection system recently [5]. In addition, to provide the users with more flexibility to control the size of each layer of the biclique or avoid returning a too skewed biclique, size constraints τ_U and τ_L are imposed on each layer of C^* (i.e., $|U(C^*)| \geq \tau_U$ and $|L(C^*)| \geq \tau_L$) in maximum biclique search [5]. However, maximum biclique search only finds the biclique with the globally maximum size, while fast microscopic (personalized) analysis is required in many real-world scenarios. For instance, when a suspected user is identified in a user-product network, it is important to find the anomalous group of this suspected user and report the group of users to human experts for further investigation. In such cases, the result returned by maximum biclique search may not include the suspected user and the user's distinctive anomalous group. Moreover, as pointed out by the *Seattle Report* [14], *"high latency reduces the rate at which users make observations"*. Thus, the response efficiency when handling such cases is also critical.

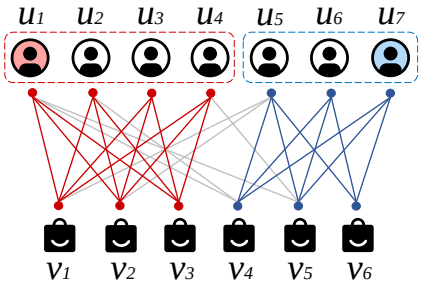


Fig. 1: A user-product network

To fill the above-mentioned research gap, for the first time, we study the efficient *personalized maximum biclique search* problem, which aims to find the maximum biclique containing a query vertex in real time. Specifically, given a bipartite graph G, a query vertex q, and size constraints τ_U and τ_L, we aim to find the personalized maximum biclique $C^q_{\tau_U,\tau_L}$ that satisfies $q \in C^q_{\tau_U,\tau_L}$, $|U(C^q_{\tau_U,\tau_L})| \geq \tau_U$ and $|L(C^q_{\tau_U,\tau_L})| \geq \tau_L$, with the maximum number of edges. Consider the user-product network in Figure 1. We can find $C^{u_1}_{1,1}$ (i.e., the subgraph induced by $\{u_1, u_2, u_3, u_4, v_1, v_2, v_3\}$ in red) for the query $q = u_1, \tau_U = 1, \tau_L = 1$ and $C^{u_7}_{1,1}$ (i.e., the subgraph induced by $\{u_5, u_6, u_7, v_4, v_5, v_6\}$ in blue) for the query $q = u_7, \tau_U = 1, \tau_L = 1$.

Applications. Efficiently finding personalized maximum bicliques has many real-world applications. One typical application is anomaly detection in many scenarios including finance, insurance and online shopping networks. For instance, on E-commerce platforms like eBay, Alibaba, and Amazon, user-product bipartite networks are prevalent [1] where each edge represents an interaction between a user and a product. On these platforms, the popularity of a product is heavily influenced by its ratings, reviews and previous sales numbers [15]. This phenomenon motivates many store-owners to mislead their users by providing fake ratings, reviews or transactions. As studied in the literature, these malicious users and the products they promote often form a closely-connected group, which is very likely to be a maximum biclique [5]. In addition, nowadays, many fraud detection techniques require a query seed (i.e., a confirmed risky account) as the input [3], [16], [17], where the seeds can be obtained by user reporting or

other security mechanisms (e.g. monitoring accounts that make a large number of fake transactions within a short time period). When a seed is detected for any reason, other users in the personalized maximum biclique of this seed become highly suspicious and should be investigated by a human expert. Consider the example in Figure 1. If user u_1 is identified as a seed (i.e., a suspicious user), the system manager can perform personalized maximum biclique search and should thus check whether users u_2, u_3 and u_4 are also anomalies.

Personalized maximum biclique search can also be useful in many other application scenarios such as customized recommendation in user-movie networks (to find users with similar taste) [9] and gene expression analysis in gene-condition bipartite networks (to identify the group of genes that exhibit similar expression patterns) [6], [8].

Challenges. Although there exist a broad spectrum of applications, the problem itself is very challenging. Since the maximum biclique search problem is NP-hard [5], the personalized maximum biclique search problem is naturally also NP-hard. Given query parameters q, τ_U, and τ_L, it is observed that the size of q's personalized maximum biclique in G is equal to the size of the maximum biclique in $\mathcal{H}_q$, where $\mathcal{H}_q$ is the subgraph induced by the one-hop and two-hop neighbors of q. Based on this fact, an online computation algorithm PMBC-OL can be designed which first shrinks the search space by obtaining $\mathcal{H}_q$. Then, the state-of-the-art algorithm [5] for maximum biclique search is performed on $\mathcal{H}_q$ to find the result. Although PMBC-OL can be finished in reasonable time in many cases, this totally online computation approach cannot satisfy the real-time requirements in many scenarios where queries are frequently posted and responses are quickly needed. As a result, we resort to index-based solutions to solve the personalized maximum biclique search problem in a more efficient way. In this problem, a straightforward index can be built by enumerating all q, τ_U, τ_L combinations and obtaining all the personalized maximum bicliques using PMBC-OL. This idea is clearly impractical since for a query vertex $q \in U(G)$, the valid τ_U and τ_L values can reach $deg(q)$ and $\max_{u \in N(q)} deg(u)$, respectively. Here $deg(q)$ denotes the degree of a vertex q, and $N(q)$ denotes the neighbor set of q. Thus, it is cost-prohibitive to compute the results of all combinations of q, τ_U, and, τ_L values. To make the ideas of indexing practical, we need to address the following main challenges:

- *Challenge 1.* How to design a size-bounded index which allows both efficient construction and query processing.
- *Challenge 2.* How to further reduce the indexing time via cost-sharing and optimization techniques.

Our approaches. To address Challenge 1, we propose the PMBC-Index by refining unique personalized maximum bicliques. The fundamental insight underlying the PMBC-Index is that each unique personalized maximum biclique of a vertex v can cover the result of multiple query combinations. Based on this observation, the PMBC-Index can be built by taking the coverage scope of each personalized maximum biclique into consideration and identifying the critical τ_U, τ_L combinations (which need to be computed with PMBC-OL) for each vertex. In this manner, it can be built without examining all combinations of q, τ_U, and τ_L values. Such an index can also support efficient query processing with a bounded size.

To address Challenge 2, we propose further optimizations for the PMBC-Index construction. Firstly, we explore cost-sharing among the construction process of different vertices based on the fact that queries for different vertices can lead to the same personalized maximum biclique. Secondly, since the construction process need to invoke PMBC-OL to compute each personalized maximum biclique, we propose upper bounding techniques to further accelerate PMBC-OL. Additionally, by leveraging the advantages of multi-core architecture, shared-memory parallelization techniques are devised to achieve a significant speedup.

Contributions. Our principal contributions are as follows:

- *The First Work to Study Personalized Maximum Biclique Search.* To the extent of our knowledge, this is the first work to study this important problem on bipartite graphs. Although the problem is NP-hard, we aim to devise solutions to efficiently answer a user query in practice.
- *A Light-weight Index Structure with Bounded Index Size.* We propose a light-weight PMBC-Index which allows for both efficient construction and query processing. As evaluated in our experiments, our index-based algorithm is up to five orders of magnitude faster than the baseline algorithms. Additionally, we theoretically prove that the index-based query can be answered within $O(deg(q)+|C^q_{\tau_U,\tau_L}|)$ time and the index size is bounded by $O(\sum_{q \in V(G)} deg(q) \cdot (deg(q) + \max_{u \in N(q)} deg(u)))$, with it being much smaller in practice.
- *Effective Techniques for Efficient Index Construction.* We study how to efficiently construct the PMBC-Index by exploring the relations of results with different combinations of parameters. Effective optimization techniques are proposed by exploring cost-sharing among vertices and upper bounds. In addition, a significant speedup ratio can be achieved by utilizing our parallelization techniques.
- *Extensive Empirical Studies on Real-world Graphs.* We conduct comprehensive experimental evaluations on 10 real-world bipartite graphs. Experimental results validate both the effectiveness and efficiency of our query algorithms and indexing techniques.

Organization. The rest of the paper is organized as follows. Section II reviews the related work. Section III presents the problem definition. Section IV introduces the background and baselines. Section V presents the PMBC-Index for efficient personalized maximum biclique search. Section VI introduces the index construction techniques. Section VII reports the experimental results. Section VIII concludes the paper.

II. Related Work

In this section, we review the related works on finding bicliques in the literature, including maximum biclique search and maximal biclique enumeration.

Maximum Biclique Search. The maximum (edge) biclique search problem aims to find the biclique with the maximum number of edges, which is the most relevant to our problem since they both aim to maximize the number of edges in a biclique. In the literature, integer programming based methods are proposed for both bipartite [11], [12] and general graphs [18] but they are not scalable to large graphs. [10] presents a probabilistic algorithm based on subspace clustering, which can find the optimal solution with a fixed probability. Recently, a practically efficient exact algorithm is proposed in [5], which adopts a progressive bounding framework to gradually adjust the search space for the optimal solution.

Existing works also have other different ways of defining "maximum", notably the maximum balanced biclique (MBB) and maximum vertex biclique (MVB). The MVB problem aims to maximize the number of vertices in a biclique and is polynomial-time solvable via reduction to the integer linear programming problem or the maximum matching problem [4].

The MBB problem aims to find the largest biclique with an equal number of vertices in each layer which is NP-hard [4]. Due to its hardness, most solutions are heuristic algorithms [19], [20] that reduce the problem to an instance of the maximum balanced independent set problem on the complement bipartite graph by devising vertex deletion rules. Also, the local-search-based [20], [21], the swap-based [22], and the tube-search-based [23] approaches are not guaranteed to find the optimal solution. Few exact algorithms are proposed to solve the MBB problem. [24] adopts a branch-and-bound framework with upper-bound-based pruning and [25] further tightens the upper bounds. Recently, [26] presents a near polynomial-time algorithm to solve the MBB problem on large bipartite graphs. Large sparse bipartite graphs are treated as a collection of size-restricted dense subgraphs to improve the efficiency of the algorithm.

Maximal Biclique Enumeration. A biclique is maximal if it is not contained in any other bicliques. Given a bipartite graph, enumerating all maximal bicliques is a fundamental problem (hereafter denoted as MBE). Since the number of maximal bicliques can be exponential to the graph size, the MBE problem cannot be solved in polynomial time [27], [28]. Most works devoted to the MBE problem focus on exhaustive search [28]–[31]. [28] proposes a consensus algorithm (`MICA`) that initializes a set of bicliques and iteratively applies transformations to them to find the maximal bicliques. [29] uses a divide-and-conquer approach (`MineLMBC`) and exploits the size constraints to efficiently prune the non-maximal or duplicate biclique. [31] improves upon `MineLMBC` by reduction techniques and proposes the `FMBE` algorithm. [30] adopts a branch and bound framework with backtracking (`iMBEA`) to limit the search space for maximal bicliques. Another class of algorithms for MBE rely on graph inflation [32], [33], which adds edges to the bipartite graph and then applies maximal clique enumeration algorithms. There are also attempts to reduce the MBE problem to the classic frequent closed itemset mining problem [34]–[38]. In addition, [31] studies parallel algorithms for MBE and [39] proposes a change-sensitive algorithm to maintain the maximal bicliques on dynamic bipartite graphs. However, the techniques proposed in the above works cannot be directly used to solve our problem since they do not focus on finding personalized maximum bicliques.

III. Problem Definition

In this section, we formally introduce the notation and definitions. Mathematical notations used throughout this paper are summarized in Table I.

TABLE I: The summary of notations

Notation	Definition
G	a bipartite graph
$V(G)/E(G)$	the vertex/edge set of G
$U(G)/L(G)$	the upper/lower layer of G
$\|G\|$	the size of G = $\|E(G)\|$
u, v, q	a vertex in a bipartite graph
$(u, v), e$	an edge in a bipartite graph
$N(u)$	the set of neighbors of u
$deg(u)$	the degree of u
$\mathcal{H}_q$	the two-hop subgraph of q in G
C, C^*	a biclique
τ_U, τ_L	size constraints of bicliques
$C^q_{\tau_U,\tau_L}$	a personalized maximum biclique of q
$\mathcal{T}_q$	the search tree of q in the PMBC-Index
$\mathcal{N}$	a tree node in a search tree
$\mathcal{A}$	the biclique array of the PMBC-Index
n, m	the number of vertices and edges in G ($m > n$)

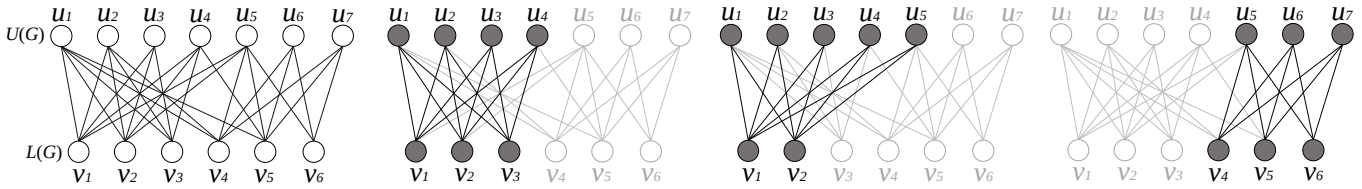
Our problem is defined over an undirected, unweighted bipartite graph $G(V{=}(U, L), E)$, where $U(G)$ denotes the set of vertices in the upper layer, $L(G)$ denotes the set of vertices in the lower layer ($U(G) \cap L(G) = \emptyset$) and $V(G) = U(G) \cup L(G)$ denotes the vertex set. $E(G)$ denotes the edge set. An edge e between two vertices u and v in G is denoted as (u, v) or (v, u). We use n and m to denote the number of vertices and edges in G respectively and we assume each vertex has at least one incident edge. The size of G is denoted as $|G| = |E(G)|$. The set of neighbors of a vertex u in G is denoted as $N(u) = \{v \in V(G) \mid (u, v) \in E(G)\}$, and the degree of u is denoted as $deg(u) = |N(u)|$.

Before formally defining the problem, we introduce the following critical concepts.

Definition 1 (Biclique)**.** *Given a bipartite graph G, a biclique C in G is a complete bipartite subgraph (i.e., for each pair of vertices $u \in U(C)$ and $v \in L(C)$, $(u, v) \in E(C)$). We also call a biclique with $|U(C)| = a$ and $|L(C)| = b$ an $(a \times b)$-biclique.*

Definition 2 (Maximum Biclique [5])**.** *Given a bipartite graph G, and a pair of integers τ_U and τ_L, a biclique C_{τ_U,τ_L} is the maximum biclique in G if $|U(C_{\tau_U,\tau_L})| \geq \tau_U$, $|L(C_{\tau_U,\tau_L})| \geq \tau_L$, and $|C_{\tau_U,\tau_L}|$ is maximized.*

In this paper, we aim to find the biclique with the maximum size containing a query vertex. In addition, size constraints can provide the users with more flexibility to control the size of each layer of the biclique [5]. Now we define the personalized maximum biclique as following.



(a) A bipartite graph G (b) $C^q_{\tau_U,\tau_L}$ with $q = u_1, \tau_U = 1, \tau_L = 1$ (c) $C^q_{\tau_U,\tau_L}$ with $q = u_1, \tau_U = 5, \tau_L = 1$ (d) $C^q_{\tau_U,\tau_L}$ with $q = u_7, \tau_U = 1, \tau_L = 1$

Fig. 2: A bipartite graph G and personalized maximum bicliques in G

Definition 3 (Personalized Maximum Biclique)**.** *Given a bipartite graph G, a query vertex q, and a pair of integers τ_U and τ_L, a biclique $C^q_{\tau_U,\tau_L}$ in G is the personalized maximum biclique if $q \in C^q_{\tau_U,\tau_L}$, $|U(C^q_{\tau_U,\tau_L})| \geq \tau_U$, $|L(C^q_{\tau_U,\tau_L})| \geq \tau_L$, and $|C^q_{\tau_U,\tau_L}|$ is maximized.*

Problem Statement. Given a bipartite graph G, a query vertex q, and a pair of integers τ_U and τ_L, the *personalized maximum biclique search problem* aims to find the personalized maximum biclique $C^q_{\tau_U,\tau_L}$ in G.

Example 1. *Considering the bipartite graph G in Figure 2(a), we show three examples of personalized maximum bicliques in Figures 2(b)-(d). As shown in Figure 2(b), when performing the search with $q = u_1$, $\tau_U = 1$ and $\tau_L = 1$ on G, the returned personalized maximum biclique $C^{u_1}_{1,1}$ is the subgraph induced by $\{u_1, u_2, u_3, u_4, v_1, v_2, v_3\}$. In addition, $C^{u_1}_{5,1}$ is the subgraph induced by $\{u_1, u_2, u_3, u_4, u_5, v_1, v_2\}$ as shown in Figure 2(c), and $C^{u_7}_{1,1}$ is the subgraph induced by $\{u_5, u_6, u_7, v_4, v_5, v_6\}$ as shown in Figure 2(d).*

IV. Background and Baseline Solutions

Background and the online algorithm PMBC-OL**.** As discussed in Section II, the maximum (edge) biclique search and maximal biclique enumeration problems are highly related to our problem. Notably, the state-of-the-art maximum (edge) biclique search algorithm in [5] follows a branch&bound paradigm, which is adapted from the maximal biclique enumeration algorithm in [30] and prunes branches not leading to the maximum biclique. Since we aim to obtain the personalized maximum biclique of a query vertex, it is intuitive to design an approach which can firstly shrink the search space according to the query vertex and then apply the state-of-the-art maximum biclique search algorithm in [5].

To propose an efficient online algorithm, we first introduce the following basic concepts.

Definition 4 (Two-hop Subgraph)**.** *Given a bipartite graph G and a query vertex q, the two-hop subgraph of q, denoted by $\mathcal{H}_q$, is the subgraph induced by the vertices in $N(q) \bigcup \forall_{v \in N(q)} N(v)$ (i.e., the one-hop and two-hop neighbors of q).*

Based on Definition 2 and Definition 4, it is immediate that the personalized maximum biclique $C^q_{\tau_U,\tau_L}$ must be a subgraph of the two-hop subgraph of q (i.e., $C^q_{\tau_U,\tau_L} \subseteq \mathcal{H}_q$). In addition, the following lemma can be derived.

Lemma 1. *Given a bipartite graph G, a query vertex q, and size constraints τ_U and τ_L, $|C^q_{\tau_U,\tau_L}| = |C^*|$, where $C^q_{\tau_U,\tau_L}$ is the personalized maximum biclique in G, and C^* denotes the maximum biclique in $\mathcal{H}_q$.*

Proof. (1) $|C^q_{\tau_U,\tau_L}| \leq |C^*|$. According to Definition 2, for any biclique C with $|U(C)| \geq \tau_U$ and $|L(C)| \geq \tau_L$ in $\mathcal{H}_q$, $|C| \leq |C^*|$. Then, $|C^q_{\tau_U,\tau_L}| \leq |C^*|$ is obvious; (2) $|C^*| \leq |C^q_{\tau_U,\tau_L}|$. According to Definition 4, it is easy to see that $q \in C^*$ (otherwise, $C^* \cup q$ will be a larger biclique). Then, based on Definition 3, we have $|C^*| \leq |C^q_{\tau_U,\tau_L}|$. Combining (1) and (2), we have $|C^q_{\tau_U,\tau_L}| = |C^*|$, and this lemma holds. □

Based on Lemma 1, computing the personalized maximum biclique of q is equivalent to compute the maximum biclique in $\mathcal{H}_q$. Thus, we propose the online computation algorithm PMBC-OL to retrieve the personalized maximum biclique as shown in Algorithm 1. Firstly, we obtain the two-hop subgraph $\mathcal{H}_q$ of q (Line 1). Then, we perform the state-of-the-art maximum biclique search algorithm [5] to obtain the maximum biclique in $\mathcal{H}_q$. A progressive bounding framework is adopted that iteratively adjusts the search space for $C^q_{\tau_U,\tau_L}$. It starts with a sub-optimal solution C^*_0 obtained using a greedy strategy. Specifically, it first initializes C^*_0 as $\{q\}$ and then iteratively adds a vertex from $V(G)$ that maximizes $|C^*_0|$. In each iteration, the search space for $C^q_{\tau_U,\tau_L}$ (bounded by τ_U^{k+1} and τ_L^{k+1}) is updated such that the algorithm is guaranteed to finish within logarithmic iterations (Lines 4 - 5). In addition, the procedure `Branch&Bound` is invoked to calculate the maximum biclique subject to size constraints τ_U^{k+1} and τ_L^{k+1}. `Branch&Bound` takes in four disjoint vertex sets (P, W, R_W, X_W) and a sub-optimal solution C^* as parameters. P and W represent the upper and lower vertices of the current largest biclique found, respectively. R_W includes all the candidate lower vertices that can be added to W, and X_W contains all the lower vertices that cannot be in the maximum biclique. The procedure `Branch&Bound` explores each vertex v^* in R_W to find possible larger bicliques which satisfy the size constraints, and the four vertex sets are updated accordingly (Lines 14 - 20). Note that v^* is added to X_W at the end of each iteration (Line 21). PMBC-OL also uses the following pruning strategies proposed in [5] to accelerate the `Branch&Bound` procedure. (1) Using one-hop (degree-based) and two-hop (wedge-based) reductions to prune invalid vertices before running `Branch&Bound` (Line 6); (2) Pruning non-maximal bicliques when running `Branch&Bound`. Note

that the time complexity of PMBC-OL is the same as the complexity of the algorithm proposed in [5].

Algorithm 1: PMBC-OL

Input: G, q, τ_U, τ_L, C_0^*: a biclique found by the greedy approach
Output: $C^q_{\tau_U,\tau_L}$

$\mathcal{H}_q \leftarrow$ the two-hop subgraph of q;
$k \leftarrow 0$; $\tau_L^0 \leftarrow$ the maximum degree of upper vertices in $\mathcal{H}_q$;
while $\tau_L^k > \tau_L$ **do**
  $\tau_U^{k+1} \leftarrow max(\lfloor \frac{|C_k^*|}{\tau_L^k} \rfloor, \tau_U)$;
  $\tau_L^{k+1} \leftarrow max(\lfloor \frac{\tau_L^k}{2} \rfloor, \tau_L)$;
  $\mathcal{H}_q^* \leftarrow$ the maximum biclique preserved subgraph of $\mathcal{H}_q$;
  $C_{k+1}^* \leftarrow$ Branch&Bound$(U(\mathcal{H}_q^*), \emptyset, L(\mathcal{H}_q^*), \emptyset, C_k^*, \tau_U^{k+1}, \tau_L^{k+1})$;
  $k \leftarrow k+1$;
return C_k^*;
Procedure Branch&Bound$(P, W, R_W, X_W, C^*, \tau_U, \tau_L)$;
if $|P| \geq \tau_U$ *and* $|W| \geq \tau_L$ *and* $|P| \times |W| > |C^*|$ **then**
  $C^* \leftarrow$ the biclique induced by P and W;
while $R_W \neq \emptyset$ **do**
  $v^* \leftarrow R_W.pop()$;
  $P' \leftarrow \{u \in P | (u, v^*) \in E(\mathcal{H}_q)\}$;
  $W' \leftarrow W \cup \{v^*\} \cup \{v \in R_W | P' \subseteq N(v, \mathcal{H}_q)\}$;
  $R'_W \leftarrow \{v \in R_W \setminus W' | \ |N(v, \mathcal{H}_q) \cap P'| \geq \tau_P\}$;
  $X'_W \leftarrow \{v \in X_W | \ |N(v, \mathcal{H}_q) \cap P'| \geq \tau_U\}$;
  if P', W', R'_W *and* X'_W *can lead to a biclique larger than* C^* **then**
    Branch&Bound$(P', W', R'_W, X'_W, C^*)$;
  $X_W \leftarrow X_W \cup \{v^*\}$;
return C^*;

Basic index-based approaches. Due to the NP-hardness of the problem, the online computation algorithm cannot efficiently answer queries which require fast responses. Thus, index-based approaches are essential for this problem since an index can be built offline to reduce the costs of online queries. Straightforwardly, we can construct a basic index structure by computing all valid personalized maximum biclique searches with PMBC-OL and storing the results. In this manner, the query can be efficiently answered by directly returning the desired biclique. However, the scalability of such index is limited in practice since the combinations of q, τ_U, τ_L can reach $O(n^3)$ in the worst case. A simple observation is that given a specific vertex q, if we change τ_L by fixing τ_U, $C^q_{\tau_U,\tau_L}$ remains the same biclique in a fixed region. Based on this observation, we can use binary search to decide the boundary of the region and skip computing some τ_U, τ_L combinations. However, it still incurs a large number of computations of q, τ_U, τ_L combinations and brings additional verification costs. As a result, our objective in this paper is to design a practical index which can be constructed without enumerating numerous q, τ_U, τ_L combinations and can support efficient query processing.

V. The PMBC-Index

In this section we propose a novel index structure, the Personalized Maximum BiClique Index (PMBC-Index) which

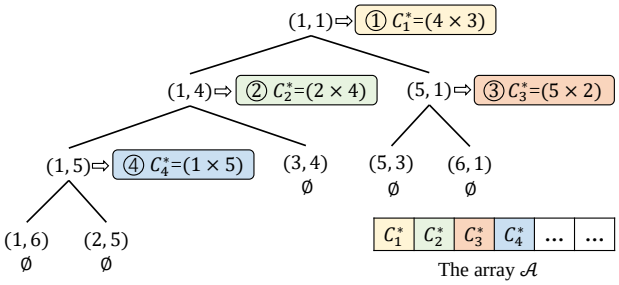


Fig. 3: The search tree $\mathcal{T}_{u_1}$ of u_1 and the array $\mathcal{A}$

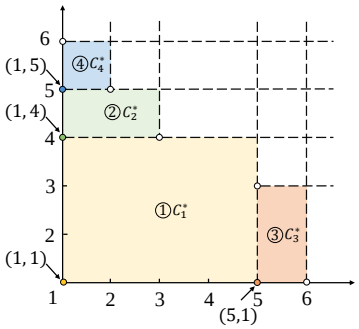


Fig. 4: Illustrating the rationale of the PMBC-Index

can enable both efficient index construction and query processing. The fundamental insight underlying the PMBC-Index is that given a vertex q, each personalized maximum biclique of q can be the result of multiple q, τ_U, τ_L combinations. By identifying the coverage scope of each personalized maximum biclique, we can build the index by only considering these critical combinations.

The structure of the PMBC-Index. The PMBC-Index consists of two parts: (1) the forest $\mathcal{T}$ which contains a set of search trees, and (2) the array $\mathcal{A}$ which contains a set of personalized maximum bicliques. In the forest $\mathcal{T}$, a search tree $\mathcal{T}_q$ is built for each vertex q . For example, Figure 3 shows the search tree $\mathcal{T}_{u_1}$ of u_1 derived from the bipartite graph G in Figure 2. Each tree node $\mathcal{N} \in \mathcal{T}_q$ contains the following information:

- two integers τ_U and τ_L;
- a pointer p_c which points to the address of $C^q_{\tau_U,\tau_L}$ in $\mathcal{A}$;
- two pointers p_l and p_r which point to $\mathcal{N}'s$ children in $\mathcal{T}_q$.

Specifically, the root node of $\mathcal{T}_q$ stores two integers $\tau_U = 1$ and $\tau_L = 1$, and the address of $C^q_{1,1}$ in $\mathcal{A}$. In addition, a tree node $\mathcal{N}' \in \mathcal{T}_q$ is a child of another node $\mathcal{N}$ if it satisfies one of the following two conditions:

$$\tau'_U = |U(C^q_{\tau_U,\tau_L})| + 1, \ \tau'_L = \tau_L; \quad (1)$$

$$\tau'_U = \tau_U, \ \tau'_L = |L(C^q_{\tau_U,\tau_L})| + 1. \quad (2)$$

Here τ'_U (τ'_L) = $\mathcal{N}'.\tau_U$ ($\mathcal{N}'.\tau_L$) and τ_U (τ_L) = $\mathcal{N}.\tau_U$ ($\mathcal{N}.\tau_L$), respectively. Note that we store the pointer p_c instead of the entire personalized maximum biclique (i.e, $C^q_{\tau_U,\tau_L}$) in a tree node. This is because multiple query vertices may share the same instance of a personalized maximum biclique, meaning significant savings in space. For example, the result of queries

$q = u_1$, $\tau_U = 1$, $\tau_L = 1$ and $q = v_1$, $\tau_U = 1$, $\tau_L = 1$ are both the (4×3)-biclique in Figure 2(b).

Example 2. *Consider the bipartite graph G in Figure 2. Figure 3 shows the search tree of u_1 and a part of the array $\mathcal{A}$. We can see that $\mathcal{T}_{u_1}$ organizes four personalized maximum bicliques of u_1 with different τ_U and τ_L. Note that $\mathcal{A}$ also contains other personalized maximum bicliques in G such as $C^{u_7}_{1,1}$ in Figure 2(d), we omit them due to the short of space.*

The Rationale of the PMBC-Index**.** In this part, we show the rationale of the PMBC-Index with the following lemmas.

Lemma 2. *Given a bipartite graph G and a query vertex q, we have $|C^q_{\tau_U,\tau_L}| \geq |C^q_{\tau'_U,\tau'_L}|$ if $\tau_U \leq \tau'_U$ and $\tau_L \leq \tau'_L$.*

Proof. We prove this lemma by contradiction. Given $\tau_U \leq \tau'_U$ and $\tau_L \leq \tau'_L$, suppose $|C^q_{\tau_U,\tau_L}| < |C^q_{\tau'_U,\tau'_L}|$. Then, according to Definition 3, $C^q_{\tau_U,\tau_L}$ should be replaced by $C^q_{\tau'_U,\tau'_L}$ since $\tau_U \leq \tau'_U$ and $\tau_L \leq \tau'_L$. Therefore, this lemma holds. □

Lemma 3. *Given a bipartite graph G and a query vertex q, $|C^q_{\tau_U,\tau_L}| = |C^q_{\tau'_U,\tau'_L}|$ if $\tau_U \leq \tau'_U \leq |U(C^q_{\tau_U,\tau_L})|$ and $\tau_L \leq \tau'_L \leq |L(C^q_{\tau_U,\tau_L})|$.*

Proof. This lemma directly follows Lemma 2. □

Lemma 4. *Given a bipartite graph G, a query vertex q, and two pairs of integers (τ_U, τ_L), (τ'_U, τ'_L) where $\tau_U \leq \tau'_U$ and $\tau_L \leq \tau'_L$. If $|C^q_{\tau_U,\tau_L}| \neq |C^q_{\tau'_U,\tau'_L}|$, one of the following two conditions must hold: (1) $\tau'_U \geq |U(C^q_{\tau_U,\tau_L})|+1$ and $\tau'_L \geq \tau_L$; or (2) $\tau'_U \geq \tau_U$ and $\tau'_L \geq |L(C^q_{\tau_U,\tau_L})|+1$.*

Proof. This lemma follows Lemma 2 and Lemma 3. □

Lemma 2 and Lemma 3 illustrate the intuition behind PMBC-Index as we know that the size of personalized maximum bicliques can only increase as the parameters τ_U and τ_L decrease and multiple (τ_U, τ_L) combinations may lead to the same personalized maximum biclique. For example, consider the search tree of u_1 in Figure 3. We illustrate the idea of our indexing techniques in Figure 4. In this example, the personalized maximum biclique of $q = u_1, \tau_U = 1, \tau_L = 1$ is a (4×3)-biclique. Thus, for the query vertex u_1, the answer of any query size constraints (τ'_U, τ'_L) dominated by $(4, 3)$ (i.e., $\tau'_U \leq 4$ and $\tau'_L \leq 3$) is the (4×3)-biclique C^*_1 as illustrated in Figure 4. On the other hand, there must exist no personalized maximum bicliques of u_1 with size large than 4×3 since $\tau_U = 1, \tau_L = 1$. As shown in Figure 4, to cover the rest of the solution spaces where there may exist a new personalized maximum biclique, we continue searching with parameters $(\tau_U = 5, \tau_L = 1)$ and $(\tau_U = 1, \tau_L = 4)$ according to Lemma 4. Finally, all the valid solution spaces are covered by personalized maximum bicliques as shown in Figure 4.

Queries on the PMBC-Index**.** Given a query vertex q, size constraints τ_U and τ_L, Algorithm 2 illustrates the query process of the personalized maximum biclique (i.e., $C^q_{\tau_U,\tau_L}$) based on the PMBC-Index. We start from the root of $\mathcal{T}_q$ and use $\mathcal{N}$ to denote the current processing tree node (Line 2). Utilizing the address stored in $\mathcal{N}.p_c$, we verify the size of the personalized maximum biclique $C^* = \mathcal{A}[\mathcal{N}.p_c]$. If it satisfies that $|U(C^*)| \geq \tau_U$ and $|L(C^*)| \geq \tau_L$, we return C^* as the result. Note that we only need to use the size of C^* to do the verification and retrieve the entire personalized maximum biclique only if it satisfies the size constraints. Otherwise, we process the child $\mathcal{N}'$ of $\mathcal{N}$ which satisfies $\mathcal{N}'.\tau_U \leq \tau_U$ and $\mathcal{N}'.\tau_L \leq \tau_L$ to find the result in the next level of the search tree. We return empty if no valid result can be found from the search tree.

Algorithm 2: PMBC-IQ

```
Input: G, q, τ_U, τ_L, T, A
Output: C^q_{τ_U,τ_L}
1  C^q_{τ_U,τ_L} ← ∅;
2  N ← T_q.root;
3  while N is not null do
4      if |U(A[N.p_c])| ≥ τ_U and |L(A[N.p_c])| ≥ τ_L then
5          C^q_{τ_U,τ_L} ← A[N.p_c];
6          return;
7      foreach child N' of N in T_q do
8          if N'.τ_U ≤ τ_U and N'.τ_L ≤ τ_L then
9              N ← N';
10             break;
11 return C^q_{τ_U,τ_L};
```

Example 3. *Consider the bipartite graph G in Figure 2. We show how to find the personalized maximum biclique for $q = u_1, \tau_U = 2, \tau_L = 4$ based on the search tree $\mathcal{T}_{u_1}$ of u_1 in Figure 3. Firstly, we obtain the root node of $\mathcal{T}_{u_1}$ and verify the size of $C^{u_1}_{1,1}$ (which is denoted as C^*_1 in Figure 3). Since $|L(C^*_1)| = 3 < \tau_L = 4$, C^*_1 cannot be the result. Then, we check the children of the root node and continue searching the child with $\mathcal{N}'.\tau_U = 1, \mathcal{N}'.\tau_L = 4$. We find that C^*_2 is a (2×4)-biclique which satisfies the query constraints. Thus, we return C^*_2 as the result.*

Analysis of the PMBC-Index**.** In this part, we provide theoretical analysis of the PMBC-Index.

Theorem 1. *Given a bipartite graph G, a query vertex q, and a pair of integers τ_U and τ_L, Algorithm 2 correctly computes $C^q_{\tau_U,\tau_L}$ based on the* PMBC-Index *of G.*

Proof. According to Lemma 2, in each level of the search tree $\mathcal{T}_q$, the size of the personalized maximum biclique in the next level is no larger than the size of the personalized maximum biclique in the current level. Thus, we can return the first personalized maximum biclique found in $\mathcal{T}_q$ that satisfies the size constraints. In addition, according to Lemma 4, each valid combination of τ_U and τ_L which can lead to a personalized maximum biclique is covered by the PMBC-Index. Therefore, this theorem holds. □

Lemma 5. *Given a bipartite graph G and a vertex q, the number of tree nodes in $\mathcal{T}_q$ is $O(deg(q))$.*

Proof. In the search tree $\mathcal{T}_q$ of q, each tree node contains two distinctive size constraints τ_U and τ_L. Thus we only need to prove that there exists at most $O(deg(q))$ combinations of

(τ_U, τ_L) in $\mathcal{T}_q$. According to the structure of the PMBC-Index, the (τ_U, τ_L) combination in non-leaf nodes should lead to a distinctive personalized maximum biclique of q. Without loss of generality, we suppose q is an upper layer vertex (i.e., $q \in U(G)$). Since q has $deg(q)$ neighbors, according to the definition of the personalized maximum biclique, there exist at most $deg(q)$ distinctive personalized maximum bicliques where the number of their lower layer vertices is chosen from $[1, deg(q)]$. Thus, the number of non-leaf nodes is $O(deg(q))$. In addition, since each non-leaf node has at most 2 children, the number of leaf nodes is also bounded by $O(deg(q))$. Therefore, the number of tree nodes in $\mathcal{T}_q$ is $O(deg(q))$, and this lemma holds. □

Theorem 2. *Given a bipartite graph G and a query vertex q, Algorithm 2 computes $C^q_{\tau_U,\tau_L}$ in $O(deg(q) + |C^q_{\tau_U,\tau_L}|)$ time.*

Proof. The first term of the time complexity is for traversing the search tree, and the second term is for retrieving the personalized maximum biclique from the tree node. As the second term is obvious we only need to prove the first term. In each level of the search tree $\mathcal{T}_q$, we only need to choose one child to proceed based on Lemma 4, and each tree node will not be processed more than one time. Thus, according to Lemma 5, the total number of tree nodes we need to traverse is bounded by $O(deg(q))$. In addition, according to Algorithm 2, we only need $O(1)$ time to examine the size constraints in each tree node and retrieve the entire personalized maximum biclique only once. Therefore, this theorem holds. □

Theorem 3. *Given a bipartite graph G, storing the* PMBC-Index *$\mathcal{T}$ of G needs $O(\sum_{q\in V(G)} deg(q) \cdot |\mathcal{H}_q|)$ space. Here $\mathcal{H}_q$ is the two-hop subgraph of q and $O(|\mathcal{H}_q|) = O(deg(q) + \max_{v\in N(q)} deg(v))$.*

Proof. According to Definition 3 and Definition 4, $C^q_{\tau_U,\tau_L} \subseteq \mathcal{H}_q$. Thus, the size of each personalized maximum biclique of q is bounded by $O(|\mathcal{H}_q|)$. Since each tree node of the search tree $\mathcal{T}_q$ contains at most one personalized maximum biclique, according to Lemma 5, this theorem holds. □

VI. Construction of the PMBC-Index

In this section, we present our techniques for constructing the PMBC-Index.

A. The Main Framework

Given a bipartite graph G, the main process of constructing the PMBC-Index is to build the search tree for each vertex q (i.e., $\mathcal{T}_q$) along with the array $\mathcal{A}$ which stores the corresponding personalized maximum bicliques. When computing the tree nodes of $\mathcal{T}_q$, we need to retrieve personalized maximum bicliques under different thresholds τ_U and τ_L. Given two nodes $\mathcal{N}$ and $\mathcal{N}' \in \mathcal{T}_q$, we suppose $\mathcal{N}'$ is a child of $\mathcal{N}$. In addition, we denote $\mathcal{N}'.\tau_U$ ($\mathcal{N}'.\tau_L$) as τ'_U (τ'_L) and $\mathcal{N}.\tau_U$ ($\mathcal{N}.\tau_L$) as τ_U (τ_L), respectively. From Lemma 2, we can get that $|C^q_{\tau'_U,\tau'_L}| \leq |C^q_{\tau_U,\tau_L}|$. In other words, if we traverse the search tree from the root to the leaves, the size of personalized maximum bicliques (of the visited nodes) is non-increasing.

Algorithm 3: PMBC-IC

```
Input: G
Output: T, A
1  foreach vertex q ∈ V(G) do
2      create the root node N on T_q;
3      N.τ_U ← 1; N.τ_L ← 1;
4      Q ← N;
5      while Q is not empty do
6          N ← Q.pop();
7          C* ←a biclique found by the greedy approach;
8          PMBC-OL (G, q, N.τ_U, N.τ_L, C*) with Lemma 6
           applied;
9          if C* is not null then
10             if C* ∉ A then
11                 store C* into A;
12             N.p_c ← the address of C* in A;
13             create two children N' and N'' of N on T_q;
14             N.p_l ← the address of N';
15             N.p_r ← the address of N'';
16             N'.τ_U ← |U(C*)| + 1; N'.τ_L ← N.τ_L;
17             N''.τ_U ← N.τ_U; N''.τ_L ← |L(C*)| + 1;
18             push N' and N'' into Q if the size constraints
               are satisfied;
19 return T, A;
```

This observation uncovers an upper bound of $|C^q_{\tau'_U,\tau'_L}|$ (of a child node) from $|C^q_{\tau_U,\tau_L}|$ (of its parent). Note that for the tree node $\mathcal{N}$ and its child $\mathcal{N}'$, we either have $\tau'_U = |U(C^q_{\tau_U,\tau_L})|+1$, or $\tau'_L = |L(C^q_{\tau_U,\tau_L})| + 1$ according to Lemma 4. Thus, the following lemma also holds.

Lemma 6. *If $\tau'_U = |U(C^q_{\tau_U,\tau_L})| + 1$, then $|L(C^q_{\tau'_U,\tau'_L})| < |L(C^q_{\tau_U,\tau_L})|$; If $\tau'_L = |L(C^q_{\tau_U,\tau_L})| + 1$, then $|U(C^q_{\tau'_U,\tau'_L})| < |U(C^q_{\tau_U,\tau_L})|$.*

Proof. This lemma directly follows Lemma 2. □

The details of the index construction algorithm PMBC-IC are shown in Algorithm 3. For each vertex $q \in V(G)$, we first create the root $\mathcal{N}$ of $\mathcal{T}_q$ and set both $\mathcal{N}.\tau_U$ and $\mathcal{N}.\tau_L$ as 1 (lines 1-3). Then, we obtain the personalized maximum biclique C^* using PMBC-OL. Note that the constraints in Lemma 6 are applied when performing PMBC-OL for non-root nodes. If a non-empty result is found and $C^* \notin \mathcal{A}$, we store C^* into $\mathcal{A}$. After that, we create two children of the current processing node if they satisfy the size constraints (according to Lemma 2). We repeat this process until no valid tree node can be created.

Example 4. *Consider the bipartite graph G in Figure 2. We illustrate how to create the search tree of u_1 and (a part of) the array $\mathcal{A}$ as shown in Figure 3. Firstly, we create the root node $\mathcal{N}$ of $\mathcal{T}_{u_1}$. We assign $\mathcal{N}.\tau_U = 1$ and $\mathcal{N}.\tau_L = 1$. Then, we use* PMBC-OL *to find the personalized maximum biclique under $q = u_1, \tau_U = 1, \tau_L = 1$. We push the result (i.e., the 4×3-biclique C^*_1) into $\mathcal{A}$ and assign $\mathcal{N}.p_c$ as the address of C^*_1 in $\mathcal{A}$. Then, according to Lemma 4, we create the children $\mathcal{N}'$ and $\mathcal{N}''$. We then process each child and store their personalized maximum bicliques (C^*_2 and C^*_3) into $\mathcal{A}$. By iteratively running this process, the search tree of u_1 and the array $\mathcal{A}$ are built.*

Theorem 4. *Given a bipartite graph G, Algorithm 3 correctly builds the* PMBC-Index *of G.*

Proof. According to Algorithm 3 lines 1-4, each search tree is processed and starts from the root node $\mathcal{N}$ with $\mathcal{N}.\tau_U = 1$ and $\mathcal{N}.\tau_L = 1$. For each tree node, its personalized maximum biclique is correctly computed by PMBC-OL according to Lemma 1. Then, based on Lemma 4, each child node is correctly generated and each valid (τ_U, τ_L) combination of the child node will be examined during the index construction process. Therefore, Algorithm 3 builds the PMBC-Index of G correctly. □

Theorem 5. *Given a bipartite graph G, the time complexity of Algorithm 3 is $O(\sum_{q \in V(G)} deg(q) \cdot TC(\text{PMBC-OL}))$, where* TC(PMBC-OL*) *denotes the time complexity of* PMBC-OL.

Proof. In Algorithm 3, we run PMBC-OL to compute the personalized maximum biclique for each tree node in the search tree $\mathcal{T}_q$. According to Lemma 5, the total number of tree nodes in $\mathcal{T}_q$ is bounded by $O(deg(q))$. Therefore, this theorem holds. □

B. Cost-sharing across Different Search Trees

Utilizing the PMBC-IC algorithm, the PMBC-Index can be correctly constructed. When we examine the algorithm, we observe that in the process of computing a search tree (of a vertex), its tree nodes may contain results from previously built search trees. Naturally, we wish to use these previously computed results to accelerate the construction of the current search tree. We thus propose a light-weight auxiliary structure to achieve this goal.

Preserving skyline maximal bicliques. Firstly, we give the following lemma.

Lemma 7. *Given two query vertices u and v, and a pair of integers τ_U and τ_L, if a personalized maximum biclique C^* of u contains v and satisfies $|U(C^*)| \geq \tau_U$ and $|L(C^*)| \geq \tau_L$, then the personalized maximum biclique $C^v_{\tau_U,\tau_L}$ of the query v, τ_U and τ_L must satisfy $|C^v_{\tau_U,\tau_L}| \geq |C^*|$ (i.e., $|C^*|$ is the lower bound of $|C^v_{\tau_U,\tau_L}|$).*

Proof. Suppose we have a $|C^v_{\tau_U,\tau_L}| < |C^*|$. However C^* contains v which contradicts the fact that $C^v_{\tau_U,\tau_L}$ is the personalized maximum biclique of v. Thus, $|C^v_{\tau_U,\tau_L}| \geq |C^*|$, and this lemma holds. □

Based on Lemma 7, when constructing a search tree, it is possible to utilize the previous results by storing the computed personalized maximum bicliques. Specifically, we build an inverted index $\mathcal{S}[v]$ for each vertex v, which stores a set of IDs of bicliques containing the vertex v. To effectively achieve this goal, $\mathcal{S}[v]$ only stores the skyline maximal bicliques of v, which is defined as follows.

Definition 5 (Skyline Maximal Biclique)**.** *Given a bipartite graph G, a vertex v and the auxiliary index $\mathcal{S}$, a biclique C is a skyline maximal biclique of v if there does not exist any biclique $C' \in \mathcal{S}[v]$ s.t. $|U(C')| \geq |U(C)|$ and $|L(C')| \geq |L(C)|$.*

Using the inverted index $\mathcal{S}$, we can provide a lower bound result C^* from the previous computed bicliques before running PMBC-OL. In addition, we do not need to insert C^* into $\mathcal{A}$ again if C^* is the final result after running PMBC-OL. Note that the auxiliary index $\mathcal{S}$ is light-weight, as illustrated in the following lemma.

Lemma 8. *Given a bipartite graph G and a vertex $v \in V(G)$, $|\mathcal{S}[v]| \leq deg(v)$, where $|\mathcal{S}[v]|$ is the number of elements in $\mathcal{S}[v]$.*

Proof. Without loss of generality, we suppose $v \in U(G)$. Since each biclique $C \in \mathcal{S}[v]$ is a skyline maximal biclique containing v, $|L(C)| \in [1, deg(v)]$. According to Definition 5, there does not exist a biclique $C' \in \mathcal{S}[v]$ with $|U(C')| \geq |U(C)|$ and $|L(C')| \geq |L(C)|$. Thus, for each $|L(C)|$ value, there can only exist one $|U(C)|$ value accordingly. Therefore, the number of elements in the auxiliary index $\mathcal{S}$ is at most $deg(v)$, and this lemma holds. □

Algorithm 4: PMBC-IC*

```
   Input: G
   Output: T, A
1  initialize S[v] as empty for each v ∈ V(G);
2  foreach vertex q ∈ V(G) do
3      run Algorithm 3 Lines 2 - 4;
4      while Q is not empty do
5          N ← Q.pop();
6          C* ←a biclique found by the greedy approach;
7          if there exists C ∈ S[q] s.t. |U(C)| ≥ N.τ_U and
             |L(C)| ≥ N.τ_L and |C| ≥ |C*|; then
8              C* ← C;
9          PMBC-OL (G, q, N.τ_U, N.τ_L, C*) with Lemma 6
           applied;
10         if C* is not null then
11             if C* ∉ A then
12                 store C* into A;
13                 foreach vertex v ∈ C* do
14                     S[v].update(C*);
15             run Algorithm 3 Lines 12 - 18;
16 return T, A;
```

The PMBC-IC* algorithm. Utilizing the above observations, the details of the PMBC-IC* algorithm are shown in Algorithm 4. We first initialize $\mathcal{S}[v]$ as empty for each $v \in V(G)$. Then, for each vertex $q \in V(G)$, we create the root of $\mathcal{T}_q$ and set both τ_U and τ_L as 1. Before running PMBC-OL, we search $\mathcal{S}[q]$ to obtain a lower bound result C^*. If C^* is the final result, we link the address of C^* to $\mathcal{N}.p_c$, and C^* does not need to be inserted into $\mathcal{A}$. Otherwise, we store C^* into $\mathcal{A}$ and update $\mathcal{S}$ according to Definition 5. After that, we create two children of the current processing node and keep this process until no valid tree node can be created. The time complexity of PMBC-IC* is the same as the PMBC-IC algorithm since it only use the additional light-weighted inverted index to provide cost-sharing across different search trees.

Algorithm 5: PMBC-OL*

Input: G, q, τ_U, τ_L, C_0^*: a biclique found by the greedy approach
Output: $C^*_{\tau_U,\tau_L}$

// the following three lines can be completed offline
compute z_v for each vertex $v \in V(G)$; //Lemma 9
compute $\overleftarrow{z_u}[i] = \max_{\alpha\in[1,i]} \alpha \cdot s_a(u,\alpha)$ for each vertex $u \in U(G)$ and $i \in [1, deg(u)]$;
compute $\overrightarrow{z_v}[i] = \max_{\beta\in[i,deg(u)]} \beta \cdot s_b(u,\beta)$ for each vertex $v \in L(G)$ and $i \in [1, deg(v)]$;
$\mathcal{H}_q \leftarrow$ the two-hop subgraph of q;
$k \leftarrow 0$; $\tau_L^0 \leftarrow$ the maximum degree of upper vertices in $\mathcal{H}_q$;
while $\tau_L^k > \tau_L$ **do**
  $\tau_U^{k+1} \leftarrow max(\lfloor \frac{|C_k^*|}{\tau_L^k} \rfloor, \tau_U)$;
  $\tau_L^{k+1} \leftarrow max(\lfloor \frac{\tau_L^k}{2} \rfloor, \tau_L)$;
  prune the vertex $v \in V(\mathcal{H}_q)$ with $z_v \le |C_k^*|$; //Lemma 9
  $\mathcal{H}_q^* \leftarrow$ the maximum biclique preserved subgraph of $\mathcal{H}_q$;
  $C_{k+1}^* \leftarrow$ `Branch&Bound`$(U(\mathcal{H}_q^*), \emptyset, L(\mathcal{H}_q^*), \emptyset, C_k^*, \tau_U^{k+1}, \tau_L^{k+1})$;
  $k \leftarrow k+1$;
return C_k^*;
Procedure `Branch&Bound`(P, W, R_W, X_W, C^*, τ_U, τ_L);
run Algorithm 1 Lines 12 - 22, skip v^* if $\overrightarrow{z_{v^*}}[|W|] \le |C^*|$ after Line 15, prune the vertex $u \in P'$ with $\overleftarrow{z_u}[|P|] \le |C^*|$ in Line 16;
return C^*;

C. Upper Bounding Techniques for PMBC-OL

Since the index construction algorithms invoke PMBC-OL to compute the personalized maximum bicliques, in this subsection, we explore how to further accelerate PMBC-OL. Note that PMBC-OL actually follows a branch&bound framework. Under this framework, the bounding ability significantly affects the performance of the algorithm. Thus, we propose upper bounding techniques for improving PMBC-OL. We first present the definition of (α, β)-core as follows.

Definition 6 ((α, β)-core)**.** *Given a bipartite graph G and degree constraints α and β, a subgraph $R_{\alpha,\beta}$ is an (α,β)-core of G if (1) $deg(u, R_{\alpha,\beta}) \ge \alpha$ for each $u \in U(R_{\alpha,\beta})$ and $deg(v, R_{\alpha,\beta}) \ge \beta$ for each $v \in L(R_{\alpha,\beta})$; (2) $R_{\alpha,\beta}$ is maximal (i.e., any supergraph $G' \supset R_{\alpha,\beta}$ is not an (α, β)-core).*

Based on the degree constraints of (α, β)-core, obviously, a ($\beta \times \alpha$)-biclique is also a subgraph of the (α, β)-core. This observation is already used to prune some invalid vertices in PMBC-OL (i.e., the one-hop reduction introduced in [5]). Here we give the definition of α-/β-offsets and further derive a lemma based on the definition.

Definition 7 (α-/β-offset)**.** *Given a vertex $u \in U(G)$ and an α value, its α-offset, denoted as $s_a(u,\alpha)$, is the maximal β value where u can be contained in an (α, β)-core. Symmetrically, the β-offset $s_b(v,\beta)$ of $v \in L(G)$ is the maximal α value where v can be contained in an (α, β)-core.*

Based on Definition 7, we define the upper bounds z_u and z_v as follows. Given a vertex $u \in U(G)$, $z_u = \max_{\alpha\in[1,deg(u)]} \alpha \cdot s_a(u,\alpha)$. Similarly, given a vertex $v \in L(G)$, $z_v = \max_{\beta\in[1,deg(v)]} \beta \cdot s_b(u,\beta)$. The following lemma can then be derived.

Lemma 9. *Given a vertex $u \in U(G)$, for any biclique C containing u, $|C| \le z_u$. Similarly, given a vertex $v \in L(G)$, for any biclique C containing v, $|C| \le z_v$.*

Proof. Given a vertex $u \in U(G)$, suppose it is contained in a biclique C with size larger than z_u (i.e., $|U(C)| + |L(C)| > z_u$). According to Definition 1 and Definition 6, a ($\beta \times \alpha$)-biclique is a subgraph of the (α, β)-core. Additionally, according to Definition 7, the maximum ($\alpha \times \beta$) value of the (α, β)-core (and subsequent biclique) where u can exist is z_u. There thus exists a contradiction, and this lemma holds. □

Example 5. *Consider the bipartite graph G in Figure 2. The upper bounds (z_u for each vertex $u \in U(G)$ and z_v for each vertex $v \in L(G)$) are shown in Figure 5. Taking u_5 and v_5 as an example, we illustrate how to compute the upper bounds. The α-offset for u_5 regarding each α is shown in the right side of Figure 2. Correspondingly, $z_{u_5} = 12$ which is the maximal $\alpha \times \alpha$-offset value of u_5. Similarly, $z_{v_5} = 12$ which is the maximal $\beta \times \beta$-offset value of v_5.*

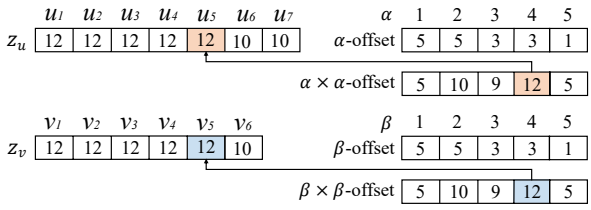


Fig. 5: Illustrating the upper bound

Lemma 9 uncovers the size constraints provided by the (α, β)-core model that can be used to further prune invalid vertices. Given a sub-optimal solution C^*, if $z_u \le |C^*|$ (or $z_v \le |C^*|$), then $u \in U(G)$ (or $v \in L(G)$) does not exist in a biclique with size larger than $|C^*|$ and can be pruned. For instance, suppose we perform the query with $q = u_1$, $\tau_U = 1$, and $\tau_L = 1$ on the bipartite graph G in Figure 2. If $|C^*| = 10$, then vertices u_6, u_7, and v_6 can be pruned in further iterations since they cannot exist in a larger biclique. In addition, we observe that the `Branch&Bound` procedure iteratively adds one lower vertex $v^* \in L(G)$ to W as shown in Algorithm 1 Line 14 (i.e., $|W|$ increments in each iteration). Thus, if v^* cannot belong to a larger biclique with more than $|W|$ lower vertices, we can skip adding v^* into W'. To utilize this observation, for each vertex $v \in L(G)$ and $i \in [1, deg(v)]$, we can pre-compute $\overrightarrow{z_v}[i] = \max_{\beta\in[i,deg(v)]} \beta \cdot s_b(v,\beta)$. Then, we can prune the vertex v^* if $\overrightarrow{z_{v^*}}[|W|] \le |C^*|$. In addition, the number of upper vertices ($|P|$) in the current expanding biclique is non-increasing as the number of iterations increases. Thus, we can also compute $\overleftarrow{z_u}[i] = \max_{\alpha\in[1,i]} \alpha \cdot s_a(u,\alpha)$ for each vertex $u \in U(G)$ and $i \in [1, deg(u)]$ to prune the upper vertex u with $\overleftarrow{z_u}[|P|] \le |C^*|$. Note that the α-/β-

offsets can be pre-computed in $O(\delta \cdot m)$ time by using the decomposition algorithm proposed in [40]. The arrays derived from α-/β-offsets such as z_v for each vertex v can be computed in linear time. Here, δ is the maximal value where the (δ, δ)-core is non-empty in a bipartite graph G, where δ is bounded by $\sqrt{m}$.

Algorithm 5 details our improved baseline PMBC-OL* using the optimizations discussed above. Firstly, we compute the auxiliary structures (which can be conducted offline). Then, before running the `Branch&Bound` procedure, we prune each vertex $v \in \mathcal{H}_q$ where $z_v \leq |C_k^*|$ (Line 10). In addition, utilizing the arrays $\overleftarrow{z}$ and $\overrightarrow{z}$, invalid upper and lower vertices are also pruned when running the `Branch&Bound` procedure. Note that in PMBC-OL*, we skip checking whether the current biclique is maximal since a non-maximal biclique is easily discarded by the above (α, β)-core-based pruning strategies.

D. Shared-memory Parallelization

In PMBC-IC*, the most time-consuming part is building the search trees, which is actually an independent procedure for each vertex in $V(G)$. In other words, PMBC-IC* does not need to update any data structures shared with other vertices when building the search trees. This observation motivates us to speed up Algorithm 4 with shared-memory parallelization (i.e., handling different vertices simultaneously with multiple threads). Since the array of bicliques $\mathcal{A}$ and the inverted index $\mathcal{S}$ are shared by multiple vertices (i.e., threads), write conflicts can occur when updating them. In other words, if several threads try to add new elements into $\mathcal{A}$ or $\mathcal{S}$ simultaneously, the size of the array will become ambiguous without handling the conflicts. To solve this issue, we first give the following lemma to estimate the sizes of these index structures.

Lemma 10. *Given a bipartite graph G, the total number of personalized maximum bicliques in G is at most $\sum_{v \in V(G)} deg(v)$.*

Proof. This lemma can be proved similarly as Lemma 5. □

Based on the above lemma, the number of elements in $\mathcal{A}$ is at most $\sum_{v \in V(G)} deg(v)$. Thus, the space usage of $\mathcal{A}$ is at most $\sum_{v \in V(G)} deg(v) \cdot \mathsf{size}(C^*)$, where $\mathsf{size}(C^*)$ denotes the maximal size of a personalized maximum biclique instance which can be computed according to Theorem 3. In addition, the space usage of $\mathcal{S}[q]$ is at most $deg(q) \cdot \mathsf{size}(C^*)$ for each vertex $q \in V(G)$ (according to Lemma 8). Based on the above facts, we can allocate enough memory to these arrays before processing each vertex. After that, when adding a new element into the array $\mathcal{A}$ or $\mathcal{S}$, we only need to atomically increase the value of an integer which represents the current array size to allocate a location for this element. This atomic addition can be implemented by the atomic *fetch-and-add*[1] operation. Then, we can write the element into the allocated location of the array without conflict. In this manner, we transform the atomic element writing operations into light-weight atomic *fetch-and-add* operations. In addition, since the workload for each vertex is different, we use the dynamic scheduling strategy in OpenMP [2] to support workload balance.

We show details of the parallel index construction algorithm in Algorithm 6. We first allocate enough memory for $\mathcal{A}$ and $\mathcal{S}$ as discussed above (Lines 1-2). The values which represent the number of elements in the arrays are initialized as zero (Line 3). Then, we process each vertex $q \in V(G)$ the way same as in Algorithm 4. Before running PMBC-OL*, we search $\mathcal{S}[q]$ for a lower bound result. Note that we need to atomically obtain the size of $\mathcal{S}[q]$ to avoid conflicts. After running PMBC-OL*, we store C^* into $\mathcal{A}$ and update $\mathcal{S}$ atomically. Since Algorithm 6 does not need additional heavy data structures to support parallelization, the time complexity of Algorithm 6 (under the one core environment) is the same as Algorithm 4. Note that the techniques introduced in this subsection can also be used to parallelize PMBC-IC and the basic index discussed in Section IV in a similar way.

Algorithm 6: PMBC-IC* in parallel

```
   Input: G
   Output: T, A
1  allocate memory of size Σ_{v∈V(G)} deg(v) to A;
2  allocate memory of size deg(q) to S[q] for each q ∈ V(G);
3  z_A ← 0; z_S[q] ← 0 for each q ∈ V(G);
4  foreach q ∈ V(G) in parallel do
5      run Algorithm 3 Lines 2 - 4;
6      while Q is not empty do
7          N ← Q.pop();
8          C* ←a biclique found by the greedy approach;
9          if there exists C ∈ S[q] s.t. |U(C)| ≥ N.τ_U and
             |L(C)| ≥ N.τ_L and |C| ≥ |C*| then
10             C* ← C;
11         PMBC-OL* (G, q, N.τ_U, N.τ_L, C*) with Lemma
           6 applied;
12         if C* is not null then
13             if C* ∉ A then
14                 z_A ← z_A + 1 in atomic;
15                 store C* into A[z_A];
16                 foreach vertex v ∈ C* do
17                     z_S[v] ← z_S[v] + 1 in atomic;
18                     add the address of C into S[v][z_S[v]];
19             run Algorithm 3 Lines 12 - 18;
20 return T, A;
```

VII. Experiments

A. Experimental Settings

Algorithms. Our empirical studies are conducted against the following designs:

Query algorithms: 1) The online computation algorithms PMBC-OL introduced in Section IV and PMBC-OL* introduced in Section VI-C; 2) The PMBC-Index-based query algorithm PMBC-IQ proposed in Section V.

Index-based techniques: 1) The naive index discussed in Section IV. 2) The PMBC-Index proposed in Section V. We

[1]The *fetch-and-add* operation atomically adds a value to an existing value at a specified memory location. This operation is directly supported by most modern machines.

[2]https://www.openmp.org/

TABLE II: Summary of Datasets

Dataset	Category	$\|U(G)\|$	Type of U	$\|L(G)\|$	Type of L	$\|E(G)\|$	Type of E
Writers	Authorship	89,355	Writer	46,213	Work	144,340	Authorship
YouTube	Affiliation	94,238	User	30,087	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
Wikipedia	Feature	1,853,493	Article	182,947	Category	3,795,796	Inclusion
Amazon	Rating	2,146,057	User	1,230,915	Product	5,743,258	Rating
DBLP	Authorship	1,425,813	Author	4,000,150	Publication	8,649,016	Authorship

report the construction time and size of the index as well as evaluate the optimization and parallelization strategies for PMBC-Index construction in Section VI.

The algorithms are implemented in C++, and the experiments are run on a Linux server with 2 × Intel Xeon E5-2698 processor (2.20GHz, 40 Cores) and 512GB main memory. *We terminate an algorithm if the running time is more than* 10^4 *seconds by default.*

Datasets. In our experiments, we use 10 real-world datasets, which are `Writers`, `Youtube`, `Github`, `BookCrossing`, `StackOverflow`, `Teams`, `ActorMovies`, `Wikipedia`, `Amazon`, and `DBLP`. All the datasets used here can be found in KONECT [3]. The summary of datasets is shown in Table II. $|U(G)|$ and $|L(G)|$ denote the number of vertices in each vertex layers, and $|E(G)|$ represents the number of edges. The types of vertices and edges are also shown in the table. In these datasets, all the vertices with degree equal to zero are removed.

B. Performance of Query Algorithms

In this subsection, we evaluate the performance of query algorithms (PMBC-OL, PMBC-OL*, and PMBC-IQ) for searching personalized maximum bicliques. In each test, we randomly select 200 query vertices from the top-500 high degree vertices with the reported results being the average.

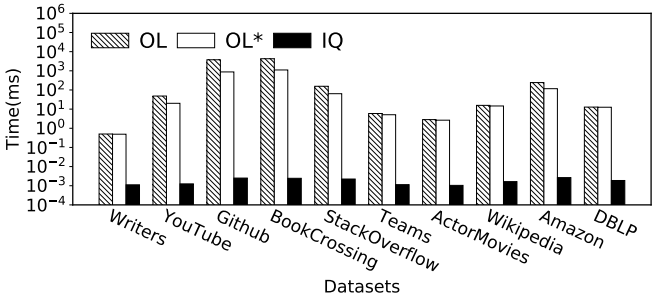
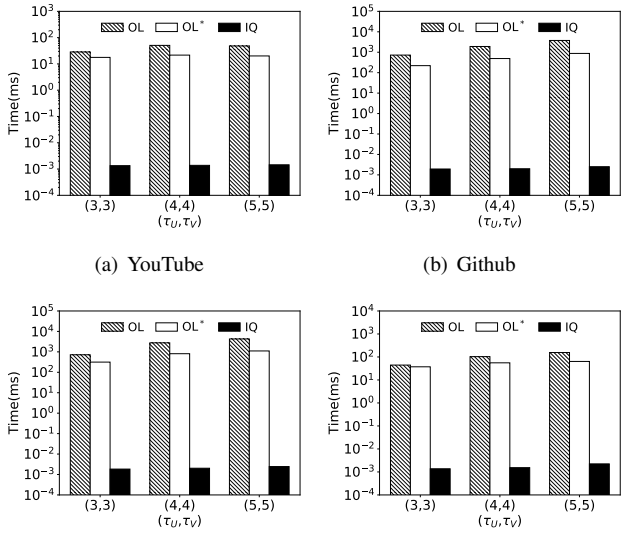


Fig. 6: Evaluating query time

Evaluating query time. In Figure 6, we evaluate the performance of PMBC-OL, PMBC-OL*, and PMBC-IQ on all datasets by setting $\tau_U = 5$ and $\tau_L = 5$, which are also the largest τ_U and τ_L settings in [5]. We can see that benefiting from the PMBC-Index, PMBC-IQ significantly outperforms the online computation algorithms PMBC-OL and PMBC-OL* by up to 5 orders of magnitude. On all the datasets, the PMBC-IQ algorithm can answer the queries within milliseconds. In addition, PMBC-OL* outperforms PMBC-OL as expected due to the upper bounding techniques.



(a) YouTube (b) Github

(c) BookCrossing (d) StackOverflow

Fig. 7: Query $C^q_{\tau_U,\tau_L}$, varying τ_U and τ_L

Evaluating effect of the query parameters τ_U and τ_L. In Figure 7, we vary query parameters τ_U and τ_L on datasets `Actor`, `Wikipedia`, `Amazon`, and `DBLP`. We can see that when τ_U and τ_L increases, the query time for all the query algorithms increases slightly. As expected, PMBC-IQ significantly outperforms PMBC-OL and PMBC-OL* under all settings.

C. Evaluation of Indexing Techniques

In this subsection, we evaluate our indexing techniques. Firstly, we report the index construction time and index size. Secondly, we evaluate the effect of the number of threads t. We set t as 48 by default. Then, we evaluate the scalability of

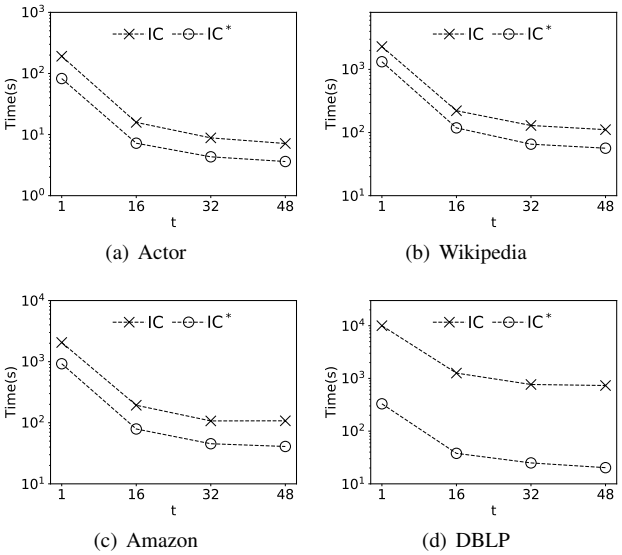
[3]http://konect.cc/

our index construction algorithms. Note that since PMBC-OL* outperforms PMBC-OL as evaluated above, we invoke PMBC-OL* instead of PMBC-OL when computing the indexes.

TABLE III: Evaluating indexing time and index size

Dataset	Index Time (s)		Graph & Index Size (MB)		
	IC	IC*	$\|G\|$	$\|\mathcal{T}\|$	$\|\mathcal{A}\|$
Writers	0.35	0.27	1.10	3.79	2.59
YouTube	7.70	7.51	2.24	4.96	8.20
Github	120.33	99.21	3.36	6.77	13.85
BookCrossing	235.65	228.74	8.77	16.04	31.17
StackOverflow	21.12	19.77	9.93	19.91	25.18
Teams	7.69	6.95	10.43	28.12	25.63
ActorMovies	7.14	3.62	11.22	17.71	21.91
Wikipedia	111.08	56.47	28.96	67.66	65.29
Amazon	107.58	40.88	43.82	100.38	103.06
DBLP	733.88	20.27	65.99	185.41	148.27

Evaluating indexing time and index size. In Table III, we list index construction times and index sizes on different datasets. We can see that the PMBC-Index can be efficiently constructed on all the datasets. In addition, with the cost-sharing optimizations proposed in Section VI-B, PMBC-IC* outperforms PMBC-IC most notably on large datasets such as `Wikipedia`, `Amazon`, and `DBLP`. In addition, the total size of the complete PMBC-Index (i.e., $|\mathcal{T}| + |\mathcal{A}|$) constructed by PMBC-IC* is only $3.5\times$-$6.1\times$ to the graph size (i.e., $|G|$). Note that, the basic index discussed in Section IV is also evaluated with the same setting in this part. The experimental results show that it cannot be built within 10^4 seconds on all the datasets except `Writers` (which needs 1.5 seconds and takes 15.8MB space). Thus, we omit it in Table III and the following experiments.

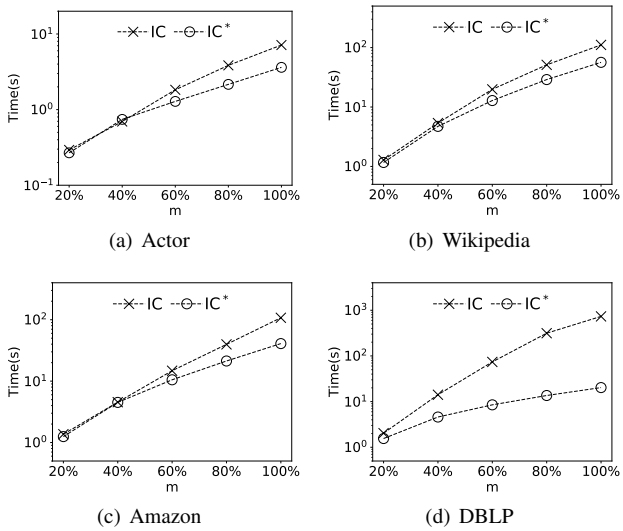


(a) Actor (b) Wikipedia

(c) Amazon (d) DBLP

Fig. 8: Index construction, varying t

Evaluating effect of the number of threads t. Here we evaluate the parallelization algorithms by varying the number of threads t on four datasets `Actor`, `Wikipedia`, `Amazon`, and `DBLP`. As shown in Figure 8, the parallelization techniques significantly improve the efficiency of PMBC-IC and PMBC-IC* on all these datasets. Notably, it can achieve $23.3\times$ speedup for PMBC-IC* with 48 threads on `Actor`. Note that PMBC-IC* outperforms PMBC-IC on all these datasets with or without the parallelization techniques.



(a) Actor (b) Wikipedia

(c) Amazon (d) DBLP

Fig. 9: Index construction, varying m

Scalability of index construction algorithms. *Evaluating the effect of graph size.* In Figure 9, we study the scalability of PMBC-IC and PMBC-IC* by varying the graph size m on datasets `Actor`, `Wikipedia`, `Amazon`, and `DBLP`. When varying m, we randomly sample 20% to 100% edges of the original graphs. We observe that the algorithms PMBC-IC and PMBC-IC* are scalable. Unsurprisingly, the computation costs of all algorithms increase as the percentage of vertices increases. As discussed before, PMBC-IC* outperforms PMBC-IC as expected.

VIII. Conclusion

In this paper we study the efficient personalized maximum biclique search problem. An online computation algorithm PMBC-OL is proposed by taking the advantages of the state-of-the-art solution for maximum biclique search [5]. To solve the problem more efficiently, we resort to index-based techniques and propose the novel PMBC-Index. With the PMBC-Index, query performance improves by several orders of magnitude compared with online computation approaches. In addition, effective optimizations and parallelization techniques are designed to accelerate the index construction process. Extensive experiments on 10 real-world graphs validate both the effectiveness and the efficiency of our query processing and indexing techniques. Furthermore, solutions for solving this problem under a dynamic environment is an interesting research direction for future studies.

REFERENCES

[1] J. Wang, A. P. de Vries, and M. J. T. Reinders, "Unifying user-based and item-based collaborative filtering approaches by similarity fusion," in *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, E. N. Efthimiadis, S. T. Dumais, D. Hawking, and K. Järvelin, Eds. ACM, 2006, pp. 501–508. [Online]. Available: https://doi.org/10.1145/1148170.1148257

[2] M. Ley, "The DBLP computer science bibliography: Evolution, research issues, perspectives," in *String Processing and Information Retrieval, 9th International Symposium, SPIRE 2002, Lisbon, Portugal, September 11-13, 2002, Proceedings*, ser. Lecture Notes in Computer Science, A. H. F. Laender and A. L. Oliveira, Eds., vol. 2476. Springer, 2002, pp. 1–10. [Online]. Available: https://doi.org/10.1007/3-540-45735-6_1

[3] 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*, D. Schwabe, V. A. F. Almeida, H. Glaser, R. Baeza-Yates, and S. B. Moon, Eds. International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 119–130. [Online]. Available: https://doi.org/10.1145/2488388.2488400

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[5] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum biclique search at billion scale," *Proc. VLDB Endow.*, vol. 13, no. 9, pp. 1359–1372, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p1359-lyu.pdf

[6] A. Tanay, R. Sharan, and R. Shamir, "Discovering statistically significant biclusters in gene expression data," *Bioinformatics*, vol. 18, no. suppl_1, pp. S136–S144, 2002.

[7] 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*. IEEE Computer Society, 2003, pp. 187–194. [Online]. Available: https://doi.org/10.1109/ICDM.2003.1250919

[8] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: A survey," *IEEE ACM Trans. Comput. Biol. Bioinform.*, vol. 1, no. 1, pp. 24–45, 2004. [Online]. Available: https://doi.org/10.1109/TCBB.2004.2

[9] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the web for emerging cyber-communities," *Comput. Networks*, vol. 31, no. 11-16, pp. 1481–1493, 1999. [Online]. Available: https://doi.org/10.1016/S1389-1286(99)00040-7

[10] 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*, S. C. Venkatasubramanian and W. M. Jr., Eds. SIAM, 2016, pp. 315–323. [Online]. Available: https://doi.org/10.1137/1.9781611974348.36

[11] M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. R. Tayur, "On bipartite and multipartite clique problems," *J. Algorithms*, vol. 41, no. 2, pp. 388–403, 2001. [Online]. Available: https://doi.org/10.1006/jagm.2001.1199

[12] M. Sözdinler and C. C. Özturan, "Finding maximum edge biclique in bipartite networks by integer programming," in *2018 IEEE International Conference on Computational Science and Engineering, CSE 2018, Bucharest, Romania, October 29-31, 2018*, F. Pop, C. Negru, H. González-Vélez, and J. Rak, Eds. IEEE Computer Society, 2018, pp. 132–137. [Online]. Available: https://doi.org/10.1109/CSE.2018.00025

[13] R. Peeters, "The maximum edge biclique problem is np-complete," *Discret. Appl. Math.*, vol. 131, no. 3, pp. 651–654, 2003. [Online]. Available: https://doi.org/10.1016/S0166-218X(03)00333-0

[14] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. Bernstein, P. Boncz, S. Chaudhuri, A. Cheung, A. Doan *et al.*, "The seattle report on database research," *ACM SIGMOD Record*, vol. 48, no. 4, pp. 44–53, 2020.

[15] C. Wang, Y. Li, X. Luo, Q. Ma, W. Fu, and H. Fu, "The effects of money on fake rating behavior in e-commerce: electrophysiological time course evidence from consumers," *Frontiers in neuroscience*, vol. 12, p. 156, 2018.

[16] J. Ma, D. Zhang, Y. Wang, Y. Zhang, and A. Pozdnoukhov, "Graphrad: a graph-based risky account detection system," in *Proceedings of ACM SIGKDD conference, London, UK*, 2018, p. 9.

[17] S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos, "Netprobe: a fast and scalable system for fraud detection in online auction networks," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 201–210.

[18] S. Shahinpour, S. Shirvani, Z. Ertem, and S. Butenko, "Scale reduction techniques for computing maximum induced bicliques," *Algorithms*, vol. 10, no. 4, p. 113, 2017.

[19] A. A. Al-Yamani, S. Ramsundar, and D. K. Pradhan, "A defect tolerance scheme for nanotechnology circuits," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 54-I, no. 11, pp. 2402–2409, 2007. [Online]. Available: https://doi.org/10.1109/TCSI.2007.907875

[20] B. Yuan and B. Li, "A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 10, no. 3, pp. 25:1–25:19, 2014. [Online]. Available: https://doi.org/10.1145/2517137

[21] Y. Wang, S. Cai, and M. Yin, "New heuristic approaches for maximum balanced biclique problem," *Inf. Sci.*, vol. 432, pp. 362–375, 2018. [Online]. Available: https://doi.org/10.1016/j.ins.2017.12.012

[22] M. Li, J. Hao, and Q. Wu, "General swap-based multiple neighborhood adaptive search for the maximum balanced biclique problem," *Comput. Oper. Res.*, vol. 119, p. 104922, 2020. [Online]. Available: https://doi.org/10.1016/j.cor.2020.104922

[23] Y. Zhou and J. Hao, "Tabu search with graph reduction for finding maximum balanced bicliques in bipartite graphs," *Eng. Appl. Artif. Intell.*, vol. 77, pp. 86–97, 2019. [Online]. Available: https://doi.org/10.1016/j.engappai.2018.09.017

[24] C. McCreesh and P. Prosser, "An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem," in *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings*, ser. Lecture Notes in Computer Science, H. Simonis, Ed., vol. 8451. Springer, 2014, pp. 226–234. [Online]. Available: https://doi.org/10.1007/978-3-319-07046-9_16

[25] Y. Zhou, A. Rossi, and J. Hao, "Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs," *Eur. J. Oper. Res.*, vol. 269, no. 3, pp. 834–843, 2018. [Online]. Available: https://doi.org/10.1016/j.ejor.2018.03.010

[26] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient exact algorithms for maximum balanced biclique search in bipartite graphs," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, G. Li, Z. Li, S. Idreos, and D. Srivastava, Eds. ACM, 2021, pp. 248–260. [Online]. Available: https://doi.org/10.1145/3448016.3459241

[27] D. Eppstein, "Arboricity and bipartite subgraph listing algorithms," *Inf. Process. Lett.*, vol. 51, no. 4, pp. 207–211, 1994. [Online]. Available: https://doi.org/10.1016/0020-0190(94)90121-X

[28] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone, "Consensus algorithms for the generation of all maximal bicliques," *Discret. Appl. Math.*, vol. 145, no. 1, pp. 11–21, 2004. [Online]. Available: https://doi.org/10.1016/j.dam.2003.09.004

[29] G. Liu, K. Sim, and J. Li, "Efficient mining of large maximal bicliques," in *Data Warehousing and Knowledge Discovery, 8th International Conference, DaWaK 2006, Krakow, Poland, September 4-8, 2006, Proceedings*, ser. Lecture Notes in Computer Science, A. M. Tjoa and J. Trujillo, Eds., vol. 4081. Springer, 2006, pp. 437–448. [Online]. Available: https://doi.org/10.1007/11823728_42

[30] 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 Bioinform.*, vol. 15, p. 110, 2014. [Online]. Available: https://doi.org/10.1186/1471-2105-15-110

[31] A. Das and S. Tirthapura, "Shared-memory parallel maximal biclique enumeration," in *26th IEEE International Conference on High Performance Computing, Data, and Analytics, HiPC 2019, Hyderabad, India, December 17-20, 2019*. IEEE, 2019, pp. 34–43. [Online]. Available: https://doi.org/10.1109/HiPC.2019.00016

[32] A. Gély, L. Nourine, and B. Sadi, "Enumeration aspects of maximal cliques and bicliques," *Discret. Appl. Math.*, vol. 157, no. 7, pp. 1447–1459, 2009. [Online]. Available: https://doi.org/10.1016/j.dam.2008.10.010

[33] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," in *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004, Proceedings*, ser. Lecture Notes in Computer Science, T. Hagerup and

J. Katajainen, Eds., vol. 3111. Springer, 2004, pp. 260–272. [Online]. Available: https://doi.org/10.1007/978-3-540-27810-8_23

[34] J. Li, H. Li, D. Soh, and L. Wong, "A correspondence between maximal complete bipartite subgraphs and closed patterns," in *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005, Proceedings*, ser. Lecture Notes in Computer Science, A. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. Gama, Eds., vol. 3721. Springer, 2005, pp. 146–156. [Online]. Available: https://doi.org/10.1007/11564126_18

[35] G. Fang, Y. Wu, M. Li, and J. Chen, "An efficient algorithm for mining frequent closed itemsets," *Informatica (Slovenia)*, vol. 39, no. 1, 2015. [Online]. Available: http://www.informatica.si/index.php/informatica/article/view/754

[36] C. Lucchese, S. Orlando, and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 21–36, 2006. [Online]. Available: https://doi.org/10.1109/TKDE.2006.10

[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*, A. Kementsietsidis and M. A. V. Salles, Eds. IEEE Computer Society, 2012, pp. 270–281. [Online]. Available: https://doi.org/10.1109/ICDE.2012.51

[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, Washington, DC, USA, August 24 - 27, 2003*, L. Getoor, T. E. Senator, P. M. Domingos, and C. Faloutsos, Eds. ACM, 2003, pp. 236–245. [Online]. Available: https://doi.org/10.1145/956750.956779

[39] A. Das and S. Tirthapura, "Incremental maintenance of maximal bicliques in a dynamic bipartite graph," *IEEE Trans. Multi Scale Comput. Syst.*, vol. 4, no. 3, pp. 231–242, 2018. [Online]. Available: https://doi.org/10.1109/TMSCS.2018.2802920

[40] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation: An index-based approach," in *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, and L. Zia, Eds. ACM, 2019, pp. 1130–1141. [Online]. Available: https://doi.org/10.1145/3308558.3313522