SQAC: Scalable Querying of Attribute-Constrained (α, β)-Cores over Large Bipartite Graphs

Xin Deng[1], Peng Peng[1], Baoqing Sun[1], Shuo Dai[1], Zheng Qin[1£], Lijun Chang[2]
[1]*College of Computer Science and Electronic Engineering, Hunan University, China;*
[2]*School of Computer Science, The University of Sydney, Australia;*

[1]{xindeng,hnu16pp,xueqing,ds0230,zqin}@hnu.edu.cn [2]{Lijun.Chang}@sydney.edu.au

***Abstract*—Many important real-world networks can be effectively modeled as bipartite graphs, where vertex attributes convey critical semantic information essential for graph analysis. As a fundamental subgraph structure in bipartite graphs, (α, β)-cores have attracted extensive attention and been widely used. However, existing studies have largely ignored the attribute property in bipartite graphs, which hinders attribute-aware applications in the real world. In this paper, we formulate a new problem of querying attribute-constrained (α, β)-cores over bipartite graphs, and study index-based methods. We first propose a vertex-based core index (VC-Index), which maintains some pivotal attribute sets for each vertex and each possible (α, β) pair. To improve the query efficiency, we then design an attribute-based core index (AC-Index) that computes an index for each possible attribute set. However, these two indexes suffer from either unsatisfactory query performance or excessive space overhead. Therefore, we further construct the minimized AC-Index, named MAC-Index, which significantly reduces the index size by leveraging the containment relationship of the attribute sets for the (α, β)-cores, while guaranteeing efficient query processing. Extensive experiments over real-world datasets confirm the efficiency and effectiveness of our index-based algorithms.**

I. Introduction

Bipartite graphs have been widely used in the real world, such as purchase networks over custom-product relation [1]–[3] and citation networks over author-paper relation [4], [5]. A bipartite graph is represented as $G = (U, L, E)$, with U and L being two disjoint vertex sets that represent different types of entities. An edge $e \in E$ connects a vertex in U and another in L. Analysis of bipartite graphs is of great significance, and computing (α, β)-core is one of the fundamental problems over bipartite graphs [6]–[12]. The (α, β)-core is defined as a maximal subgraph of the given bipartite graph G whose vertices in the upper layer (i.e., U) have a degree of at least α and vertices in the lower layer (i.e., L) have a degree of at least β. Efficient querying of (α, β)-cores has wide-ranging applications, such as group recommendation [13], [14], community detection [11], [15] and fraud detection [3].

However, existing studies over (α, β)-core mainly emphasize the structure information of bipartite graphs, such as (α, β)-core decomposition [9], [10], [16], [17] and (α, β)-core maintenance [6], [18], while disregarding the attribute properties of vertices. Many real-world bipartite graphs are vertex-attributed, in which attributes can reflect important

[£]Corresponding author

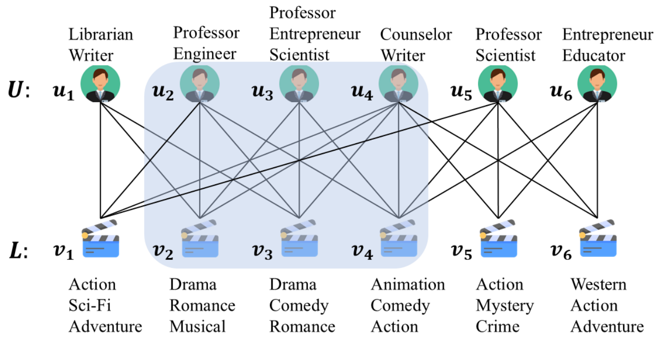


Fig. 1: A user-movie network

semantic information. For example, in E-commerce websites such as Amazon and Alibaba, customers and products form a bipartite graph, where an edge indicates the purchase relationship between a customer and a product. The attribute of a customer may indicate age or occupation, and the attribute of a product may represent the category or popularity.

In this paper, we study a new problem of Scalable Querying of Attribute-constrained (α, β)-Cores (SQAC, for short) over bipartite graphs. Given two integers α, β and two query attribute sets Q_U, Q_L, the SQAC aims to identify an attribute-aware (α, β)-core H from an input graph G in which each vertex $u \in U(H)$ (resp. $v \in L(H)$) has at least one attribute from Q_U (resp. Q_L).

Application. Efficiently analyzing attribute-aware (α, β)-core is significant. We present two real applications as follows.
① *Personalized group recommendation.* Although Ding et al. [16] propose a fault-tolerant group recommendation using the non-attributed (α, β)-core, this work cannot be effectively applied to personalized group recommendation. Consider a user-movie network G of IMDB [19] in Fig. 1, where an edge indicates a user's preference for a movie. A vertex in U represents a user, and the associated attributes indicate its occupation. A vertex in V represents a movie, and the associated attributes indicate its genres. Assume a new user u_0 joins IMDB with the objective of connecting with faculty members (e.g., "Professor" or "Counselor") who share similar tastes in movies (e.g., "Drama", "Comedy" or "Romance"), our SQAC can return a tailored cohesive group H (marked in blue) for u_0, which consists of the users $U(H) = \{u_2, u_3, u_4\}$ and the movies $L(H) = \{v_2, v_3, v_4\}$. While, [16] may

directly return the whole graph G. We can see from Fig. 1 that, although both H and G are $(3,3)$-core, H is evidently more appropriate for the new user u_0, since both members and movies in H are highly compatible with u_0's preferences.
② *Accelerate the querying of attributed community.* Our SQAC can serve as a building block for complicated models. For example, Xu et al. [15] formulate an attributed (α, β)-community model, which aims to identify a connected (α, β)-core H containing the query vertex, and the vertices of H in the same layer share the most attributes under query attribute sets. Apparently, the community is a subset of our SQAC under the same query conditions, as our SQAC relaxes the strict attribute constraints on vertices. There is a state-of-the-art (SOTA) method, called Inc [15], proposed for the attributed (α, β)-community. An essential step of its query processing is to compute the unit (α, β)-cores by traversing the whole graph. On one hand, we can obtain the unit (α, β)-cores by scanning the resulting subgraph of our SQAC, rather than the whole input graph, which significantly narrows the search scope. On the other hand, if the query result of SQAC is empty, we do not need to execute the subsequent steps since the community cannot exist either, avoiding unnecessary time costs. Therefore, our SQAC can function as an optimized procedure for such attributed variations of (α, β)-core.

Challenges. Efficiently computing the SQAC is challenging. Firstly, although we can derive a naive online approach of SQAC based on the existing online approach [16] for the computation of (α, β)-core, such an online approach suffers from extremely poor query efficiency due to the time-consuming peeling process. Its time complexity is presented in Table I. Secondly, when we resort to index-based approaches, a basic idea is to construct the existing Bicore-Index [10] for each possible attribute set pair, where the Bicore-Index has been proposed for the (α, β)-core query over non-attributed bipartite graphs. Although this index exhibits optimal query performance, it requires $O(m \cdot 2^{|\mathcal{A}_U|+|\mathcal{A}_L|})$ space for storage and $O(\delta \cdot m \cdot 2^{|\mathcal{A}_U|+|\mathcal{A}_L|})$ time for construction; this exponential complexity is not acceptable in real applications. Thirdly, the (α, β)-core has two correlated parameters that need to be considered simultaneously. As a result, analysis of the relationship between (α, β)-cores is more difficult than k-cores [20], which requires a two-dimensional method rather than one-dimensional monotonicity. Also, when all possible pairs of attribute sets are considered, the containment relationship among (α, β)-cores becomes intensely complicated.

Our Approach. In response to the challenges outlined above, we first propose a Vertex-based Core index (VC-Index) to answer the SQAC. We construct a VC-Index by maintaining some pivotal attribute sets for each vertex and each possible (α, β) pair. However, this method is inefficient in terms of query performance, since it requires checking the index entries of each vertex of G. To improve the query performance, we propose an Attribute-based Core index (AC-Index), which computes the Bicore-Index for each possible attribute set, optimizing the query performance to be related to each query rather than the input graph size. While the AC-Index offers outstanding query efficiency, it incurs exponential space complexity. Furthermore, we design a superior-optimized index called Minimized attribute-based Core index (MAC-Index), which significantly compresses redundant vertices in the AC-Index leveraging the containment relationship of the attribute sets for (α, β)-cores, while ensuring efficient query efficiency. Note that our proposed index structures rely on the attributes of the input graph rather than the given query attribute set.

Index	Query time	Index space	Construction time
Online	$O(m)$	–	–
$\mathcal{I}_{ABi}$	$O(\|R\|)$	$O(m \cdot 2^{\|\mathcal{A}_U\|+\|\mathcal{A}_L\|})$	$O(\delta \cdot m \cdot 2^{\|\mathcal{A}_U\|+\|\mathcal{A}_L\|})$
$\mathcal{I}_{VC}$	$O(n \cdot \mu \cdot D_{max} + m')$	$O(n \cdot D_{max}^2 \cdot \mu)$	$O(n \cdot \eta \cdot \overline{d} \cdot \mu)$
$\mathcal{I}_{AC}$	$O(\lambda_{ac} \cdot n' + m')$	$O(m \cdot (2^{\|\mathcal{A}_U\|} + 2^{\|\mathcal{A}_L\|}))$	$O(n \cdot \eta \cdot \overline{d} \cdot \log \gamma)$
$\mathcal{I}_{MAC}$	$O(\lambda_{mac} \cdot n' + m')$	$O(\rho \cdot n)$	$O(n \cdot \eta \cdot \overline{d} \cdot \log n)$

TABLE I: **Comparison of different solutions. n' (resp. m') denotes the vertex (resp. edge) number in the intermediate subgraph G', $\mathcal{A}_U$ (resp. $\mathcal{A}_L$) denotes the set of attribute types in $U(G)$ (resp. $L(G)$), $|R|$ denotes the size of query result, δ denotes the maximum value such that the (δ, δ)-core is non-empty and is bounded by $\sqrt{m}$, λ_{ac} (resp. λ_{mac}) denotes the average occurrence frequency of all vertices in G' regarding $\mathcal{I}_{AC}$ (resp. $\mathcal{I}_{MAC}$) and is the ratio of the total number of vertices collected (may contain duplicates) to the number of vertices in G' during the query process, η denotes the average number of changes in the bi-core number of each vertex in G during the construction of MAC-Index. μ denotes the average number of MASs for each vertex and each possible (α, β) pair in $\mathcal{I}_{VC}$, γ denotes the average number of vertices for each vertex block in $\mathcal{I}_{AC}$, D_{max} denotes the maximum degree in G, $\overline{d}$ denotes the average vertex degree in G, and ρ denotes the average occurrence frequency of all vertices in G within $\mathcal{I}_{MAC}$ and is the ratio of the total number of vertices in MAC-Index to the number of vertices in G.**

Contributions. Our contributions are summarized as follows:

- **The SQAC problem.** To the best of our knowledge, we are the first work to propose and solve the attribute-constrained (α, β)-core queries over large attributed bipartite graphs.
- **Two basic index structures.** We propose two basic indexes, called VC-Index and AC-Index, which can correctly answer the query of our proposed SQAC and provide better query efficiency than online query processing.
- **A superior index structure.** We construct a superior index structure, called MAC-Index, leveraging the containment property of attribute sets for the (α, β)-cores. MAC-Index not only significantly reduces the index size from $O(m \cdot (2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|}))$ to $O(\rho \cdot n)$, but also ensures efficient query performance, where ρ is a small integer and $\rho \ll 2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|}$ in practice.
- **Efficient algorithm for index construction and maintenance.** We also propose an efficient algorithm for constructing MAC-Index, and effective techniques to enable the maintenance of MAC-Index.
- **Extensive experiments.** Comprehensive performance

studies on real datasets demonstrate the efficiency and effectiveness of our approaches proposed in this paper.

II. Background

A. Preliminaries

Definition 1 (Attributed Bipartite Graph [15])**.** *An attributed bipartite graph is defined as $G = (V = (U \cup L), E, \mathcal{A}_U, \mathcal{A}_L)$, where U, L are two disjoint vertex sets and $E \subseteq U \times L$ represents the edge set. Each vertex $u \in V$ is associated with a set of attributes denoted by $A(u)$. We use $A_i(u)$ to denote the i-th attribute of u. $\mathcal{A}_U$ (resp. $\mathcal{A}_L$) denotes the set of attribute types in U (resp. L), i.e., $\mathcal{A}_U = \bigcup_{u \in U} A(u)$. Each edge $e \in E$ between two vertices $u \in U$ and $v \in L$ is denoted by (u, v). We may also use $V(G)$, $U(G)$, $L(G)$, $E(G)$, $\mathcal{A}_U(G)$, $\mathcal{A}_L(G)$ to denote V, U, L, E, $\mathcal{A}_U$, $\mathcal{A}_L$ of G, respectively.*

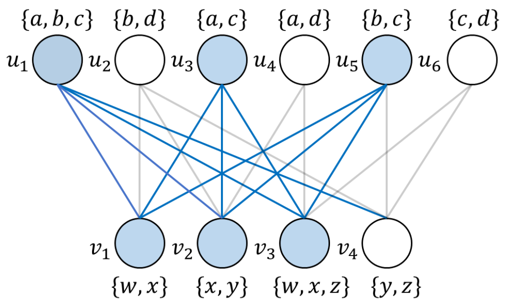


Fig. 2: An attributed bipartite graph G

Considering the graph G in Fig. 2, we define the attribute-induced subgraph of G over an attribute set pair (A_U, A_L) as $G[A_U, A_L]$, where each vertex $u \in U(G[A_U, A_L])$ (resp. $v \in L(G[A_U, A_L])$) contains at least one attribute $A_i(u) \in A_U$ (resp. A_L), i.e., $U(G[A_U, A_L]) = \{u \in U(G) \mid A(u) \cap A_U \neq \emptyset\}$. Particularly, we refer to $G[A_U, \mathcal{A}_L]$ with $A_U \in \mathcal{A}_U$ as a subgraph of G induced by the attribute set A_U. We have a symmetrical statement for $G[\mathcal{A}_U, A_L]$.

Example 1. *Fig. 2 presents an attributed bipartite graph G. Given an attribute set pair ($A_U = \{a, c\}$, $A_L = \{w, x\}$), the attribute-induced subgraph $G[\{a, c\}, \{w, x\}] = \{U = \{u_1, u_3, u_4, u_5, u_6\}, L = \{v_1, v_2, v_3\}\}$, as shown in Fig. 3.*

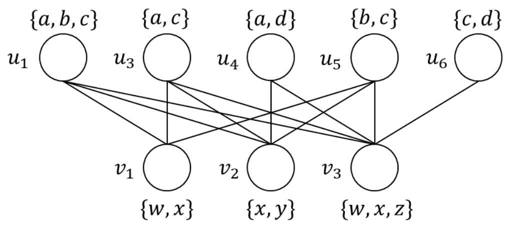


Fig. 3: The attribute-induced subgraph $G[\{a, c\}, \{w, x\}]$

We use n and m to denote the number of vertices and edges in G, respectively. The degree of a vertex u in G is denoted as $deg(u, G)$. In addition, we use $d^U_{max}(G)$ (resp. $d^L_{max}(G)$) to denote the maximum degree in $U(G)$ (resp. $L(G)$). We also use $D_{max}(G)$ to denote the maximum degree in G.

Definition 2 ((α, β)-core [16])**.** *Given a bipartite graph $G = (U, L, E)$ and an integer pair (α, β), the (α, β)-core of G, denoted by $C_{\alpha,\beta}(G)$, is the maximal subgraph such that each vertex in $U(C_{\alpha,\beta})$ (resp. $L(C_{\alpha,\beta})$) has a degree of at least α (resp. β). We may also use $C_{\alpha,\beta}$ for $C_{\alpha,\beta}(G)$.*

There are some typical properties for (α, β)-core [10], [18]: (1) (α, β)-cores have partial nested relationship, i.e., for two (α, β)-core $C_{\alpha,\beta}$ and (α', β')-core $C_{\alpha',\beta'}$, the $C_{\alpha,\beta}$ is **contained** in the $C_{\alpha',\beta'}$ if ($\alpha' < \alpha$, $\beta' \leq \beta$) or ($\alpha' \leq \alpha$, $\beta' < \beta$). (2) An (α, β)-core may be disconnected, but its connected components are usually the "communities" that have been extensively studied.

Definition 3 (Bi-core Number [18])**.** *Given a bipartite graph G and a vertex u, an integer pair (α, β) is a bi-core number of u if the (α, β)-core $C_{\alpha,\beta}$ contains u and there does not exist any other (α', β')-core $C_{\alpha',\beta'}$ containing u such that $C_{\alpha',\beta'}$ is contained in $C_{\alpha,\beta}$. u may have multiple bi-core numbers.*

All bi-core numbers can be computed by a peeling process that iteratively removes the vertices with the smallest degree, which would cost $O(m)$ time [16].

Definition 4 (The SQAC Problem)**.** *Given an attributed bipartite graph $G = (V = (U \cup L), E, \mathcal{A}_U, \mathcal{A}_L)$, two integers α, β and two query attribute sets Q_U, Q_L, SQAC aims to identify an attribute-aware (α, β)-core H from the graph G in which each vertex $u \in U(H)$ (resp. $v \in L(H)$) has at least one attribute from Q_U (resp. Q_L).*

Let's discuss another interesting problem, called "AND" problem. The goal is to identify an (α, β)-core H from the graph G, in which the attribute set associated with each vertex $u \in U(H)$ (resp. $v \in L(H)$) is a subset of Q_U (resp. Q_L). Generally, the "AND" problem tends to provide groups with higher attribute cohesiveness, while our SQAC problem can capture more interesting and diverse cohesive structures, which are crucial for the analysis of potential groups.

Example 2. *Consider the graph G in Fig. 2. Given two integers $\alpha = 3$, $\beta = 3$ and two query attribute sets $Q_U = \{a, c\}$, $Q_L = \{w, x\}$, the result of SQAC is $\{U = \{u_1, u_3, u_5\}, L = \{v_1, v_2, v_3\}\}$, marked in blue in the graph.*

B. Online Query Approach

Existing work [16] provides an efficient online algorithm for querying the non-attributed (α, β)-core, which can be easily adapted to our problem. Specifically, given two integers α, β and two query attribute sets Q_U, Q_L, we first remove the vertices that violate attribute constraints from the input graph G to construct the attribute-induced subgraph $G[Q_U, Q_L]$. We then perform a peeling process [16] to iteratively remove the vertices with the smallest degree from $G[Q_U, Q_L]$. The search process terminates when the remaining vertices form an (α, β)-core. The time cost is $O(m)$. Although this approach can correctly answer the SQAC, it is difficult to apply in large-scale datasets due to time-consuming query processing, prompting us to explore efficient indexing approaches.

C. Naive Index

In this subsection, we design a naive index named ABi-Index, denoted by $\mathcal{I}_{ABi}$, which is adapted from the existing work Bicore-Index [10]. Before introducing ABi-Index, let's discuss the Bicore-Index for the query of non-attributed (α, β)-core. For convenience, we use $\beta_{max,\alpha}(u, G)$ (resp. $\alpha_{max,\beta}(u, G)$) denote the maximum β value for the vertex u and integer α such that u is in the (α, β)-core of G.

1) Bicore-Index: Given a bipartite graph G, the Bicore-Index of G, denoted by $\mathcal{I}_{Bi}$, is a three-level tree structure with two parts for vertices in U and L, respectively, denoted by $\mathcal{I}^U_{Bi}$ and $\mathcal{I}^L_{Bi}$. Since $\mathcal{I}^L_{Bi}$ is symmetrical to $\mathcal{I}^U_{Bi}$, We present only $\mathcal{I}^U_{Bi}$ as follows: the first level is an array containing d^U_{max} pointers to the arrays at the second level, where d^U_{max} denotes the maximum degree among all vertices in U; the second level is a sub-table containing d^U_{max} arrays and the length of the α-th array is equal to the maximum β value (i.e., if the (α, β)-core exists); the third level consists of vertex blocks, where each vertex block $\mathcal{I}^U_{Bi}(\alpha, \beta)$ is associated with an integer pair (α, β) and contains the vertices $u \in U$ with $\beta_{max,\alpha}(u) = \beta$.

The size of $\mathcal{I}_{Bi}$ is $O(m)$ and the construction time is $O(\delta \cdot m)$, where $\delta \leq \sqrt{m}$ [10]. For query processing, it retrieves the (α, β)-core by collecting the vertices in any vertex block $\mathcal{I}^U_{Bi}(\alpha, \beta')$ (resp. $\mathcal{I}^U_{Bi}(\alpha', \beta)$) with $\beta' \geq \beta$ (resp. $\alpha' \geq \alpha$) based on the partial nest relationship of (α, β)-core. The query time is $O(|R|)$, where $|R|$ denotes the size of query result.

Example 3. *Fig. 4 presents the $\mathcal{I}_{Bi}$ for the subgraph in Fig. 3, which also illustrates the procedure of computing $(3, 3)$-core. Processing steps are shown in bold arrows and the visited elements are marked in blue.*

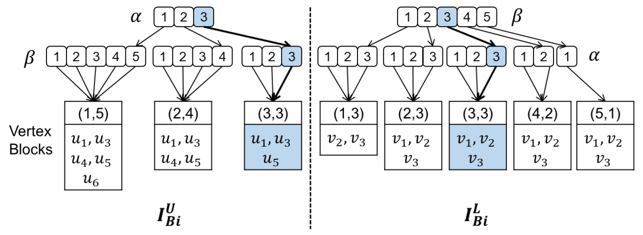


Fig. 4: The Bicore-Index $\mathcal{I}_{Bi}$ for $G[\{a, c\}, \{w, x\}]$ in Fig. 3

2) ABi-Index: A naive index for our SQAC is to compute the Bicore-Index for each possible attribute-induced subgraph. We refer to this index as the Attributed Bicore-Index (ABi-Index, for short), denoted by $\mathcal{I}_{ABi}$. Although ABi-Index can achieve optimal query time cost $O(|R|)$, it takes $O(m \cdot 2^{|\mathcal{A}_U|+|\mathcal{A}_L|})$ space for storage and $O(\delta \cdot m \cdot 2^{|\mathcal{A}_U|+|\mathcal{A}_L|})$ time for construction, which suffers from extremely poor scalability due to its exponential complexity.

III. Vertex-based core Index

In this section, we first analyze the containment property of attribute sets for the (α, β)-cores, and introduce the concept of minimal attribute-constrained set (Section III-A), based on which we design a vertex-based core index (Section III-B). Finally, we present the index construction in Section III-C.

A. Minimal Attribute-constrained Set

Let's discuss the containment property of attribute sets for the (α, β)-cores. Since considering two types of attribute sets simultaneously is inherently more complicated than considering just one, we propose an important lemma as follows.

Lemma 1. *Given an attributed bipartite graph G and two integers α, β, for two subgraphs $G[A_U, \mathcal{A}_L]$ and $G[A'_U, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, A_L]$ and $G[\mathcal{A}_U, A'_L]$), if $A_U, A'_U \subseteq \mathcal{A}_U$ and $A_U \subset A'_U$, then the (α, β)-core $C_{\alpha,\beta}(G[A_U, \mathcal{A}_L])$ is contained in the (α, β)-core $C_{\alpha,\beta}(G[A'_U, \mathcal{A}_L])$.*

Proof. Since A_U is a subset of A'_U, it follows that $G[A_U, \mathcal{A}_L]$ is a subgraph of $G[A'_U, \mathcal{A}_L]$. The lemma holds. $\square$

According to Lemma 1, for any vertex $u \in C_{\alpha,\beta}(G[A_U, \mathcal{A}_L])$, if $C_{\alpha,\beta}(G[A_U, \mathcal{A}_L])$ is contained in $C_{\alpha,\beta}(G[A'_U, \mathcal{A}_L])$, then u must be in $C_{\alpha,\beta}(G[A'_U, \mathcal{A}_L])$. For convenience, we call A an attribute-constrained set for u and (α, β) if the vertex u is in the (α, β)-core of $G[A, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, A]$). Since there may exist multiple attribute-constrained sets for u and (α, β), we also use $AS(u, (\alpha, \beta))$ to denote the set of attribute-constrained sets over u and (α, β). For example, considering the graph in Fig. 2, for the vertex u_5, it is contained in the $(3, 3)$-core of the graph $G[A = \{a, c\}, \mathcal{A}_L = \{w, x, y, z\}]$, thus the attribute set A is an attribute-constrained set over u_5 and $(3, 3)$. Obviously, there also exist multiple attribute-constrained sets for u_5 and $(3, 3)$, such as $\{c\}$, $\{a, b, c\}$, $\{a, c, d\}$, and so on. From this perspective, we propose a novel concept of minimal attribute-constrained set as follows.

Definition 5 (Minimal Attribute-constrained Set)**.** *Given an attributed bipartite graph G, for an attribute-constrained set $A \in AS(u, (\alpha, \beta))$, we define A as a Minimal Attribute-constrained Set (MAS, for short) over u and (α, β), if there does not exist any other attribute-constrained set $A' \in AS(u, (\alpha, \beta))$ such that $A' \subset A$. Also, we use $MAS(u, (\alpha, \beta))$ to denote the set of the minimal attribute-constrained set over u and (α, β).*

Given the graph G in Fig. 2, for the vertex u_5 and integer pair $(3, 3)$, we have $AS(u_5, (3, 3)) = \{\{b\}, \{c\}, \{a, c\}, \{a, b, c\}, \cdots\}$ (just list a few). We can see that both $\{b\}$ and $\{c\}$ are minimal attribute-constrained sets, as no other attribute set in $AS(u_5, (3, 3))$ exists that is a subset of $\{b\}$ or $\{c\}$. Based on Definition 5, we can infer that, for an attribute set $A_U \subseteq \mathcal{A}_U$ (resp. $\mathcal{A}_L$), if there does not exist any MAS $A'_U \in MAS(u, (\alpha, \beta))$ such that $A'_U \subset A_U$, then the vertex u would not be in the (α, β)-core of any subgraph $G[A_U, A_L]$ with $A_L \subseteq \mathcal{A}_L$. Therefore, computing the MASs can help check if a vertex may be included in the (α, β)-core of a given attribute-induced subgraph.

B. Overview of the VC-Index

In this subsection, we propose a Vertex-based Core Index (VC-Index, for short), which maintains some pivotal attribute sets for each vertex and each possible (α, β) pair. We formally define the VC-Index as follows.

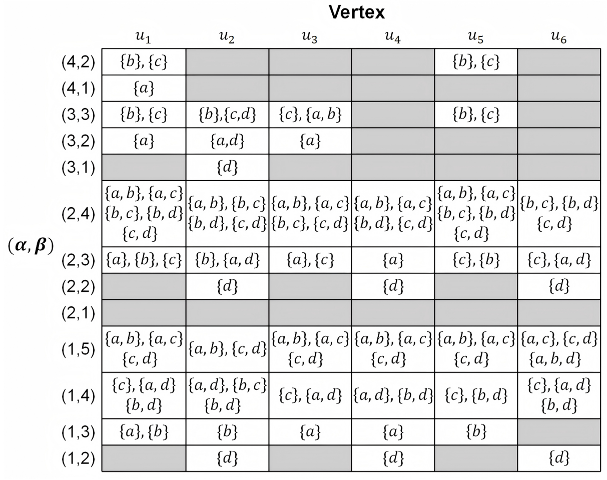
Definition 6 (VC-Index)**.** *Given an attributed bipartite graph $G = (V = (U, L), E, \mathcal{A}_U, \mathcal{A}_L)$, the VC-Index of G, formally denoted by $\mathcal{I}_{VC}$, is essentially a two-dimensional matrix structure with two parts for all vertices in $U(G)$ and $L(G)$, respectively, denoted by $\mathcal{I}^U_{VC}$ and $\mathcal{I}^L_{VC}$. Each cell $\mathcal{I}^U_{VC}(u, (\alpha, \beta))$ in $\mathcal{I}^U_{VC}$ is determined by both a vertex u and an integer pair (α, β), which maintains some attribute sets for u and (α, β), adhering to the following criteria ($\mathcal{I}^L_{VC}$ is symmetric to $\mathcal{I}^U_{VC}$):*

- *Attribute constraint. For each attribute set $A \in \mathcal{I}^U_{VC}(u, (\alpha, \beta))$, A must be a MAS for u and (α, β).*
- *Nested constraint. For each attribute set $A \in \mathcal{I}^U_{VC}(u, (\alpha, \beta))$, there would not exist any $MAS(u, (\alpha, \beta'))$ with $\beta < \beta' \leq \beta_{max}(u, \alpha)$ that contains A, where $\beta_{max}(u, \alpha)$ denotes the maximum β value of the vertex $u \in U(G)$ for α in $\mathcal{I}^U_{VC}$.*

For convenience, we divide $\mathcal{I}^U_{VC}$ into two parts: $\mathcal{I}^{U,\mathcal{A}_U}_{VC}$ and $\mathcal{I}^{U,\mathcal{A}_L}_{VC}$, where the attribute sets in $\mathcal{I}^{U,\mathcal{A}_U}_{VC}$ (resp. $\mathcal{I}^{U,\mathcal{A}_L}_{VC}$) are subsets of $\mathcal{A}_U$ (resp. $\mathcal{A}_L$). Moreover, we do not consider the case of $(\alpha, \beta) = (1, 1)$ in $\mathcal{I}_{VC}$, as the query can be efficiently done by checking if the vertices have neighbors over query attribute sets. Fig. 5 presents $\mathcal{I}^{U,\mathcal{A}_U}_{VC}$ of the graph G in Fig. 2.

Theorem 1. *Given an attributed bipartite graph G, the size of $\mathcal{I}_{VC}$ is $O(n \cdot D^2_{max} \cdot \mu)$, where μ denotes the average number of MASs for each vertex and each possible (α, β) pair in $\mathcal{I}_{VC}$ and D_{max} denotes the maximum degree of all vertices in G.*

Proof. The possible (α, β) pairs of all vertices in $\mathcal{I}_{VC}$ take up no more than $O(\sum_{u \in V(G)}(deg(u, G) \cdot D_{max}))$ space. Since $deg(u, G) < D_{max}$, we have $\sum_{u \in V(G)}(deg(u, G) \cdot D_{max}) \leq \sum_{u \in V(G)} D^2_{max} = n \cdot D^2_{max}$. Hence, the space is bounded by $O(n \cdot D^2_{max} \cdot \mu)$. □



(α, β)	u_1	u_2	u_3	u_4	u_5	u_6
(4,2)	$\{b\}, \{c\}$				$\{b\}, \{c\}$	
(4,1)	$\{a\}$					
(3,3)	$\{b\}, \{c\}$	$\{b\},\{c,d\}$	$\{c\}, \{a, b\}$		$\{b\}, \{c\}$	
(3,2)	$\{a\}$	$\{a,d\}$	$\{a\}$			
(3,1)		$\{d\}$				
(2,4)	$\{a, b\}, \{a, c\}$ $\{b, c\}, \{b, d\}$ $\{c, d\}$	$\{a, b\}, \{b, c\}$ $\{b, d\}, \{c, d\}$	$\{a, b\}, \{a, c\}$ $\{b, c\}, \{c, d\}$	$\{a, b\}, \{a, c\}$ $\{b, d\}, \{c, d\}$	$\{a, b\}, \{a, c\}$ $\{b, c\}, \{b, d\}$ $\{c, d\}$	$\{b, c\}, \{b, d\}$ $\{c, d\}$
(2,3)	$\{a\}, \{b\}, \{c\}$	$\{b\}, \{a, d\}$	$\{a\}, \{c\}$	$\{a\}$	$\{c\}, \{b\}$	$\{c\}, \{a, d\}$
(2,2)		$\{d\}$		$\{d\}$		$\{d\}$
(2,1)						
(1,5)	$\{a, b\}, \{a, c\}$ $\{c, d\}$	$\{a, b\}, \{c, d\}$	$\{a, b\}, \{a, c\}$ $\{c, d\}$	$\{a, b\}, \{a, c\}$ $\{c, d\}$	$\{a, b\}, \{a, c\}$ $\{c, d\}$	$\{a, c\}, \{c, d\}$ $\{a, b, d\}$
(1,4)	$\{c\}, \{a, d\}$ $\{b, d\}$	$\{a, d\}, \{b, c\}$ $\{b, d\}$	$\{c\}, \{a, d\}$	$\{a, d\}, \{b, d\}$	$\{c\}, \{b, d\}$	$\{c\}, \{a, d\}$ $\{b, d\}$
(1,3)	$\{a\}, \{b\}$	$\{b\}$	$\{a\}$	$\{a\}$	$\{b\}$	
(1,2)		$\{d\}$		$\{d\}$		$\{d\}$

Fig. 5: The partial VC-Index $\mathcal{I}^{U,\mathcal{A}_U}_{VC}$ for G in Fig. 2

Correctness of the VC-Index. Given a VC-Index $\mathcal{I}_{VC}$ of an attributed bipartite graph G, two integers α, β and two query attribute sets Q_U, Q_L. For a vertex $u \in U(G)$, if there do not exist two MASs $A_U, A_L \in \mathcal{I}^U_{VC}(u, (\alpha, \beta'))$ ($\beta \leq \beta' \leq \beta_{max}(u, \alpha)$) in $\mathcal{I}_{VC}$ such that $A_U \subseteq Q_U$ and $A_L \subseteq Q_L$, then u would not be in the (α, β)-core of $G[Q_U, Q_L]$ according to Definition 5. In this way, the vertex u would not be contained in the final result and can be safely pruned.

Query Processing. Given an index $\mathcal{I}_{VC}$ of G, two integers α, β, and two query attribute sets Q_U, Q_L, we first traverse all vertices in G and push the candidate vertices that may be included in the final result into a temporary set R. For convenience, we refer to the graph induced by R as **intermediate subgraph**, denoted by G', which can be easily derived as the intersection of the (α, β)-cores in $G[Q_U, \mathcal{A}_L]$ and $G[\mathcal{A}_U, Q_L]$ based on Definition 5, i.e., $G' = C_{\alpha,\beta}(G[Q_U, \mathcal{A}_L]) \cap C_{\alpha,\beta}(G[\mathcal{A}_U, \mathcal{Q}_L]$. Also, we use n' and m' to denote the number of vertices and edges in G', respectively. Finally, to guarantee the correctness of the final result, it costs $O(m')$ time to iteratively remove the vertices in G' that do not satisfy the degree constraints. The overall time cost is $O(n \cdot \mu \cdot D_{max} + m')$.

C. Index Construction

A basic idea of constructing $\mathcal{I}_{VC}$ is to compute the bi-core numbers (i.e., Definition 3) for each subgraph induced by any possible attribute set and derive the MASs of VC-Index based on Definition 6. However, this approach would cost $O((m + n \cdot D_{max} \cdot \mu) \cdot (2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|}))$ time, making it difficult to apply in large datasets due to its poor scalability.

In real applications, for two attribute sets $A, A' \in \mathcal{A}_U$ (resp. $\mathcal{A}_L$), assume $A \backslash A' = \{a\}$ with a being an attribute, we observe that the bi-core numbers of most vertices in $G[A, \mathcal{A}_L]$ are usually equivalent to that in $G[A', \mathcal{A}_L]$. Consequently, it is natural for us to consider employing the (α, β)-core maintenance technology of dynamic graphs to optimize the computation of bi-core numbers, thereby enhancing construction efficiency. There are two feasible maintenance-oriented frameworks for index construction: ① bottom-up: From the subgraph induced by an individual attribute, we gradually expand it to a larger attribute-induced subgraph through insertion algorithm of maintenance; ② top-down: from the original graph, we progressively shrink it to a smaller attribute-induced subgraph through deletion algorithm of maintenance. These two approaches will update the temporary $\mathcal{I}_{VC}$ during the computation of bi-core numbers. Although the time cost for edge insertion and deletion is the same according to the SOTA method of (α, β)-core maintenance [18], the top-down approach evidently incurs extra time overhead, primarily because it updates the temporary index regardless of whether the vertex's bi-core number changes, as a smaller attribute set may be a new MAS. In contrast, the bottom-up approach updates the temporary index only when the vertex's bi-core number changes. Thus, we will apply the bottom-up approach to construct the VC-Index in this paper.

Construction Algorithm. The pseudocode of constructing $\mathcal{I}_{VC}$ is shown in Algorithm 1. For convenience, we use $\beta_{max,\alpha}(u, A)$ to denote the maximum β value for u and α such that u is in (α, β)-core of $G[A, \mathcal{A}_L]$. In Lines 2-6, we obtain the bi-core numbers for each subgraph $G[\{a\}, \mathcal{A}_L]$ induced by any attribute $a \in \mathcal{A}_U$ using the peeling process [16], and then directly add $\{a\}$ to $\mathcal{I}_{VC}^{U(or.L)}(u, (\alpha, \beta_{max,\alpha}(u, \{a\})))$ for every vertex u and each possible α value, as $\{a\}$ is evidently a MAS. In Lines 7-14, we employ the bottom-up approach to derive the bi-core numbers of subgraphs induced by larger attribute sets. For a vertex u with bi-core numbers changed during the computation of a graph $G[A'_U, \mathcal{A}_L]$, if there is no attribute set in cell $\mathcal{I}_{VC}^{U(or.L)}(u, (\alpha, \beta_{max,\alpha}(u, A'_U)))$ that is a subset of A'_U, we need to add A'_U to this cell according to Definition 6, which takes $O(\bar{d} \cdot \mu)$ time and $\bar{d}$ denotes the average degree in G. Finally, we construct the full $\mathcal{I}_{VC}$ by performing symmetrical operations for each attribute set $A_L \in \mathcal{A}_L$. The total cost is $O(n \cdot \eta \cdot \bar{d} \cdot \mu)$, where η denotes the average number of changes in the bi-core number of each vertex in G and is bounded by $2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|}$ ($\eta \ll 2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|}$ in practice).

Algorithm 1: Construction for VC-Index

```
Input: An attributed bipartite graph $G = (V, E, \mathcal{A}_U, \mathcal{A}_L)$
Output: The VC-Index $\mathcal{I}_{VC}$ of $G$
1  $\mathcal{C}_U, \mathcal{C}_L, \mathcal{I}_{VC} \leftarrow \emptyset$ ;
2  foreach individual attribute $a \in \mathcal{A}_U$ do
3      $CD[\{a\}, \mathcal{A}_L] \leftarrow$ the bi-core numbers for $G[\{a\}, \mathcal{A}_L]$ ;
4      Let $\beta_{max,\alpha}(u, A)$ be the maximum $\beta$ value for $u$ and $\alpha$
         such that $u$ is in the ($\alpha$, $\beta$)-core of $G[A, \mathcal{A}_L]$ ;
5      Add $\{a\}$ to $\mathcal{I}_{VC}^{U(or.L)}(u, (\alpha, \beta_{max,\alpha}(u, \{a\})))$ for each
         vertex $u \in G[\{a\}, \mathcal{A}_L]$ and each possible $\alpha$ value;
6      $\mathcal{C}_U.push(CD[\{a\}, \mathcal{A}_L])$ ;
7  while $\mathcal{C}_U \neq \emptyset$ do
8      $CD[A_U, \mathcal{A}_L] \leftarrow \mathcal{C}_U.pop()$;
9      foreach individual attribute $a \in \mathcal{A}_U$ do
10         $A'_U \leftarrow A_U \cup \{a\}$ ;
11         if $G[A'_U, \mathcal{A}_L]$ has not been processed then
12             $V^* \leftarrow$ Vertices with bi-core numbers changed;
13             $\mathcal{C}_U.push(CD[A'_U, \mathcal{A}_L])$ ;
14             Add $A'_U$ to $\mathcal{I}_{VC}^{U(or.L)}(u, (\alpha, \beta_{max,\alpha}(u, A'_U)))$
                 for each vertex $u \in V^*$ and each possible $\alpha$
                 value if there is no attribute set in $\mathcal{I}'$ that is a
                 subset of $A'_U$ ;
15 Do symmetrical operations in Lines 2-14 for each $A \subseteq \mathcal{A}_L$ ;
16 return $\mathcal{I}_{VC}$ ;
```

IV. Attribute-based Core Index

While the VC-Index can correctly answer the query of the SQAC problem, its query efficiency is limited due to the factor n. In this section, we design a query-optimized index that reduces the query time cost from $O(n \cdot \mu \cdot D_{max} + m')$ to $O(n' + m')$, where n' and m' denote the number of vertices and edges in the intermediate subgraph, respectively.

A. Overview of the AC-Index

To eliminate the influence of factor n on the query efficiency of VC-Index, a natural idea is to optimize the query performance to be related to each query rather than the graph size. Considering the existing Bicore-Index [10] for the query of (α, β)-core over non-attributed bipartite graphs, its query efficiency depends solely on the size of each query result. Drawing inspiration from this, we propose a new index structure, called Attribute-based Core Index (AC-Index, for short), which computes the Bicore-Index for the subgraphs induced by each possible attribute set and can help directly extract the intermediate subgraph.

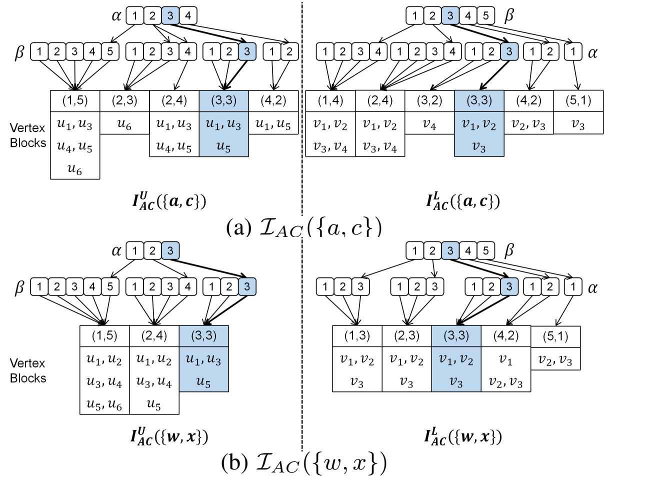
Note that, although both AC-Index and ABi-Index compute the Bicore-Index for the attribute-induced subgraphs, ABi-Index considers the subgraph induced by a pair of attribute sets (A_U, A_L), while AC-Index considers the subgraph induced by an attribute set A_U (resp. A_L). Therefore, the AC-Index can essentially be viewed as a one-dimensional version of the ABi-Index. These attributes (e.g., A_U or A_L) are from the vertices in G. We formally define the AC-Index as follows.

Definition 7 (AC-Index)**.** *Given an attributed bipartite graph $G = (V = (U, L), E, \mathcal{A}_U, \mathcal{A}_L)$, the AC-Index of G, formally denoted by $\mathcal{I}_{AC}$, where each sub-index $\mathcal{I}_{AC}(A)$ maintains a Bicore-Index for a subgraph $G[A, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, A]$) induced by an attribute set $A \in \mathcal{A}_U$ (resp. $\mathcal{A}_L$) and is composed of $\mathcal{I}_{AC}^U(A)$ and $\mathcal{I}_{AC}^L(A)$.*

Example 4. *Considering the attributed bipartite graph in Fig. 2, a part of $\mathcal{I}_{AC}$ is given in Fig. 6, where we only present two sub-indexes: $\mathcal{I}_{AC}(\{a, c\})$ and $\mathcal{I}_{AC}(\{w, x\})$ due to space limit.*

Theorem 2. *Given an AC-Index $\mathcal{I}_{AC}$ of G, the index size is bounded by $O(m \cdot (2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|}))$.*

Proof. The size of sub-index in $\mathcal{I}_{AC}$ is $O(m)$ [10]. The theorem holds since there are $(2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|})$ such possible sub-indexes. □



(a) $\mathcal{I}_{AC}(\{a, c\})$

(b) $\mathcal{I}_{AC}(\{w, x\})$

Fig. 6: The partial AC-Index $\mathcal{I}_{AC}$ of G in Fig. 2

Query Algorithm. Given an AC-Index $\mathcal{I}_{AC}$ of G, two integers α, β and two query attribute sets Q_U, Q_L. We first locate the sub-index $\mathcal{I}_{AC}(Q_U)$ and $\mathcal{I}_{AC}(Q_L)$ in $\mathcal{I}_{AC}$, and retrieve the (α, β)-core of $G[Q_U, \mathcal{A}_L]$ from $\mathcal{I}_{AC}(Q_U)$. We then perform a symmetrical operation for $\mathcal{I}_{AC}(Q_L)$ and obtain the candidate vertex set R Apparently, under the same query parameters, the intermediate subgraph G' induced by R is equivalent to that of VC-Index's intermediate subgraph. Last, we iteratively remove the vertices that violate the degree constraints to obtain the final result. For convenience, we use $\hat{n}'$ to denote the total number of vertices collected (may contain duplicates) during the query process. The overall time cost is $O(\lambda_{ac} \cdot n' + m')$, where λ_{ac} denotes the average occurrence frequency of all vertices in G' regarding AC-Index, i.e., $\lambda_{ac} = \frac{\hat{n}'}{n'}$.

Construction Algorithm. Given that the computation of bi-core number is an essential step for deriving the Bicore-Index [10], we also consider the bottom-up framework to construct AC-Index. We first construct the Bicore-Index for the subgraph $G[\{a\}, \mathcal{A}_L]$ induced by each attribute $a \in \mathcal{A}_U$. Based on these smaller subgraphs that have been computed, we then perform (α, β)-core maintenance to compute the bi-core numbers for larger subgraphs and derive the corresponding Bicore-Index. During the computation process, if the bi-core number of a vertex changes, it costs $O(\overline{d} \cdot \log \gamma)$ time to update the temporary $\mathcal{I}_{AC}$, where γ denotes the average number of vertices for each vertex block. Finally, to obtain a full $\mathcal{I}_{AC}$, we perform symmetrical operations for the subgraph induced by the attribute set $A \in \mathcal{A}_L$. The total cost is $O(n \cdot \eta \cdot \overline{d} \cdot \log \gamma)$.

V. Minimized Attribute-based core Index

Although the AC-Index achieves a great improvement of query performance, it still suffers from poor scalability in terms of index size. In this section, we demonstrate the compressibility of vertex blocks in the AC-Index, proposing the minimized AC-Index, which significantly optimizes the space cost of AC-Index, while ensuring efficient query efficiency.

A. *Overview of the MAC-Index*

In the AC-Index $\mathcal{I}_{AC}$, we observe that a vertex u may be stored in different sub-indexes of $\mathcal{I}_{AC}$ for the same (α, β), which would result in significant storage redundancy in $\mathcal{I}_{AC}$. Next, we will introduce the concept of the attribute-dominant vertex and illustrate how such a vertex utilizes the dominant property to compress redundant information in $\mathcal{I}_{AC}$. Finally, we propose the minimized AC-Index based on these properties.

Definition 8 (Attribute-dominant Vertex)**.** *Given an AC-Index $\mathcal{I}_{AC}$ of G, for any vertex $u \in \mathcal{I}^U_{AC}(A, (\alpha, \beta))$, if there does not exist any other vertex block $\mathcal{I}^U_{AC}(A', (\alpha, \beta))$ containing u with $A' \subset A$, then the vertex u is an attribute-dominant vertex over the attribute set A and the integer pair (α, β). The symmetric case for the vertices in any $\mathcal{I}^L_{AC}(A, (\alpha, \beta))$.*

Considering $\mathcal{I}_{AC}$ of the graph G in Fig. 2, the vertex u_3 is contained in $\mathcal{I}^U_{AC}(\{c\}, (3, 3))$, $\mathcal{I}^U_{AC}(\{a, c\}, (3, 3))$, and so on, then u_3 is an attribute-dominant vertex over $\{c\}$ and $(3, 3)$. According to Definition 8, we can infer that, for a vertex $u \in U(G)$, if u is an attribute-dominant vertex in the vertex block $\mathcal{I}^U_{AC}(A, (\alpha, \beta))$, then u must also be contained in the (α, β)-core of any subgraph $G[A', \mathcal{A}_L]$ with $A \subseteq A'$. Consequently, when querying the (α, β)-core H of the subgraph $G[A, \mathcal{A}_L]$, we can also retrieve the result vertices $U(H)$ (resp. $L(H)$) from the attribute-dominant vertices in each vertex block $\mathcal{I}^U_{AC}(A', (\alpha, \beta'))$ with $A' \subseteq A$ and $\beta' \geq \beta$ (resp. $\mathcal{I}^L_{AC}(A', (\alpha', \beta))$ with $A' \subseteq A$ and $\alpha' \geq \alpha$).

With the dominant property among the vertices of different vertex blocks in $\mathcal{I}_{AC}$, we construct a superior-optimized index, named Minimized Attribute-based Core Index (MAC-Index, for short). We formally define the MAC-Index as follows.

Definition 9 (MAC-Index)**.** *Given an attributed bipartite graph G, the MAC-Index of G, formally denoted by $\mathcal{I}_{MAC}$, which just maintains the attribute-dominant vertices of AC-Index, i.e., for each vertex u in any vertex block $\mathcal{I}^U_{MAC}(A, (\alpha, \beta))$ (resp. $\mathcal{I}^L_{MAC}(A, (\alpha, \beta))$), u is an attribute-dominant vertex over u and (α, β). Note that the other vertices in AC-Index are considered redundant and will be eliminated.*

For convenience, we use $\mathcal{I}_{MAC}(A)$, $\mathcal{I}^U_{MAC}(A)$, and $\mathcal{I}^L_{MAC}(A)$ to correspond to $\mathcal{I}_{AC}(A)$, $\mathcal{I}^U_{AC}(A)$ and $\mathcal{I}^L_{AC}(A)$ of the AC-Index, respectively. We also use $\mathcal{I}^{\mathcal{A},U}_{MAC}$ (resp. $\mathcal{I}^{\mathcal{A},L}_{MAC}$) to denote the set of $\mathcal{I}^U_{MAC}(A)$ (resp. $\mathcal{I}^L_{MAC}(A)$) with $A \subseteq \mathcal{A}$. Apparently, the MAC-Index can be easily serialized and deserialized in the experiment. Fig. 7 presents the partial $\mathcal{I}_{MAC}$ of the graph in Fig. 2.

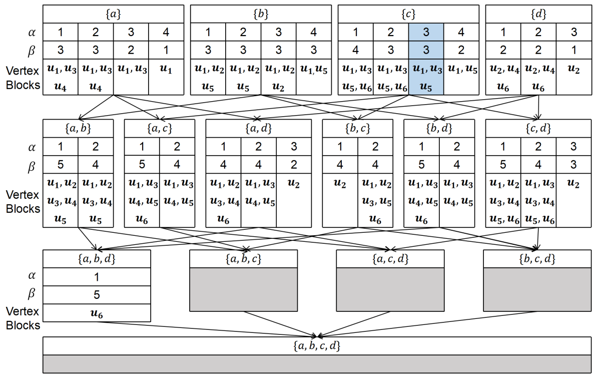

{a}
α 1 2 3 4
β 3 3 2 1
Vertex Blocks u1,u3 u4 | u1,u3 u4 | u1,u3 | u1
{b}
α 1 2 3 4
β 3 3 3 3
Vertex Blocks u1,u2 u5 | u1,u2 u5 | u1,u2 u2 | u1,u5
{c}
α 1 2 3 4
β 4 3 3 2
Vertex Blocks u1,u3 u5,u6 | u1,u3 u5,u6 | u1,u3 u5 | u1,u5
{d}
α 1 2 3
β 2 2 1
Vertex Blocks u2,u4 u6 | u2,u4 u6 | u2
{a,b}
α 1 2
β 5 4
Vertex Blocks u1,u2 u3,u4 u5 | u1,u2 u3,u4 u5
{a,c}
α 1 2
β 5 4
Vertex Blocks u1,u3 u4,u5 u6 | u1,u3 u4,u5
{a,d}
α 1 2 3
β 4 4 2
Vertex Blocks u1,u2 u3,u4 u6 | u1,u3 u4,u5 | u2
{b,c}
α 1 2
β 4 4
Vertex Blocks u2 | u1,u2 u3,u5 u6
{b,d}
α 1 2
β 5 4
Vertex Blocks u1,u3 u4,u5 u6 | u1,u3 u4,u5
{c,d}
α 1 2 3
β 5 4 3
Vertex Blocks u1,u2 u3,u4 u5,u6 | u1,u3 u3,u4 u5,u6 | u2
{a,b,d}
α 1
β 5
Vertex Blocks u6
{a,b,c}
{a,c,d}
{b,c,d}
{a,b,c,d}

Fig. 7: The partial MAC-Index $\mathcal{I}^{\mathcal{A}_U,U}_{MAC}$ for G in Fig. 2

Theorem 3. *The size of $\mathcal{I}_{MAC}$ is bounded by $O(\rho \cdot n)$, where ρ denotes the average occurrence frequency of all vertices in G and is the ratio of the total number of vertices in $\mathcal{I}_{MAC}$ to the number of vertices in G.*

Proof. The primary space cost of the $\mathcal{I}_{MAC}$ stems from storing the vertices of the vertex blocks. For convenience, we use $|\mathcal{I}^U_{MAC}(A)|$ to denote the total number of vertices in all vertex blocks of $\mathcal{I}^U_{MAC}(A)$. The total number of vertices in $\mathcal{I}_{MAC}$ can be represented as $\hat{n} = \sum_{A \in \{\mathcal{A}_U \cup \mathcal{A}_L\}} (|\mathcal{I}^U_{MAC}(A)| + |\mathcal{I}^L_{MAC}(A)|)$. Thus, we have $\rho = \frac{\hat{n}}{n}$. The theorem holds. In practice, ρ is a small integer and $\rho \ll n$, as shown in Table II. □

Correctness of MAC-Index. Given two query attribute sets Q_U, Q_L, the MAC-Index essentially aims to find the same intermediate subgraph as that of AC-Index under identical query parameters. For the (α, β)-core $C_{\alpha,\beta}(G[Q_U, \mathcal{A}_L])$ in $G[Q_U, \mathcal{A}_L]$, AC-Index can directly retrieve it from $\mathcal{I}_{AC}(Q_U)$. Since MAC-Index is a compressed version of AC-Index, we can obtain some vertices of $C_{\alpha,\beta}(G[Q_U, \mathcal{A}_L])$ from $\mathcal{I}_{MAC}(Q_U)$. The remaining vertices can be found in other vertex blocks $\mathcal{I}^U_{MAC}(A, (\alpha, \beta'))$ (resp. $\mathcal{I}^L_{MAC}(A, (\alpha', \beta))$) of MAC-Index with $A \subseteq Q_U$ and $\beta' \geq \beta$ (resp. $\alpha' \geq \alpha$), as they are treated as attribute-dominant vertices in storage according to Definition 8. Thus, MAC-Index and AC-Index can find the same intermediate subgraph given identical query parameters.

Algorithm 2: Query based on MAC-Index

```
Input: $G$, $\alpha$, $\beta$, $Q_U$, $Q_L$, $\mathcal{I}_{MAC}$
Output: The result of SQAC
1  $R \leftarrow \emptyset$ ;
2  $\mathbb{Q}_U \leftarrow$ All possible subsets of $Q_U$;
3  $\mathbb{Q}_L \leftarrow$ All possible subsets of $Q_L$;
4  foreach attribute set $A \in \mathbb{Q}_U$ do
5      foreach integer $\beta' \geq \beta$ do
6          if $\mathcal{I}^U_{MAC}(A, (\alpha, \beta'))$ exists then
7              $R \leftarrow R \cup \mathcal{I}^U_{MAC}(A, (\alpha, \beta'))$ ;
8      For any $\mathcal{I}^L_{MAC}(A, (\alpha', \beta))$ with $\alpha' \geq \alpha$, do the
         operations that are symmetrical to those in Line 5-7 ;
9  foreach attribute set $A \subseteq \mathbb{Q}_L$ do
10     Do the same operations as Lines 5-8 ;
11 $G' \leftarrow$ the subgraph of $G$ induced by $R$ ;
12 Do peeling process for $G'$ to obtain final results ;
13 return $G'$ ;
```

Query Algorithm. The pseudocode for MAC-Index-based query algorithm is presented in Algorithm 2. Given a MAC-Index $\mathcal{I}_{MAC}$ of G, two integers α, β, and two query attribute sets Q_U, Q_L. In Lines 4-8, we first collect the (α, β)-core of $G[Q_U, \mathcal{A}_L]$ from each vertex block $\mathcal{I}^U_{MAC}(A, (\alpha, \beta'))$ (resp. $\mathcal{I}^L_{MAC}(A, (\alpha', \beta))$) such that $A \subseteq Q_U$ and $\beta' \geq \beta$ (resp. $\alpha' \geq \alpha$). We then perform symmetrical operations to retrieve the (α, β)-core of $G[\mathcal{A}_U, Q_L]$, thereby obtaining the intermediate subgraph G' (Lines 9-11). Finally, we conduct a peeling process for G' to obtain the final result. The overall time cost is $O(\lambda_{mac} \cdot n' + m')$, where λ_{mac} denotes the average occurrence frequency of all vertices in G' regarding MAC-Index, and its calculation method is aligned with λ_{ac} (see Section IV-A). Note that, in practical applications, users generally select just a few types of attributes they prefer; thus, the query attribute set is often relatively small.

Other Supported Queries. Our MAC-Index also supports other variant problems. We present some of them as follows. ① *Attribute-constrained (α, β)-core search.* Given two integers α, β, two query attribute sets Q_U, Q_L, and a query vertex q, the problem is to identify a connected (α, β)-core containing q over the graph $G[Q_U, Q_L]$. We begin by retrieving the result subgraph R of the SQAC problem, which would take $O(\lambda_{mac} \cdot n' + m')$ time. Then, we perform a BFS search strategy in R, starting from q, until a maximal connected component is obtained. The total cost is $O(\lambda_{mac} \cdot n' + m' + |R|)$. ② *Speedup for other models.* As discussed in Section I, the attributed community model [15] is a subset of the result of our SQAC under the same query parameters. This property enables us to accelerate queries of the attributed community based on our MAC-Index. We also provide a case study to demonstrate the effectiveness of this method in Section VI-G.

Approximate Query Algorithm. Let's discuss a basic idea of the approximate query algorithm based on MAC-Index. For convenience, we use $G'[A_U, A_L]$ to denote the intermediate subgraph for the attribute set pair (A_U, A_L). Given two integers α, β and two query attribute sets Q_U, Q_L, we first retrieve $G'[a_U, a_L]$ via MAC-Index, where a_U (resp. a_L) denote an attribute from Q_U (resp. Q_L). Then, if there exists a (α, β)-core $C_{\alpha,\beta}(G'[a_U, a_L])$ during the peeling process of $G'[a_U, a_L]$, the search ends since $C_{\alpha,\beta}(G'[a_U, a_L])$ is clearly an answer. Otherwise, we iteratively expand $G'[a_U, a_L]$ into a larger one $G'[A_U, A_L]$ using MAC-Index, where $a_U \in A_U$ and $A_U \subseteq Q_U$, as well as $a_L \in A_L$ and $A_L \subseteq Q_L$. During this process, we perform the peeling operation to check for the existence of a (α, β)-core within $G'[A_U, A_L]$. The whole process ends when either a (α, β)-core is found in $G'[A_U, A_L]$, or when no (α, β)-core exists in $G'[Q_U, Q_L]$.

B. The Construction for MAC-Index

Computing the attribute-dominant vertices for an attribute set and an (α, β) pair is a crucial step in constructing MAC-Index, significantly reducing the size of the index. In this subsection, we will first introduce the construction framework for MAC-Index, and then elaborate on the computation of the attribute-dominant vertex.

Algorithm 3: Construction for MAC-Index

```
Input: An attributed bipartite graph $G = (V, E, \mathcal{A}_U, \mathcal{A}_L)$
Output: The MAC-Index $\mathcal{I}_{MAC}$ of $G$
1  $\mathcal{C}_U, \mathcal{C}_L, \mathcal{I}_{MAC} \leftarrow \emptyset$ ;
2  foreach individual attribute $a \in \mathcal{A}_U$ do
3      $CD[\{a\}, \mathcal{A}_L] \leftarrow$ compute the bi-core numbers for the
         subgraph $G[\{a\}, \mathcal{A}_L]$ induced by $\{a\}$ ;
4      Construct the sub-index $\mathcal{I}_{MAC}(\{a\})$ by using the
         existing construction algorithm for Bicore-Index [10] ;
5      $\mathcal{C}_U.push(CD[\{a\}, \mathcal{A}_L])$ ;
6  foreach individual attribute $a' \in \mathcal{A}_L$ do
7      Do the operations that are symmetrical to those in
         Lines 3-5 in Algorithm 3;
8  Call the procedure ComputeADVetex ($G$, $\mathcal{C}_U$, $\mathcal{C}_L$, $\mathcal{I}_{MAC}$) to
     iteratively compute the attribute-dominant vertices ;
9  return $\mathcal{I}_{MAC}$ ;
```

The Construction Framework. The pseudocode for MAC-Index construction is presented in Algorithm 3. In Lines 1-5, we first compute each sub-index $\mathcal{I}_{MAC}(\{a\})$ for any small subgraph $G[\{a\}, \mathcal{A}_L]$, where a denotes an attribute from $\mathcal{A}_U$. Specifically, based on Definition 8, $\mathcal{I}_{MAC}(\{a\})$ is identical to $\mathcal{I}_{AC}(\{a\})$ since $\{a\}$ can only be a subset of other sets. Given that $\mathcal{I}_{AC}(\{a\})$ is the Bicore-Index [10] of the graph $G[\{a\}, \mathcal{A}_L]$, we can invoke the existing algorithm to compute the bi-core numbers of $G[\{a\}, \mathcal{A}_L]$ and construct the sub-index $\mathcal{I}_{MAC}(\{a\})$. We then perform symmetrical operations for each small subgraph $G[\mathcal{A}_U, \{a'\}]$ with $a' \in \mathcal{A}_L$. Lastly, based on these small subgraphs that have been processed, we invoke the procedure *ComputeADVertex* (i.e., Algorithm 4) to iteratively expand them into larger ones, and simultaneously compute the attribute-dominant vertices for larger attribute sets in Line 8.

The Computation of Attribute-dominant Vertices. In Algorithm 4, we sequentially process these large subgraphs in ascending order, based on the size of their corresponding attribute sets. Next, let's discuss the computation of attribute-dominant vertices as we process a subgraph in Lines 6-15. Let $G[A'_U, \mathcal{A}_L]$ be the subgraph to be processed and $G[A_U, \mathcal{A}_L]$ be the subgraph that has been processed, where

Algorithm 4: Attribute-dominant Vertex Computation

```
1  Procedure ComputeADVertex($G, \mathcal{C}_U, \mathcal{C}_L, \mathcal{I}_{MAC}$)
2      while $\mathcal{C}_U \neq \emptyset$ do
3          $CD[A_U, \mathcal{A}_L] \leftarrow \mathcal{C}_U.pop()$ ;
4          foreach individual attribute $a \in \mathcal{A}_U$ do
5              $A'_U \leftarrow A_U \cup \{a\}$ ;
6              if $G[A'_U, \mathcal{A}_L]$ has not been processed then
7                  $V^* \leftarrow$ Vertices with the bi-core numbers
                     changed by inserting edges containing $a$
                     into $G[A_U, \mathcal{A}_L]$;
8                  $\mathcal{C}_U.push(CD[A'_U, \mathcal{A}_L])$ ;
9                  foreach vertex $u \in V^* \wedge u \in U(G)$ do
10                     Let $\beta_{max,\alpha}(u)$ be the maximum $\beta$
                         value for $u$ and $\alpha$, such that $u$ is in the
                         ($\alpha$, $\beta$)-core of the graph $G[A'_U, \mathcal{A}_L]$ ;
11                     foreach integer
                         $1 \leq \alpha \leq deg(u, G[A'_U, \mathcal{A}_L])$ do
12                         if there is no vertex block
                             $\mathcal{I}^U_{MAC}(A''_U, (\alpha, \beta_{max,\alpha}(u)))$
                             containing $u$ such that $A''_U \subset A'_U$
                             then
13                             Add $u$ to
                                 $\mathcal{I}^U_{MAC}(A'_U, (\alpha, \beta_{max,\alpha}(u)))$ ;
14                 foreach vertex $v \in V^* \wedge v \in L(G)$ do
15                     Do the operation that are symmetrical to
                         those in Lines 9-13 in Algorithm 4 ;
16     while $\mathcal{C}_L \neq \emptyset$ do
17         Do the operations that are symmetrical to those in
             Lines 3-15 of Algorithm 4 ;
```

$A'_U = \{A_U \cup \{a\}\}$ with a being an attribute. For convenience, we use $\beta_{max,\alpha}(u)$ (resp. $\alpha_{max,\beta}(u)$) to denote the maximum value of β for the vertex u and the integer α such that u is contained in the (α, β)-core of the graph $G[A'_U, \mathcal{A}_L]$. Note that we can directly obtain $\beta_{max,\alpha}(u)$ from the bi-core numbers. When inserting all edges containing a into $G[A_U, \mathcal{A}_L]$, if the bi-core numbers of a vertex $u \in U(G[A_U, \mathcal{A}_L])$ (resp. $L(G[A_U, \mathcal{A}_L])$) remain unchanged, we need to do nothing; otherwise, we need to update the temporary sub-index $\mathcal{I}^U_{MAC}(A'_U)$ (resp. $\mathcal{I}^L_{MAC}(A'_U)$) as follows (Lines 9-15): for any integer $\alpha \leq deg(u, G[A'_U, \mathcal{A}_L])$, if there does not exist any other vertex block $\mathcal{I}^U_{MAC}(A''_U, (\alpha, \beta_{max,\alpha}(u)))$ containing u such that $A''_U \subset A'_U$, then we add u to $\mathcal{I}^U_{MAC}(A'_U, (\alpha, \beta_{max,\alpha}(u)))$ (Lines 9-13), as u must be an attribute-dominant vertex over A'_U and $(\alpha, \beta_{max,\alpha}(u))$ according to Definition 8. Otherwise, we need to do nothing. This update costs $O(\overline{d} \cdot \log n)$ time. To obtain the full index, we perform symmetrical operations for each subgraph $G[\mathcal{A}_U, A_L]$ induced by the attribute set $A_L \in \mathcal{A}_L$ in Lines 16-17. The total time cost of constructing $\mathcal{I}_{MAC}$ is bounded by $O(n \cdot \eta \cdot \overline{d} \cdot \log n)$.

C. Discussions on Maintenance of MAC-Index

In this subsection, we propose the maintenance algorithm for the MAC-Index including edge insertion and deletion. When edge insertion happens, assume the incoming edge is $e_i = (u_i, v_i)$, with $A(u_i)$ (resp. $A(v_i)$) denoting the set of attributes associated with u_i (resp. v_i). Obviously, we only need to consider the affected subgraph $G[A, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, A]$) such that $A \cap A(u_i) \neq \emptyset$, since the subgraphs induced by other attribute sets remain unchanged. Next, we progressively process the affected subgraphs in ascending order according to the sizes of their corresponding attribute sets. Specifically, for an affected subgraph $G[A, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, A]$), we begin by calling (α, β)-core maintenance [18] to obtain the vertices V^* whose bi-core number changes. Then, for a vertex $u \in V^*$, we would update the sub-index $\mathcal{I}^U_{MAC}(A)$ (resp. $\mathcal{I}^L_{MAC}(A)$) as follows: if $\beta_{max,\alpha}(u)$ for $G[A, \mathcal{A}_L]$ changes to $\beta^+_{max,\alpha}(u)$ and there is no vertex block $\mathcal{I}^U_{MAC}(A', (\alpha, \beta^+_{max,\alpha}(u)))$ containing u such that $A' \subset A$, we need to add u to $\mathcal{I}^U_{MAC}(A, (\alpha, \beta^+_{max,\alpha}(u)))$. Concurrently, if u is exactly included in the vertex block $\mathcal{I}^U_{MAC}(A, (\alpha, \beta_{max,\alpha}(u)))$ before, we need to remove u from $\mathcal{I}^U_{MAC}(A, (\alpha, \beta_{max,\alpha}(u)))$. Moreover, if there exists a vertex block $\mathcal{I}^U_{MAC}(A'', (\alpha, \beta^+_{max,\alpha}(u)))$ containing u such that $A \subset A''$, we would remove u from $\mathcal{I}^U_{MAC}(A'', (\alpha, \beta^+_{max,\alpha}(u)))$.

For an edge e_j deletion, we also only focus on the affected subgraph as in the case of edge insertion. Specifically, for an affected subgraph $G[A, \mathcal{A}_L]$, if the bi-core number of a vertex u changes, we will update $\mathcal{I}^U_{MAC}(A)$ (resp. $\mathcal{I}^L_{MAC}(A)$) in the same manner as the edge insertion. In addition, we need to consider the impact of the change in the bi-core number of vertices in $\mathcal{I}^U_{MAC}(A')$ on $\mathcal{I}^U_{MAC}(A)$ when $A' \subset A$. If $\beta_{max,\alpha}(u)$ for $G[A', \mathcal{A}_L]$ changes to $\beta^-_{max,\alpha}(u)$ when deleting e_j, then we will remove u from $\mathcal{I}^U_{MAC}(A', (\alpha, \beta_{max,\alpha}(u)))$ if u exists. At this time, assume the bi-core number of u in $G[A, \mathcal{A}_L]$ does not change after deleting e_j; if $\beta_{max,\alpha}(u)$ of $G[A, \mathcal{A}_L]$ is identical to that of $G[A', \mathcal{A}_L]$, and there is no vertex block $\mathcal{I}^U_{MAC}(A'', (\alpha, \beta_{max,\alpha}(u)))$ containing u such that $A'' \subset A$, we need to add u to $\mathcal{I}^U_{MAC}(A, (\alpha, \beta_{max,\alpha}(u)))$ since u is a new attribute-dominant vertex for A and $(\alpha, \beta_{max,\alpha}(u))$. The maintenance process will terminate when all the affected subgraphs have been handled.

VI. Experimental Evaluation

In this section, we evaluate the performance of the proposed algorithms. All algorithms are implemented in C++ and run on a CentOS machine of 1TB memory and an Intel(R) Xeon(R) Silver-4210R 2.40GHz CPU. We terminate an algorithm if it runs more than 72 hours, which is denoted as INF.

Datasets	$\lvert E\rvert$	$\lvert U\rvert$	$\lvert L\rvert$	$\lvert\mathcal{A}_U\rvert$	$\lvert\mathcal{A}_L\rvert$	d^U_{max}	d^L_{max}	λ_{ac}	λ_{mac}	ρ	**Real Labels**
IMDB-5M	5.46M	1.05M	2.13M	6	6	359	27	146.58	146.84	4.54	✓
IMDB-7M	7.66M	1.02M	2.34M	6	6	341	18	264.54	264.82	4.25	✓
IMDB-10M	10.53M	1.05M	2.57M	5	7	611	45	89.85	100.08	5.36	✓
Tmall-3M	3.56M	1.07M	0.31M	4	8	128	158	19.8	19.8	4.69	✓
Tmall-10M	10.47M	2.06M	0.69M	4	9	101	68	2.2	2.2	4.15	✓
DBLP	12.28M	1.95M	5.62M	6	6	1386	287	1.16	1.16	4.26	✗

TABLE II: **Statistics of Datasets**

A. Setup

1) Datasets: Table II presents the statistics of datasets used in the experiments. We extract three real graphs from IMDB [19] and two real graphs from Tmall [21]. The **IMDB** dataset consists of a bipartite person-movie network, where an edge denotes that a person has contributed to a movie. The attributes of a person indicate occupation, and the attributes of a movie represent its genre. The **Tmall** dataset is an E-commerce website that contains the customer-product bipartite network, where the attribute of a customer indicates behavior and the attributes associated with a product represent its category. The **DBLP** dataset is obtained from KONECT [22] and does not offer the vertex attributes; we generate synthetic attributes as in previous work [15], following a specific uniform distribution. All the codes and datasets are available on GitHub [23].

2) Algorithm: There are two comparative algorithms in our work. The first is an online approach introduced in Section II-B, denoted as **Online**. The second is a SOTA solution, Bicore-Index, that is adapted to our problem (Section II-C), denoted as **ABi-Index**. In addition, there are three of our approaches, including **VC-Index** (Section III), **AC-Index** (Section IV) and **MAC-Index** (Section V). Note that we omit the experiments that assess the accuracy of various indexing strategies, since these methods mentioned above are exact algorithm. Moreover, ABi-Index cannot be constructed over all datasets in Table II, and the related experimental results will not be shown.

3) Parameters setting: For the default query, we set the size of the query attribute set $|Q_U| = 0.4 \cdot |\mathcal{A}_U|$ and $|Q_L| = 0.4 \cdot |\mathcal{A}_L|$. Also, we set the query integers $\alpha = 0.1 \cdot d^U_{max}$ and $\beta = 0.1 \cdot d^L_{max}$. To evaluate the effect of α and β, we set α from $0.05 \cdot d^U_{max}$ to $0.9 \cdot d^U_{max}$ and β from $0.05 \cdot d^L_{max}$ to $0.9 \cdot d^L_{max}$. For evaluating the effect of the size of query attribute set, we vary $|Q_U|$ as 20%, 40%, 60%, 80% of $|\mathcal{A}_U|$ and $|Q_L|$ as 20%, 40%, 60%, 80% of $|\mathcal{A}_L|$. The reported time and space under a given group of settings are obtained by averaging those from the corresponding generated queries.

B. Evaluation on the small dataset

We compare our proposed indexes with ABi-Index over a small **IMDB-60K** dataset ($|U| = 13.4K$, $|L| = 29.2K$, $|E| = 67.8K$, $|\mathcal{A}_U| = 6$, $|\mathcal{A}_L| = 7$), which is extracted from the IMDB. The result in Table III reveals that ABi-Index shows better query performance than other competitors, which is anticipated, as it can directly fetch the result subgraph. However, ABi-Index exhibits the poorest scalability among all indexes in terms of space and construction efficiency, since it needs to store all possible query results. Moreover, our MAC-Index not only indicates best performance in scalability, but also shows excellent query efficiency.

Index	Construction Time (s)	Index Size (MB)	Query Time (ms)
Online	–	–	247.7
ABi-Index	2884.4	1178.1	3.2
VC-Index	177.1	18.2	65.3
AC-Index	185.7	53.8	12.2
MAC-Index	219.8	2.1	16.1

TABLE III: Evaluation on the IMDB-60K Dataset

C. Query Efficiency

1) Performance over all datasets: The query efficiency over all dataset is presented in Fig. 8. Note that the approximate query algorithm based on MAC-Index is denoted as MAC-Index-A. We can see that the approximate approach shows a faster running time than its competitors, as it trades accuracy for better query efficiency. Also, both MAC-Index and AC-Index significantly outperform the other methods on query performance. The primary reason is that online approach incurs significant time overhead on the peeling process, and VC-Index requires substantial time to traverse each vertex. In addition, the query efficiency of MAC-Index and AC-Index is roughly equivalent, since the parameters λ_{ac} and λ_{mac} are comparable, as shown in Table II.

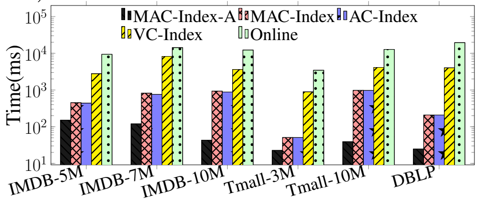


Fig. 8: Query performance over all datasets

2) Varying α and β: We evaluate the query performance by varying the values of α and β, and we report the results of three representative datasets: Tmall-3M, IMDB-10M and DBLP. The results on other datasets show similar trend. The result in Fig. 9 reveals that with the growth of α and β, the performance of the online algorithm is relatively stable, since the online algorithm always scans the entire original graph for each query. For our proposed indexes, they exhibit a downward trend in running time as the values of α and β increase, which is attributed to the smaller size of the intermediate subgraph, thus resulting in less runtime for the peeling process. In Fig. 10-11, we can also observe the similar trends in these settings.

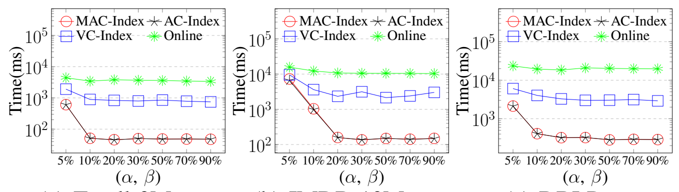
3) Varying the size of query attribute set: Fig. 12 presents the query performance by varying the size of the query attribute sets Q_U, Q_L. We can observe a considerable increase in query time for all indexes, primarily due to the growth in the size of the intermediate subgraph, which consumes more time for the peeling process. In contrast, online query is not sensitive to the size of the query attribute sets, owing to the peeling of the entire original graph from scratch. We can also see the similar trends in these settings in Fig. 10-11.

D. Index Construction & Maintenance

1) Construction performance on all datasets: In Fig. 15, we can observe that the construction efficiency among these indexes is very marginal, primarily because they all employ the bottom-up construction framework. Moreover, the construction time of the MAC-Index is slightly higher than that of AC-Index, which is also reasonable due to the computation of the attribute-dominant vertex.

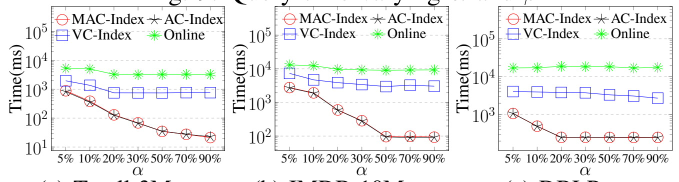
2) Construction performance varying graph size: We evaluate the scalability of our construction methods in Fig. 19. For each dataset, we randomly sample 20%, 40%, 60%, 80%, 100% of the edges from the original graph to perform the construction algorithm. As the graph size grows, the cost of all methods becomes larger since all methods rely on the graph size and the number of attribute types. In addition, the construction time of all methods is comparable, which corresponds to the results shown in Fig. 15.

3) Maintenance performance on all datasets: We report the maintenance performance of MAC-Index in Fig. 16. There are three algorithms: **MAC-Ins**, our maintenance algorithm for edge insertion; **MAC-Del**, our maintenance algorithm for



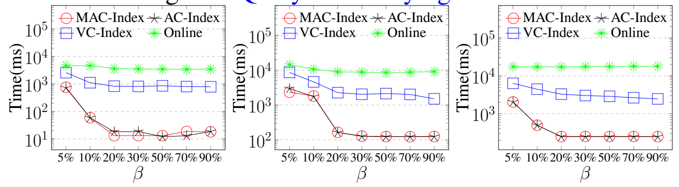
(a) Tmall-3M (b) IMDB-10M (c) DBLP

Fig. 9: Query time varying α and β



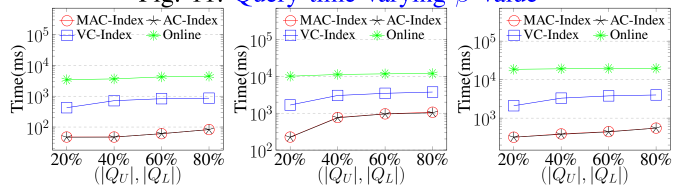
(a) Tmall-3M (b) IMDB-10M (c) DBLP

Fig. 10: Query time varying α value



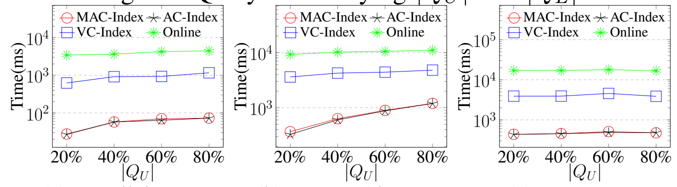
(a) Tmall-3M (b) IMDB-10M (c) DBLP

Fig. 11: Query time varying β value



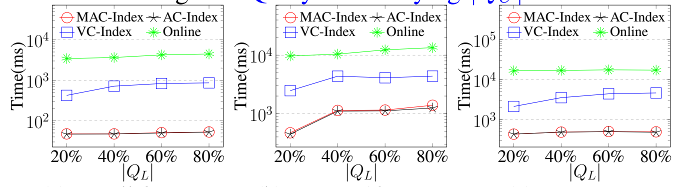
(a) Tmall-3M (b) IMDB-10M (c) DBLP

Fig. 12: Query time varying $|Q_U|$ and $|Q_L|$



(a) Tmall-3M (b) IMDB-10M (c) DBLP

Fig. 13: Query time varying $|Q_U|$



(a) Tmall-3M (b) IMDB-10M (c) DBLP

Fig. 14: Query time varying $|Q_L|$

edge deletion; **MAC-Constr**, index construction algorithm adapted for MAC-Index maintenance. For edge deletion, we randomly remove 1000 edges from the input graph and report the average time required for each edge deletion. Then, for testing edge insertion, we insert these removed edges into the graph and report the average time required for each edge insertion. The result in Fig. 16 shows that the running time of our proposed maintenance algorithm is significantly faster than

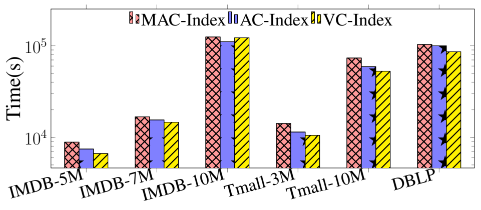


Fig. 15: Construction performance on all datasets

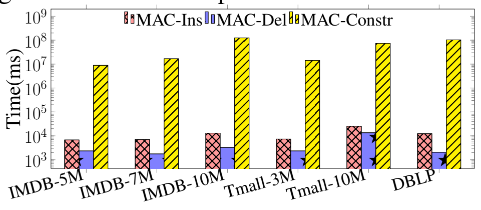


Fig. 16: Maintenance performance on all datasets

the basic solution. The primary reason is that we only perform a localized update on the index when update happens, avoiding unnecessary time overhead.

E. Index Size

1) Index size on all datasets: Fig. 17 reports the sizes of all indexes with the graph size as a comparison. As discussed in Section V, the value of ρ directly reflects the size of MAC-Index along with its compression effectiveness. We can see that MAC-Index is the most space-efficient and is comparable to the graph size. The primary reason is that ρ is a small number shown in Table II, approximately between 4 and 6 for each dataset. In addition, the size of VC-Index is smaller than AC-Index owing to the exponential complexity of AC-Index.

2) Index size varying graph size: For each dataset, we adopt the same sampling strategy as presented in Fig. 19. We can see from Fig. 20 that the space efficiency of MAC-Index significantly outperforms other indexes in all settings, demonstrating superior scalability. As the graph size grows, the size of all indexes becomes larger, since they are dependent on the graph size and the number of attribute types.

F. Further Scalability Analysis

Fig. 21 presents the performance with respect to the number of attribute types. The CiteSeer dataset ($|U| = 105.3K, |L| = 181.3K, |E| = 512.3K$) is obtained from KONECT [22], and we adopt a strategy used in previous work [15] to generate five synthetic attribute types in $U(G)$ (or $L(G)$) of varying sizes for CiteSeer: 4, 6, 8, 10 , and 12. As the attribute types grows, the construction time of ABi-Index increases sharply, while our proposed indexes exhibit slow growth. Moreover, ABi-Index and AC-Index exhibit exponential growth in space efficiency, while the others show a smooth increase. Regarding the query efficiency, all methods remain relatively stable.

Fig. 22 presents the performance by varying the graph density. According to the previous work [10], we generate five synthetic power-law (PL, for short) graphs ($|U| = 40K, |L| = 40K, |A_U| = 7, |A_L| = 7$) with average degrees of 5, 10, 15, 20, and 25, all following a power-law degree distribution. The result in Fig. 22a-22b shows that as the average degree increases, all indexes exhibit approximately linear growth in

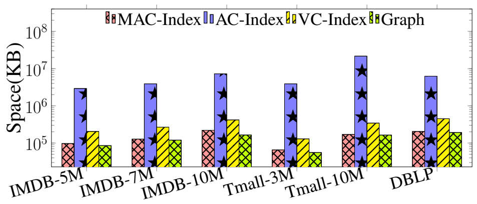


Fig. 17: Index size on all datasets

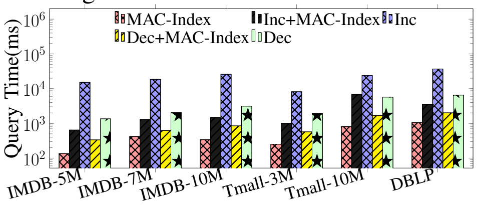
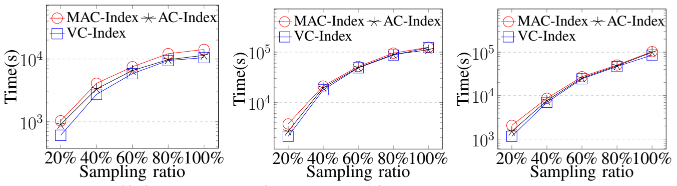
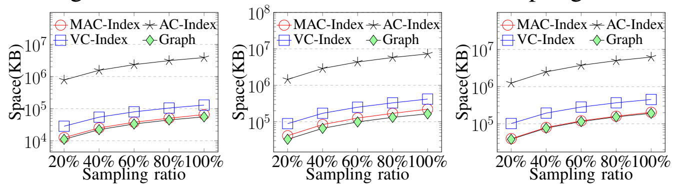


Fig. 18: Query performance of attributed-based approaches



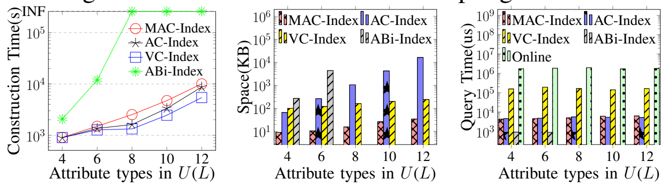
(a) Tmall-3M (b) IMDB-10M (c) DBLP

Fig. 19: Construction time for different sampling ratio



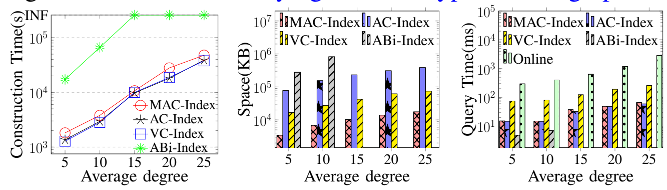
(a) Tmall-3M (b) IMDB-10M (c) DBLP

Fig. 20: Index size for different sampling ratio



(a) CiteSeer (b) CiteSeer (c) CiteSeer

Fig. 21: Performance varying attribute types of the graph

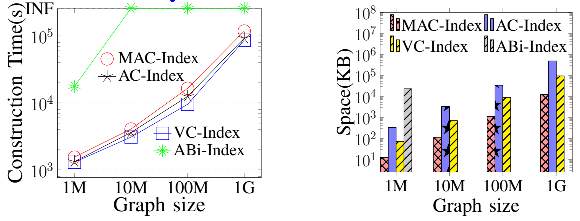


(a) PL (b) PL (c) PL

Fig. 22: Performance varying density of the graph

both construction time and space. Also, Fig. 22c reveals that all methods show an upward trend in query time.

Fig. 23 reports the scalability performance with respect to graph size. We apply a strategy from the previous works [10], [15] to generate synthetic graphs (SG, for short), where the attributes follow a specific uniform distribution and the average degree follows a power-law degree distribution. We generated four different synthetic graphs with sizes of 1M, 10M, 100M, 1G. As the graph size increases, our MAC-Index consistently outperforms ABi-Index and is comparable to the others on construction efficiency. In addition, our MAC-Index exhibits best performance on space efficiency, demonstrating its effective scalability.



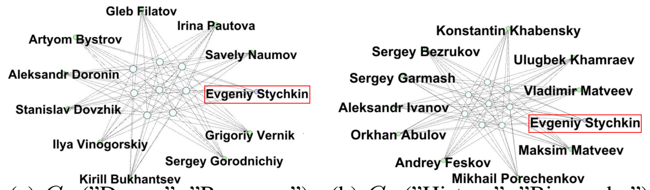
(a) SG (b) SG

Fig. 23: Scalability varying the large-scale graph size

G. Case Studies

To demonstrate the effectiveness of the SQAC problem in real applications, we conduct two case studies as follows:

① *Personalized recommendation.* We conduct a case study on the IMDB graph of the famous actor ”Evgeniy Stychkin”. According to the acting style of ”Evgeniy Stychkin”, we choose two groups of query attribute sets: {“History”, ”Biography”} and {”Drama”, ”Romance”}. Fig. 24 presents (7, 11)-cores of ”Evgeniy Stychkin”’s ego network on the IMDB graph. We can see that the resulting subgraphs show significant differences between the two query attribute sets.



(a) C_1 (”Drama”, ”Romance”) (b) C_2 (”History”, ”Biography”)

Fig. 24: Case study on IMDB graph ((7, 11)-core)

② *Accelerate the querying of the attributed communities.* We evaluate the query performance of the attributed community based on our MAC-Index. Xu et.al. [15] proposed two SOTA algorithms, namely Inc and Dec, for the computation of the attributed (α, β)-community. The Inc method has been thoroughly discussed in Section I. For the Dec method, it starts from a larger attribute-induced subgraph and checks whether the final community can be found. If not, it progressively narrows down to smaller attribute-induced graphs until the answer is discovered. The optimized algorithm employs MAC-Index to prune all vertices that cannot be the SQAC result, which can greatly narrow the search scope before computing the community. Regarding the query parameters, we set the default values of α and β to 8. Also, a query attribute set is configured as the set of attributes contained in the query vertex q, and another is configured as the set of attribute types among the neighbors of q. The result in Fig. 18 shows that our MAC-Index significantly outperforms comparative ones on query efficiency. Also, we can see that the time efficiency of the two algorithms incorporating our proposed MAC-Index, denoted as Inc+MAC-Index, Dec+MAC-Index, has improved significantly, demonstrating the effectiveness of pruning.

VII. Related Work

Dense Subgraph models Over Bipartite Graphs. There are many dense subgraph models in bipartite graphs, including (α, β)-core [6]–[10], [18], k-bitruss [24]–[26], and biclique [27]–[32]. The (α, β)-core is the extension of the k-core [33]–[36] in general graphs. Ahmed et al. [37] first introduce the concept of (α, β)-core. Ding et al. [16] propose a linear-time online approach for the computation of (α, β)-core. The SOTA work is proposed by [10], which utilizes the Bicore-Index to efficiently retrieve the (α, β)-core. Luo et al. [18] propose the SOTA algorithm for the maintenance of (α, β)-core. The k-bitruss is derived by extending the k-truss [38]–[41] to the bipartite graph. Note that, for the truss-like structure, it focuses on edge-centric cohesiveness, while the core-like structure is used to measure vertex cohesiveness. Wang et al. [25] propose the SOTA algorithm for the computation of k-bitruss. Like clique-like structure [42]–[45] in general graphs, biclique is a complete subgraph in a bipartite graph. Deng et al. [32] propose an efficient algorithm for the maintenance of top k (p, q)-bicliques over streaming bipartite graphs. However, all of these works do not consider the attributes of vertices, and their methods cannot be effectively applied to our problem.

Attributed Subgraph Query. In general graphs, Fang et al. [46] propose a community model based on k-core over vertex-attributed graphs, which aims to find a connected subgraph that meets both structural cohesiveness and attribute cohesiveness. Deng et al. [47] propose an efficient algorithm to identify a k-core subgraph that incorporates edge-labeled constraints. Guo et al. [48] introduce a k-core-based community, where the attribute constraint is based on multi-dimensional numerical attributes. Huang et al. [49] propose an attribute-driven community model that aims to find connected k-truss subgraphs with the largest attribute relevance score. Li et al. [50] aims to extract a connected k-truss that contains the query vertex, while its edges have maximum attribute similarity. However, these studies on k-core or k-truss structures are inherently difficult to adapt to our problem of (α, β)-core.

Also, several variations of attributed (α, β)-core have been formulated. As detailed in Section I, Xu et al. [15] propose an attributed (α, β)-community model, which is a subset of our proposed SQAC under the same query parameters. Li et al. [51] study a more densely (α, β)-attributed weight community, which considers the attribute score to measure the attribute cohesiveness of a bipartite graph. Wang et al. [11] study the significant (α, β)-community over edge-weighted bipartite graphs. Wu et al. [52] propose a rational community model in edge-attributed bipartite graphs. While, these works either impose strict attribute constraints on subgraphs, or rely heavily on edge attributes in querying. Hence, the exiting methods fail to be adaptive in our SQAC scenarios.

VIII. Conclusions & Future Works

In this paper, we investigated the problem of querying the (α, β)-core with attribute constraints over attributed bipartite graphs. We first proposed two basic index-based algorithms to support efficient querying for our SQAC. To further improve performance, we designed the MAC-Index, which not only significantly reduces the index size, but also guarantees efficient query performance. Additionally, efficient construction and maintenance algorithms were also proposed for MAC-Index. Extensive experiments over real-world datasets confirmed the efficiency and effectiveness of our index-based algorithms. Since the graph with enormous attribute types or high density may result in limited construction efficiency, and designing an effective parallel technique for index construction deserves more attention.

IX. AI-Generated Content Acknowledgment

It is important to mention that we have not employed any artificial intelligence (AI) techniques to produce any content in this paper.

References

[1] J. Wang, A. P. De Vries, and M. J. Reinders, "Unifying user-based and item-based collaborative filtering approaches by similarity fusion," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006, pp. 501–508.

[2] H. Wang, C. Zhou, J. Wu, W. Dang, X. Zhu, and J. Wang, "Deep structure learning for fraud detection," in *2018 IEEE international conference on data mining (ICDM)*. IEEE, 2018, pp. 567–576.

[3] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S.-M.-R. Beheshti, E. Bertino, and N. Foo, "Collusion detection in online rating systems," in *Web Technologies and Applications: 15th ASia-Pacific Web Conference, APWeb 2013, Sydney, Australia, April 4-6, 2013. Proceedings 15*. Springer, 2013, pp. 196–207.

[4] H. Deng, M. R. Lyu, and I. King, "A generalized co-hits algorithm and its application to bipartite graphs," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 239–248.

[5] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos, "Copycatch: stopping group attacks by spotting lockstep behavior in social networks," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 119–130.

[6] W. Bai, Y. Chen, D. Wu, Z. Huang, Y. Zhou, and C. Xu, "Generalized core maintenance of dynamic bipartite graphs," *Data Mining and Knowledge Discovery*, pp. 1–31, 2022.

[7] M. Cerinšek and V. Batagelj, "Generalized two-mode cores," *Social Networks*, vol. 42, pp. 80–87, 2015.

[8] Y. Guan, R. Lu, Y. Zheng, S. Zhang, J. Shao, and G. Wei, "Achieving efficient and privacy-preserving (α, β)-core query over bipartite graphs in cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 1979–1993, 2022.

[9] Q. Liu, X. Liao, X. Huang, J. Xu, and Y. Gao, "Distributed (α, β)-core decomposition over bipartite graphs," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 909–921.

[10] 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*, 2019, pp. 1130–1141.

[11] K. Wang, W. Zhang, X. Lin, Y. Zhang, L. Qin, and Y. Zhang, "Efficient and effective community search on large-scale bipartite graphs," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 85–96.

[12] Y. Zhang, K. Wang, W. Zhang, X. Lin, and Y. Zhang, "Pareto-optimal community search on large bipartite graphs," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 2647–2656.

[13] S. Gunnemann, E. Muller, S. Raubach, and T. Seidl, "Flexible fault tolerant subspace clustering for data with missing values," in *2011 IEEE 11th International Conference on Data Mining*. IEEE, 2011, pp. 231–240.

[14] A. K. Poernomo and V. Gopalkrishnan, "Towards efficient mining of proportional fault-tolerant frequent itemsets," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 697–706.

[15] Z. Xu, Y. Zhang, L. Yuan, Y. Qian, Z. Chen, M. Zhou, Q. Mao, and W. Pan, "Effective community search on large attributed bipartite graphs," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 37, no. 02, p. 2359002, 2023.

[16] D. Ding, H. Li, Z. Huang, and N. Mamoulis, "Efficient fault-tolerant group recommendation using alpha-beta-core," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 2047–2050.

[17] Y. Huang, C. Wang, J. Shi, and J. Shun, "Efficient algorithms for parallel bi-core decomposition," in *2023 Symposium on Algorithmic Principles of Computer Systems (APOCS)*. SIAM, 2023, pp. 17–32.

[18] W. Luo, Q. Yang, Y. Fang, and X. Zhou, "Efficient core maintenance in large bipartite graphs," *Proceedings of the ACM on Management of Data*, vol. 1, no. 3, pp. 1–26, 2023.

[19] "Imdb website," 2026. [Online]. Available: https://www.imdb.com/

[20] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks. corr," *arXiv preprint cs.DS/0310049*, vol. 37, 2003.

[21] "Konect website," 2026. [Online]. Available: https://www.tmall.com/

[22] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd international conference on world wide web*, 2013, pp. 1343–1350.

[23] "Codes," https://github.com/XueQingSBQ/SQAC, 2025.

[24] A. E. Sarıyüce and A. Pinar, "Peeling bipartite networks for dense subgraph discovery," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018, pp. 504–512.

[25] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Efficient bitruss decomposition for large-scale bipartite graphs," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 661–672.

[26] Z. Zou, "Bitruss decomposition of bipartite graphs," in *International conference on database systems for advanced applications*. Springer, 2016, pp. 218–233.

[27] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum biclique search at billion scale," *Proceedings of the VLDB Endowment*, 2020.

[28] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient maximal biclique enumeration for large sparse bipartite graphs," *Proceedings of the VLDB Endowment*, vol. 15, no. 8, pp. 1559–1571, 2022.

[29] J. Chen, K. Wang, R.-H. Li, H. Qin, X. Lin, and G. Wang, "Maximal biclique enumeration: A prefix tree based approach," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 2544–2556.

[30] R. Peeters, "The maximum edge biclique problem is np-complete," *Discrete Applied Mathematics*, vol. 131, no. 3, pp. 651–654, 2003.

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

[32] X. Deng, Z. Qin, P. Peng, and H. Zhou, "Topk-bc: Efficient maintenance of top k (p, q)-bicliques over streaming bipartite graphs," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2025, pp. 3696–3709.

[33] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 51–62.

[34] A. E. Sarıyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek, "Incremental k-core decomposition: algorithms and evaluation," *The VLDB Journal*, vol. 25, pp. 425–447, 2016.

[35] B. Sun, T.-H. H. Chan, and M. Sozio, "Fully dynamic approximate k-core decomposition in hypergraphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 14, no. 4, pp. 1–21, 2020.

[36] Y. Zhang and S. Parthasarathy, "Extracting analyzing and visualizing triangle k-core motifs within networks," in *2012 IEEE 28th international conference on data engineering*. IEEE, 2012, pp. 1049–1060.

[37] A. Ahmed, V. Batagelj, X. Fu, S.-H. Hong, D. Merrick, and A. Mrvar, "Visualisation and analysis of the internet movie database," in *2007 6th International Asia-Pacific Symposium on Visualization*. IEEE, 2007, pp. 17–24.

[38] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proceedings of the VLDB Endowment*, vol. 5, no. 9, 2012.

[39] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 1311–1322.

[40] E. Akbas and P. Zhao, "Truss-based community search: a truss-equivalence based indexing approach," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1298–1309, 2017.

[41] H. Chen, A. Conte, R. Grossi, G. Loukides, S. P. Pissis, and M. Sweering, "On breaking truss-based communities," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 117–126.

[42] L. Chang, "Efficient maximum k-defective clique computation with improved time complexity," *Proceedings of the ACM on Management of Data*, vol. 1, no. 3, pp. 1–26, 2023.

[43] L. Chang, R. Gamage, and J. X. Yu, "Efficient k-clique count estimation with accuracy guarantee," *Proceedings of the VLDB Endowment*, vol. 17, no. 11, pp. 3707–3719, 2024.

[44] L. Chang, "Efficient maximum clique computation and enumeration over large sparse graphs," *The VLDB Journal*, vol. 29, no. 5, pp. 999–1022, 2020.

[45] K. Wang, K. Yu, and C. Long, "Efficient k-clique listing: an edge-oriented branching strategy," *Proceedings of the ACM on Management of Data*, vol. 2, no. 1, pp. 1–26, 2024.

[46] Y. Fang, C. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *Proceedings of the VLDB Endowment*, 2016.

[47] X. Deng, P. Peng, C. Liu, X. Xie, H. Zhou, and Z. Qin, "Efficient indexing for label-constrained cohesive subgraph queries over large graphs," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2025, pp. 2135–2147.

[48] F. Guo, Y. Yuan, G. Wang, X. Zhao, and H. Sun, "Multi-attributed community search in road-social networks," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 109–120.

[49] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," vol. 10, no. 9, p. 949–960, May 2017.

[50] L. Li, Y. Zhao, S. Luo, G. Wang, and Z. Wang, "Efficient community search in edge-attributed graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 10, pp. 10 790–10 806, 2023.

[51] D. Li, X. Liang, R. Hu, L. Zeng, and X. Wang, "$(\alpha, \ \beta)$-awcs: $(\alpha, \ \beta)$-attributed weighted community search on bipartite graphs," in *IJCNN*. IEEE, 2022, pp. 1–8.

[52] Y. Wu, R. Sun, C. Chen, and X. Wang, "Efficient attribute (α, β)-core detection in large bipartite graphs (student abstract)," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, 2022, pp. 13 087–13 088.