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 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.

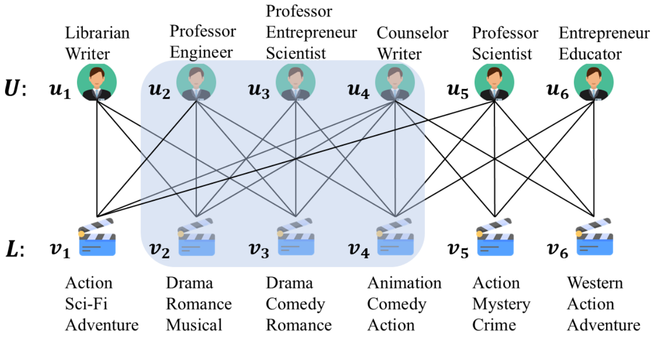


Fig. 1: A user-movie network

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, and the corresponding case studies are presented in Section VI-E. ① *Personalized group recommendation.* Consider a user-movie network G of IMDB (https://www.imdb.com) 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 $L(H) = \{v_2, v_3, v_4\}$. From Fig. 1, we can see 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.

[£] Corresponding author

② *Accelerate the querying of attributed community.* Another significant application of SQAC is to serve as a building block for complicated dense subgraph 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 given query attribute sets. Apparently, the attributed (α, β)-community is a subset of our SQAC under the same query conditions, as our SQAC problem 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, clearly 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 the attributed community search, which significantly speeds up query efficiency.

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, where m represents the number of edges and $\mathcal{A}_U$ (resp. $\mathcal{A}_L$) denotes the set of attribute types in U (resp. L); 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 [19], 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 integer pair (α, β). 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 compresses redundant information in the AC-Index leveraging the containment relationship of the attribute sets for (α, β)-cores. The MAC-Index significantly optimizes the space cost from $O(m \cdot (2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|}))$ of the AC-Index to $O(\rho \cdot n)$, while ensuring the query efficiency comparable to the AC-Index. Here, ρ denotes the average occurrence frequency of all vertices in G. For example, $\rho = 4.26$ for the real-world dataset DBLP in our experiment.

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, $\delta \leq \sqrt{m}$, λ_{ac} (resp. λ_{mac}) denotes the average occurrence frequency of all vertices in G' regarding $\mathcal{I}_{AC}$ (resp. $\mathcal{I}_{MAC}$), 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 $\mathcal{I}_{MAC}$.**

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

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 $\rho \ll n$ 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

In this section, we first give the formal definition of concepts in Section II-A. We then present an online query approach (Sections II-B) and a naive index (Sections II-C).

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.*

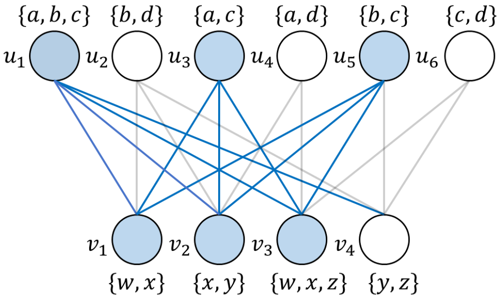


Fig. 2: An attributed bipartite graph G

Given an attributed bipartite graph $G = (V = (U \cup L), E, \mathcal{A}_U, \mathcal{A}_L)$ 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.*

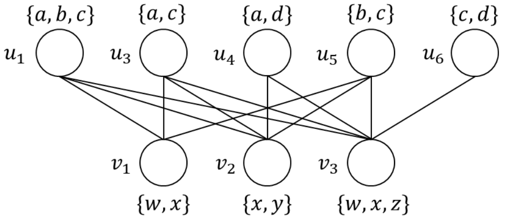


Fig. 3: The attribute-induced subgraph $G[\{a, c\}, \{w, x\}]$

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 among all vertices in $U(G)$ (resp. $L(G)$). We also use $D_{max}(G)$ to denote the maximum degree among all vertices in G, i.e., $D_{max}(G) = \max\{d^U_{max}(G), d^L_{max}(G)\}$.

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 $U(C_{\alpha,\beta})$ has a degree of at least α and each vertex in $L(C_{\alpha,\beta})$ has a degree of at least β within $C_{\alpha,\beta}(G)$. 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 $C_{\alpha,\beta}$.*

Note that, for a vertex u, there may exist multiple bi-core numbers in the graph G. All bi-core numbers can be computed by a peeling process that iteratively removes the vertices with the smallest degree (a.k.a. (α, β)-core decomposition), 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).*

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] has proposed an efficient online algorithm for the query of (α, β)-core over bipartite graphs, which can be easily adapted to the SQAC 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 remove the vertices that violate degree constraints 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 value of β for the vertex u and a specific α such that u is contained in the (α, β)-core of the graph 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 $\mathcal{I}_{Bi}^U$ and $\mathcal{I}_{Bi}^L$. We present only $\mathcal{I}_{Bi}^U$ as follows, since $\mathcal{I}_{Bi}^L$ is symmetrical to $\mathcal{I}_{Bi}^U$.

- The first level is an array containing d_{max}^U pointers to the arrays at the second level, where d_{max}^U denotes the maximum degree among all vertices in U.
- The second level is a sub-table containing d_{max}^U arrays. 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. Each vertex block $\mathcal{I}_{Bi}^U(\alpha, \beta)$ 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 $\mathcal{I}_{Bi}^U(\alpha, \beta')$ (resp. $\mathcal{I}_{Bi}^U(\alpha', \beta)$) 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.*

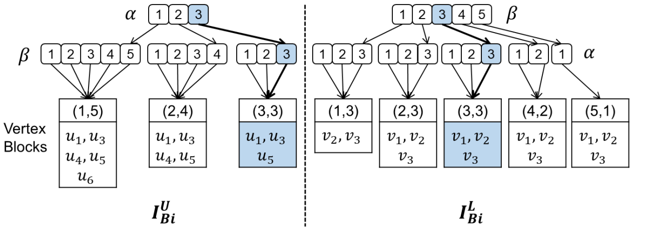


Fig. 4: The Bicore-Index $\mathcal{I}_{Bi}$ for $G[\{a, c\}, \{w, x\}]$ in Fig. 3

2) ABi-Index: Given that the SOTA solution Bicore-Index is proposed for the query of non-attributed (α, β)-core, 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 $G = (V = (U \cup L), E, \mathcal{A}_U, \mathcal{A}_L)$ 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. □

According to Lemma 1, we can easily infer that, 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 set over u and (α, β). 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 $A' \subset A$. Considering that there may be multiple minimal attribute-constrained sets, we use $MAS(u, (\alpha, \beta))$ to denote the set of the minimal attribute-constrained set over u and (α, β).*

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 determine whether 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. On one hand, VC-Index exploits attribute constraints to guarantee that each added attribute set is a MAS, thereby improving pruning efficiency. On the other hand, VC-Index makes use of the partial nested relationship of (α, β)-core to further compress storage redundancy. 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 $\mathcal{I}_{VC}^U$ and $\mathcal{I}_{VC}^L$. Each cell $\mathcal{I}_{VC}^U(u, (\alpha, \beta))$ in $\mathcal{I}_{VC}^U$ 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 ($\mathcal{I}_{VC}^L$ is symmetric to $\mathcal{I}_{VC}^U$):*

- *Attribute constraint. For each attribute set $A \in \mathcal{I}_{VC}^{U}(u, (\alpha, \beta))$, A must be a MAS for u and (α, β).*
- *Nested constraint. For each attribute set $A \in \mathcal{I}_{VC}^{U}(u, (\alpha, \beta))$, 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 $\mathcal{I}_{VC}^{U}$.*

For the sake of presentation only, we use $\mathcal{I}_{VC}^{U,\mathcal{A}_U}$ to denote such a two-dimensional matrix structure with each cell $\mathcal{I}_{VC}^{U,\mathcal{A}_U}(u, (\alpha, \beta))$ maintaining a set of attribute sets $A \in \mathcal{A}_U$ for u and (α, β). We have a symmetrical definition for $\mathcal{I}_{VC}^{U,\mathcal{A}_L}$. Note that, $\mathcal{I}_{VC}^{U}$ can be essentially viewed as a combination of $\mathcal{I}_{VC}^{U,\mathcal{A}_U}$ and $\mathcal{I}_{VC}^{U,\mathcal{A}_L}$. Moreover, we do not consider the case of $(\alpha, \beta) = (1, 1)$ for any vertex in $\mathcal{I}_{VC}$, since the query can be efficiently done by checking if the vertices have neighbors over query attribute sets. The space complexity of VC-Index is shown in Theorem 1.

Example 4. *Given the attributed bipartite graph in Fig. 3, we only present the partial VC-Index $\mathcal{I}_{VC}^{U,\mathcal{A}_U}$ in Fig. 5 due to space limit. The partial index $\mathcal{I}_{VC}^{U,\mathcal{A}_U}$ consists of a sequence of cells, where each cell $\mathcal{I}_{VC}^{U,\mathcal{A}_U}(u, (\alpha, \beta))$ maintains some attribute sets that meets both attribute and nested constraints over u and (α, β) pair.*

Theorem 1. *Given an attributed bipartite graph G, the size of $\mathcal{I}_{VC}$ is $O(n \cdot D_{max}^2 \cdot \mu)$, 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 $\sum_{u \in V(G)}(deg(u, G) \cdot D_{max}) \leq \sum_{u \in V(G)} D_{max}^2 = n \cdot D_{max}^2$. Hence, the space is bounded by $O(n \cdot D_{max}^2 \cdot \mu)$. □

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 $A_U, A_L \in \mathcal{I}_{VC}^{U}(u, (\alpha, \beta'))$ $(\beta \leq \beta' \leq \beta_{max}(u, \alpha))$ in $\mathcal{I}_{VC}$ such that $A_U \subseteq Q_U$ and $A_L \subseteq Q_L$, then u would not be in the $(\alpha$, $\beta)$-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, the VC-Index based query processing starts by traversing all vertices in $U(G)$, and pushes the candidate vertices that may be included in the final result into a temporary result set R. For each vertex $u \in U(G)$, it would take $O(\mu \cdot D_{max})$ time to check if u can be safely pruned. We then perform the symmetrical operation for all vertices in $L(G)$ and obtain the subgraph G' of G induced by R. For the sake of presentation, we refer to G' as the **intermediate subgraph**, which can be easily derived as the intersection of the $(\alpha$, $\beta)$-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. Hence, the overall time cost is $O(n \cdot \mu \cdot D_{max} + m')$.

Example 5. *Given a part of the $\mathcal{I}_{VC}$ in Fig. 5, two integers $\alpha = 3$, $\beta = 3$ and two query attribute sets $Q_U = \{a, c\}$, $Q_L = \{w, x\}$. For the vertex u_2, the maximum value of β is 3 for $\alpha = 3$. We can infer that u_2 is not in the $(3, 3)$-core of $G[\{a, c\}, \{w, x\}]$, since there does not exist any MAS in $\mathcal{I}_{VC}^{U,\mathcal{A}_U}(u_2, (3, 3))$ that is a subset of Q_U.*

C. Index Construction

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

Actually, in the real application, for two attribute sets $A_U, A'_U \in \mathcal{A}_U$ (resp. $\mathcal{A}_L$), if $A_U = \{A'_U \cup \{a\}\}$, where a denotes an individual attribute, we observe that the bi-core numbers of most vertices in $G[A_U, \mathcal{A}_L]$ are usually equivalent to that in $G[A'_U, \mathcal{A}_L]$. Consequently, it is natural for us to consider employing the $(\alpha$, $\beta)$-core maintenance technology of dynamic graphs to optimize the computation of bi-core number, thereby enhancing the 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. Note that both approaches will update the temporary $\mathcal{I}_{VC}$ during the computation of bi-core number. Although the time cost for edge insertion and edge deletion is the same according to the SOTA method of $(\alpha$, $\beta)$-core maintenance [18], the top-down approach evidently incurs extra time overhead. This primarily stems from the fact that the top-down approach needs to continuously update the temporary index during its execution, regardless of whether the vertex's bi-core number changes, since a smaller attribute set could be a 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 VC-Index $\mathcal{I}_{VC}$ is presented in Algorithm 1. We first perform the $(\alpha$, $\beta)$-core decomposition [16] for the subgraph $G[\{a\}, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, \{a\}]$) induced by each attribute $a \in \mathcal{A}_U$ (resp. $\mathcal{A}_L$) to obtain bi-core numbers, and then update the temporary $\mathcal{I}_{VC}$ (Lines 2-9). In particular, for each vertex u and its associated $(\alpha$, $\beta)$-core in $G[\{a\}, \mathcal{A}_L]$, the attribute set $\{a\}$ is obviously a MAS over u and $(\alpha$, $\beta)$ according to Definition 5. We can directly add $\{a\}$ into $\mathcal{I}_{VC}^{U}(u, (\alpha, \beta))$ if

	Vertex					
(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)	$\{a,b\},\{a,c\}$ $\{b,c\},\{b,d\}$ $\{c,d\}$	$\{a,b\},\{b,c\}$ $\{b,d\},\{c,d\}$	$\{a,b\},\{a,c\}$ $\{b,c\},\{c,d\}$	$\{a,b\},\{a,c\}$ $\{b,d\},\{c,d\}$	$\{a,b\},\{a,c\}$ $\{b,c\},\{b,d\}$ $\{c,d\}$	$\{b,c\},\{b,c\}$ $\{c,d\}$
(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)		$\{d\}$		$\{d\}$		$\{d\}$

(a) $\mathcal{I}_{VC}^{U,\mathcal{A}_U}$

	Vertex					
(α,β)	u_1	u_2	u_3	u_4	u_5	u_6
(4,2)	$\{x,z\},\{x,y\}$ $\{y,w\}$				$\{x,z\},\{x,y\}$ $\{y,w\}$	
(4,1)						
(3,3)	$\{x\},\{y,w\}$	$\{x,z\},\{x,y\}$ $\{y,w\}$	$\{x\},\{y,w\}$		$\{x\},\{y,w\}$	
(3,2)	$\{y,z\},\{z,w\}$				$\{y,z\},\{z,w\}$	
(3,1)						
(2,4)	$\{x\},\{y,z\}$ $\{z,w\},\{y,w\}$	$\{x\},\{y,z\}$ $\{z,w\},\{y,w\}$	$\{x\},\{y,z\}$ $\{z,w\},\{y,w\}$	$\{x\},\{y,z\}$ $\{z,w\},\{y,w\}$	$\{x\},\{y,z\}$ $\{z,w\},\{y,w\}$	$\{x,z\},\{x,y\}$ $\{y,z\},\{z,w\}$ $\{y,w\}$
(2,3)	$\{z,y,w\}$	$\{y\}$	$\{w\}$		$\{z,y,w\}$	$\{z\}$
(2,2)						
(2,1)						
(1,5)	$\{x,y,z,w\}$	$\{x\},\{y\}$	$\{x,y,z,w\}$	$\{x,y,z,w\}$	$\{x,y,z,w\}$	$\{x,z,w\}$
(1,4)		$\{z\},\{w\}$				$\{y\}$
(1,3)						
(1,2)						

(b) $\mathcal{I}_{VC}^{U,\mathcal{A}_L}$

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

$\{a\}$ also satisfies partial nested constraint in $\mathcal{I}_{VC}$ (Lines 4-6). After that, we adopt the bottom-up approach to iteratively derive the bi-core numbers of subgraphs induced by larger attribute sets (Lines 10-19). During the computation process, the temporary index also needs to be updated simultaneously based on Definition 6 (Lines 14-16). Finally, we can construct the $\mathcal{I}_{VC}$ when all possible attribute sets are verified. For the time complexity, computing all possible bi-core numbers for each vertex u in G would cost $O(n \cdot deg(u, G) \cdot D_{max})$ time. During the computation of MAS, checking whether an attribute set can be added to the index requires a comparison with the temporary attribute set in $\mathcal{I}_{VC}$, which takes $O(|\mathcal{I}_{VC}(u, (\alpha, \beta))|)$ time. Given that $deg(u, G) \leq D_{max}$, the total time cost is bounded by $O(n \cdot D_{max}^2 \cdot \mu)$.

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      foreach vertex $u \in G[\{a\}, \mathcal{A}_L]$ do
5          $B \leftarrow$ the set of bi-core numbers of $u$ in $G[\{a\}, \mathcal{A}_L]$;
6          Add $\{a\}$ into $\mathcal{I}_{VC}^{U}(u, (\alpha, \beta))$ if $\{a\}$ satisfies both
             attribute constraints and nested constraints ;
7      $\mathcal{C}_U.push(CD[\{a\}, \mathcal{A}_L])$ ;
8  foreach individual attribute $a' \in \mathcal{A}_L$ do
9      Do the operations that are symmetrical to those in
         Lines 3-7 in Algorithm 1;
10 while $\mathcal{C}_U \neq \emptyset$ do
11     $CD[A_U, \mathcal{A}_L] \leftarrow \mathcal{C}_U.pop()$;
12     foreach individual attribute $a \in \mathcal{A}_U$ do
13         $A'_U \leftarrow A_U \cup \{a\}$ ;
14         if the bi-core numbers of $G[A'_U, \mathcal{A}_L]$ have not been
             computed then
15             $CD[A'_U, \mathcal{A}_L] \leftarrow$ compute the bi-core numbers
                 of $G[A'_U, \mathcal{A}_L]$ by performing an updating
                 procedure for inserting the edges containing $a$
                 into $G[A_U, \mathcal{A}_L]$ ;
16             Update the temporary $\mathcal{I}_{VC}$ based on
                 Definition 6 ;
17             $\mathcal{C}_U.push(CD[A'_U, \mathcal{A}_L])$ ;
18 while $\mathcal{C}_L \neq \emptyset$ do
19     Do the operations that are symmetrical to those in
         Lines 11-17 in Algorithm 1;
20 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

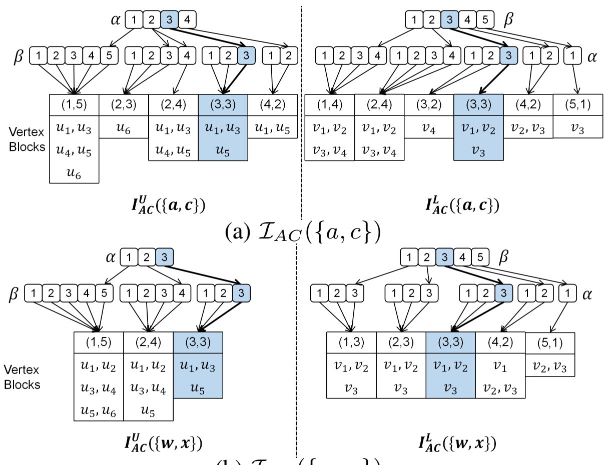
To eliminate the influence of factor n on the query efficiency of VC-Index, a natural idea is to optimize the query performance to be related to each query rather than the graph size. Considering the existing Bicore-Index [10] for the query of (α, β)-core over non-attributed bipartite graphs, its query 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 two types of attribute set, leading to extremely poor scalability; in contrast, the AC-Index focuses only on the subgraph induced by one type of attribute set. Therefore, the AC-Index can essentially be viewed as a one-dimensional version of the ABi-Index. 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 6. *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.



(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

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. □

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 final candidate vertex set R (i.e., $C_{\alpha,\beta}(G[Q_U, \mathcal{A}_L]) \cap C_{\alpha,\beta}(G[\mathcal{A}_U, Q_L])$). 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 the sake of presentation, 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., $\lambda_{ac} = \frac{\hat{n}'}{n'}$.

Example 7. *Considering the attributed bipartite graph in Fig. 2, a part of the $\mathcal{I}_{AC}$ in Fig. 6, two integers $\alpha = 3$, $\beta = 3$ and two query attribute sets $Q_U = \{a, c\}, Q_L = \{w, x\}$, the intermediate graph G' retrieved by $\mathcal{I}_{AC}(\{a, c\})$ and $\mathcal{I}_{AC}(\{w, x\})$, is $G' = \{U = \{u_1, u_3, u_5\}, L = \{v_1, v_2, v_3\}\}$. The corresponding result is marked in blue. Note that G' is also the answer, since all vertices in G' satisfy the degree constraints.*

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 (mentioned in Section III-C) to construct AC-Index. We first construct the Bicore-Index for the subgraph $G[\{a\}, \mathcal{A}_L]$ induced by each individual attribute $a \in \mathcal{A}_U$. Based on these smaller subgraphs that have been computed, we then perform insertion algorithm of (α, β)-core maintenance to compute the bi-core numbers for larger subgraphs and derive the corresponding Bicore-Index. Finally, to obtain a full $\mathcal{I}_{AC}$, we perform symmetrical operation for the subgraph induced by the attribute set $A_L \in \mathcal{A}_L$. During the computation of bi-core numbers, if the bi-core number of a vertex changes, it would take $O(\overline{d} \cdot \log \gamma)$ time to update the temporary $\mathcal{I}_{AC}$, where $\overline{d}$ denotes the average degree of each vertex in G, and γ denotes the average number of vertices for each vertex block in AC-Index. The total time cost is bounded by $O(n \cdot D_{max}^2 \cdot \overline{d} \cdot \log \gamma)$.

V. Minimized Attribute-based core Index

Although the AC-Index achieves a great improvement of query performance from $O(n \cdot \mu \cdot D_{max} + m')$ to $O(n' + m')$, it still suffers from poor scalability in terms of index size, which limits its applicability to large-scale datasets. Consequently, it is necessary to further compress redundant information in the AC-Index. In this section, we demonstrate the compressibility of vertex blocks in the AC-Index, proposing the minimized AC-Index, which not only significantly optimizes the space cost of AC-Index from $O(m \cdot (2^{|\mathcal{A}_U|} + 2^{|\mathcal{A}_L|}))$ to $O(\rho \cdot n)$, but also ensures query efficiency comparable to the AC-Index. Here, ρ denotes the average occurrence frequency of all vertices in G (ρ is a small integer and $\rho \ll n$ in practice).

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 $\mathcal{I}_{AC}$ according to the containment relationship of the attribute set for the (α, β)-core. 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 $\mathcal{I}^L_{AC}(A, (\alpha, \beta))$.*

Example 8. *Considering the AC-Index $\mathcal{I}_{AC}$ of the attributed bipartite 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 easily 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 of the subgraph $G[A, \mathcal{A}_L]$, we can also retrieve the result vertices $U(G')$ (resp. $L(G')$) from the attribute-dominant vertices in any vertex block $\mathcal{I}^{U}_{AC}(A',(\alpha,\beta'))$ with $A' \subseteq A$ and $\beta \leq \beta'$ (resp. $\mathcal{I}^{L}_{AC}(A',(\alpha',\beta))$ with $A' \subseteq A$ and $\alpha \leq \alpha'$).

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), which significantly compresses the space of $\mathcal{I}_{AC}$, while efficient query performance is guaranteed. We formally define the MAC-Index as follows.

Definition 9 (MAC-Index). *Given an attributed bipartite graph $G = (V = (U, L), E, \mathcal{A}_U, \mathcal{A}_L)$, the MAC-Index of G, formally denoted by $\mathcal{I}_{MAC}$, which just maintains the attribute-dominant vertices for each sub-index in the 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).*

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 $A \subseteq \mathcal{A}$. Additionally, we use $\mathcal{I}^{\mathcal{A}}_{MAC}$ to denote the set of $\mathcal{I}_{MAC}(A)$ with $A \in \mathcal{A}$.

Example 9. *Considering the attributed bipartite graph in Fig. 2, a part of the MAC-Index $\mathcal{I}_{MAC}$ is presented in Fig. 7. Compared to the $\mathcal{I}_{AC}$ in Fig. 6, we can see that, for the same attribute set and (α, β) pair, the corresponding vertex block in $\mathcal{I}_{MAC}$ contains fewer vertices. For example, $\mathcal{I}^{U}_{AC}(\{a,c\},(3,3)) = \{u_1,u_3,u_5\}$ and $\mathcal{I}^{U}_{MAC}(\{a,c\},(3,3)) = \emptyset$.*

Theorem 3. *Given a MAC-Index $\mathcal{I}_{MAC}$ of G, the index size is bounded by $O(\rho \cdot n)$, where ρ denotes the average occurrence frequency of all vertices in G within $\mathcal{I}_{MAC}$.*

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 a MAC-Index $\mathcal{I}_{MAC}$ of an attributed bipartite graph G, two integers α, β and two query attribute sets Q_U, Q_L, the MAC-Index essentially aims to find the same intermediate subgraph (i.e., $C_{\alpha,\beta}(G[Q_U,\mathcal{A}_L]) \cap C_{\alpha,\beta}(G[\mathcal{A}_U,Q_L])$) 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 $\mathcal{I}_{MAC}(Q_U)$. And, 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. We have a symmetrical case for the (α, β)-core$C_{\alpha,\beta}(G[\mathcal{A}_U,Q_L])$ in $G[\mathcal{A}_U,Q_L]$, thus, MAC-Index and AC-Index can find the same intermediate subgraph given identical query parameters.

Algorithm 2: Query based on MAC-Index

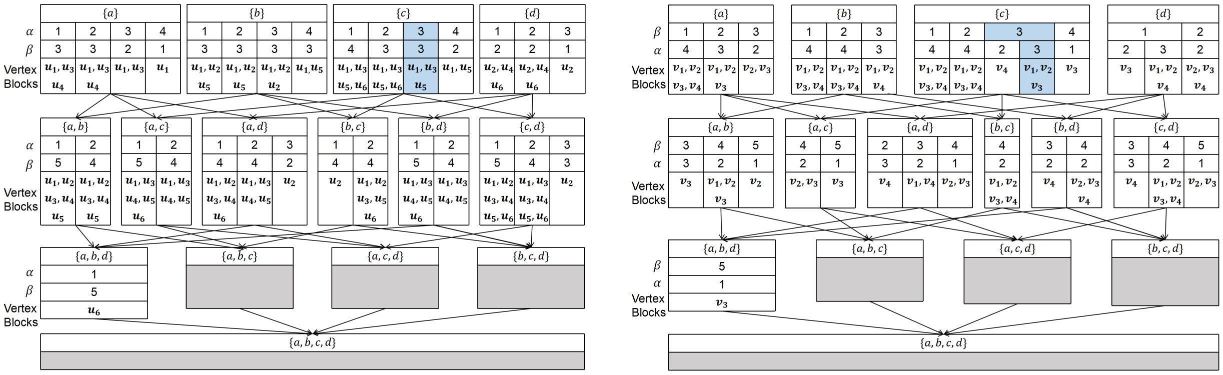
```
   Input: An attributed bipartite graph $G$, a MAC-Index $\mathcal{I}_{MAC}$
          of $G$, two integers $\alpha$, $\beta$ and two query attribute sets
          $Q_U$, $Q_L$
   Output: The result of SQAC
 1 $R \leftarrow \emptyset$ ;
 2 $\mathbb{Q}_U \leftarrow$ All possible subsets of $Q_U$;
 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 $\alpha' \geq \alpha$, we do the
         operations that are symmetrical to those in Line 5-7 in
         Algorithm 2 ;
 9 foreach attribute set $A \subseteq \mathbb{Q}_L$ do
10     Do the same operations as Lines 5-8 in Algorithm 2.
11 $G' \leftarrow$ the subgraph of $G$ induced by $R$ ;
12 Remove the vertices in $G'$ that do not satisfy degree
     constraints to obtain the 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 the vertex blocks $\mathcal{I}^{U}_{MAC}(A,(\alpha,\beta'))$ (resp. $\mathcal{I}^{L}_{MAC}(A,(\alpha',\beta))$) in $\mathcal{I}^{Q_U}_{MAC}$ with $A \subseteq Q_U$ and $\beta' \geq \beta$ (resp. $\alpha' \geq \alpha$) since these vertices could be a part of final result. We then perform a symmetrical operation for the vertex blocks in $\mathcal{I}^{Q_L}_{MAC}$ to retrieve the (α, β)-core of $G[\mathcal{A}_U,Q_L]$, and thus obtain the intermediate subgraph G' (Lines 9-11). Finally, we remove the vertices in G' that do not satisfy degree constraints 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 methods 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. We compare λ_{ac} with λ_{mac} under the default query, as shown in Table II. The results shows that λ_{mac} is nearly identical to λ_{ac}, demonstrating the superiority of MAC-Index interns of query efficiency.

B. The Construction for MAC-Index

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

The Framework. The pseudocode for MAC-Index construction is presented in Algorithm 3. Firstly, we compute the bi-core numbers for each subgraph $G[\{a\},\mathcal{A}_L]$ induced by the


{a}
α 1 2 3 4
β 3 3 2 1
Vertex Blocks u_1,u_3 u_4 | u_1,u_3 u_4 | u_1,u_3 | u_1
{b}
α 1 2 3 4
β 3 3 3 3
Vertex Blocks u_1,u_2 u_5 | u_1,u_2 u_5 | u_1,u_2 u_2 | u_1,u_5
{c}
α 1 2 3 4
β 4 3 3 2
Vertex Blocks u_1,u_3 u_5,u_6 | u_1,u_3 u_5,u_6 | u_1,u_3 u_5 | u_1,u_5
{d}
α 1 2 3
β 2 2 1
Vertex Blocks u_2,u_4 u_6 | u_2,u_4 u_6 | u_2
{a, b}
α 1 2
β 5 4
Vertex Blocks u_1,u_2 u_3,u_4 u_5 | u_1,u_2 u_3,u_4 u_5
{a, c}
α 1 2
β 5 4
Vertex Blocks u_1,u_3 u_4,u_5 u_6 | u_1,u_3 u_4,u_5
{a, d}
α 1 2 3
β 4 4 2
Vertex Blocks u_1,u_2 u_3,u_4 u_6 | u_1,u_3 u_4,u_5 | u_2
{b, c}
α 1 2
β 4 4
Vertex Blocks u_2 | u_1,u_2 u_3,u_5 u_6
{b, d}
α 1 2
β 5 4
Vertex Blocks u_1,u_3 u_4,u_5 u_6 | u_1,u_3 u_4,u_5
{c, d}
α 1 2 3
β 5 4 3
Vertex Blocks u_1,u_2 u_3,u_4 u_5,u_6 | u_1,u_3 u_3,u_4 u_5,u_6 | u_2
{a, b, d}
α 1
β 5
Vertex Blocks u_6
{a, b, c}
{a, c, d}
{b, c, d}
{a, b, c, d}
{a}
β 1 2 3
α 4 3 2
Vertex Blocks v_1,v_2 v_3,v_4 | v_1,v_2 v_3 | v_2,v_3
{b}
β 1 2 3
α 4 4 3
Vertex Blocks v_1,v_2 v_3,v_4 | v_1,v_2 v_3,v_4 | v_1,v_2 v_4
{c}
β 1 2 3 4
α 4 4 2 3 1
Vertex Blocks v_1,v_2 v_3,v_4 | v_1,v_2 v_3,v_4 | v_4 | v_1,v_2 v_3 | v_3
{d}
β 1 2
α 2 3 2
Vertex Blocks v_3 | v_1,v_2 v_4 | v_2,v_3 v_4
{a, b}
β 3 4 5
α 3 2 1
Vertex Blocks v_3 | v_1,v_2 v_3 | v_2
{a, c}
β 4 5
α 2 1
Vertex Blocks v_2,v_3 | v_3
{a, d}
β 2 3 4
α 3 2 1
Vertex Blocks v_4 | v_1,v_4 | v_2,v_3
{b, c}
β 4
α 2
Vertex Blocks v_1,v_2 v_3,v_4
{b, d}
β 3 4
α 2 2
Vertex Blocks v_4 | v_2,v_3 v_4
{c, d}
β 3 4 5
α 3 2 1
Vertex Blocks v_4 | v_1,v_2 v_3,v_4 | v_2,v_3
{a, b, d}
β 5
α 1
Vertex Blocks v_3
{a, b, c}
{a, c, d}
{b, c, d}
{a, b, c, d}

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

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 $\mathcal{C}_U, C_L, \mathcal{I}_{MAC} \leftarrow \emptyset$ ;
2 foreach individual attribute $a \in \mathcal{A}_U$ do
3     $CD[\{a\}, \mathcal{A}_L] \leftarrow$ compute bi-core numbers for $G[\{a\}, \mathcal{A}_L]$ ;
4     Construct the $\mathcal{I}^U_{MAC}(\{a\}, (\alpha, \beta))$ (resp. $\mathcal{I}^L_{MAC}(\{a\}, (\alpha, \beta))$) of the $\mathcal{I}_{MAC}$ according to Definition 8;
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 compute all attribute-dominant vertices ;
```

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 the bi-core numbers of $G[A'_U, \mathcal{A}_L]$ have not been computed then
7                  $V^* \leftarrow$ Vertices with bi-core number changed by inserting edges containing $a$ into $G[A_U, \mathcal{A}_L]$;
8                  foreach vertex $u \in V^* \wedge u \in U(G)$ do
9                      $(\alpha, \beta) \leftarrow$ the current bi-core number of the vertex $u$ ;
10                     foreach integer $\alpha' \leq \alpha$ do
11                         if $u$ is an attribute-dominant vertex for $A'_U$ and $(\alpha', \beta)$ based on Definition 8 then
12                             Add $u$ to $\mathcal{I}^U_{MAC}(A'_U, (\alpha', \beta))$ ;
13                 foreach vertex $v \in V^* \wedge v \in L(G)$ do
14                     Do the operation that are symmetrical to those in Lines 6-12 in Algorithm 4 ;
15                 $\mathcal{C}_U.push(CD[A'_U, \mathcal{A}_L])$ ;
16     while $\mathcal{C}_L \neq \emptyset$ do
17         Do the operations that are symmetrical to those in Lines 3-15 of Algorithm 4 ;
18     return $\mathcal{I}_{MAC}$ ;
```

individual attribute $a \in \mathcal{A}_U$ and construct the vertex block $\mathcal{I}^U_{MAC}(\{a\}, (\alpha, \beta))$ (resp. $\mathcal{I}^L_{MAC}(\{a\}, (\alpha, \beta))$) over $\{a\}$ and each possible (α, β) pair according to Definition 8 (Lines 2-5). Actually, the vertex blocks in $\mathcal{I}_{MAC}(\{a\})$ are identical to those in $\mathcal{I}_{AC}(\{a\})$ since $\{a\}$ can only be a subset of other sets; thus, each vertex in $\mathcal{I}_{AC}(\{a\})$ is an attribute-dominant vertex. We then perform symmetrical operations for each individual attribute $a' \in \mathcal{A}_L$ in Lines 6-7. Finally, we compute the remaining attribute-dominant vertices by invoking the procedure *ComputeADVertex* in Algorithm 4.

The Computation of Attribute-dominant Vertex. Similar to the AC-Index construction, we also employ the bottom-up approach to effectively derive the bi-core numbers of subgraph induced by larger attribute sets and simultaneously update the temporary MAC-Index. Specifically, during the process of computing the bi-core numbers of an attribute-induced subgraph $G[A'_U, \mathcal{A}_L]$, for a vertex $u \in U(G)$ (resp. $L(G)$) with a bi-core number changed (assume the current bi-core number is (α, β)), if u is an attribute-dominant vertex for A'_U and (α', β) pair with $\alpha' \leq \alpha$ based on Definition 8, then u would be added to the vertex block $\mathcal{I}^U_{MAC}(A'_U, (\alpha', \beta))$ (Lines 6-12 in Algorithm 4), which costs $O(\overline{d} \cdot \log n)$ time. Since there are $|deg(u, G) \cdot D_{max}|$ possible bi-core numbers for each vertex u in G, the time complexity of constructing $\mathcal{I}_{MAC}$ is bounded by $O(n \cdot D^2_{max} \cdot \overline{d} \cdot \log n)$.

C. Discussions on Maintenance of MAC-Index

Considering that the bipartite graph-structured data is constantly changing in the real world, forming the dynamic bipartite graphs. In this subsection, we propose the maintenance algorithm for the MAC-Index including edge insertion and edge deletion. When edge insertion happens, assume the incoming edge is $e_i = (u_i, v_i)$, with $A(u_i)$ and $A(v_i)$ denoting the set of attributes associated with u_i and v_i, respectively. Obviously, we only need to consider the affected subgraph $G[A, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, A]$)) induced by the attribute set A containing at least one attribute from $A(u_i)$ (resp. $A(v_i)$), i.e., $A \cap A(u_i) \neq \emptyset$. This is due to the bi-core numbers for the subgraphs induced by other attribute sets remaining unchanged. Next, with the properties of attribute-dominant vertices, we will adopt a bottom-up strategy to progressively update the vertex blocks of these affected subgraphs. From the smaller affected subgraph $G[A, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, A]$)), we first call the insertion algorithm of (α, β)-core maintenance [18] to obtain the vertices V^* whose bi-core number changes after the insertion of edge e_i. Then, for a vertex $u \in \{V^* \cap U(G)\}$ (resp. $v \in \{V^* \cap V(G)\}$) with a current bi-core number (α, β), if u is an attribute-dominant vertex for A and (α', β) pair with $\alpha' \leq \alpha$, we would add u to the vertex block $\mathcal{I}^U_{MAC}(A, (\alpha', \beta))$. Concurrently, to ensure the consistency of the index, if u is exactly included in $\mathcal{I}^U_{MAC}(A, (\alpha', \beta'))$ with $\beta' < \beta$ before, then u needs to be removed based on Definition 9. In this way, we can obtain the final index once all the affected subgraphs have been examined.

For an edge deletion, we also focus on the affected subgraph as in the case of edge insertion and employ the bottom-up approach to update the MAC-Index. From the smaller affected subgraph $G[A, \mathcal{A}_L]$ (resp. $G[\mathcal{A}_U, A]$)), we will first call the deletion algorithm of (α, β)-core maintenance to obtain the vertices V^* whose bi-core number changes. Then, for a vertex $u \in \{V^* \cap U(G)\}$ (resp. $v \in \{V^* \cap V(G)\}$) with a previous bi-core number (α', β') and a current bi-core number (α, β) after edge deletion, we would remove u from previous $\mathcal{I}^U_{MAC}(A, (\alpha', \beta'))$ since u would not be in the (α', β')-core of $G[A, \mathcal{A}_L]$. After that, if u is an attribute-dominant vertex for A and (α'', β) pair with $\alpha'' \leq \alpha$, then we would directly add u to the vertex block $\mathcal{I}^U_{MAC}(A, (\alpha'', \beta))$. Concurrently, to ensure the consistency of the index, if u is exactly included in $\mathcal{I}^U_{MAC}(A, (\alpha'', \beta''))$ with $\beta'' > \beta$ before, then u should be removed since u has not been an attribute-dominant vertex in $\mathcal{I}^U_{MAC}(A, (\alpha'', \beta''))$. We can obtain the final index, until all the affected subgraphs have been examined.

TABLE II: **Statistics of Datasets**

Datasets	$\|E\|$	$\|U\|$	$\|L\|$	$\|\mathcal{A}_U\|$	$\|\mathcal{A}_L\|$	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	✗

VI. Experimental Evaluation

In this section, we evaluate the performance of the proposed algorithms. We also conduct two case studies for our proposed SQAC. 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.

A. Setup

1) Datasets: Table II presents the statistics of datasets used in the experiments, where ρ denotes the average occurrence frequency of all vertices in G within $\mathcal{I}_{MAC}$, and λ_{ac} (resp. λ_{mac}) denotes the average occurrence frequency of all vertices in the intermediate subgraph regarding $\mathcal{I}_{AC}$ (resp. $\mathcal{I}_{MAC}$), under the default query (Section VI-A3). We extract three real graphs from IMDB (https://www.imdb.com) and two real graphs from Tmall (https://www.tmall.com/). 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 (e.g., ”actor”, ”producer”), and the attributes associated with 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 (e.g., ”click”, ”purchase”) and the attributes associated with a product represent its category. The **DBLP** data set is obtained from KONECT [21] and lacks 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 [22].

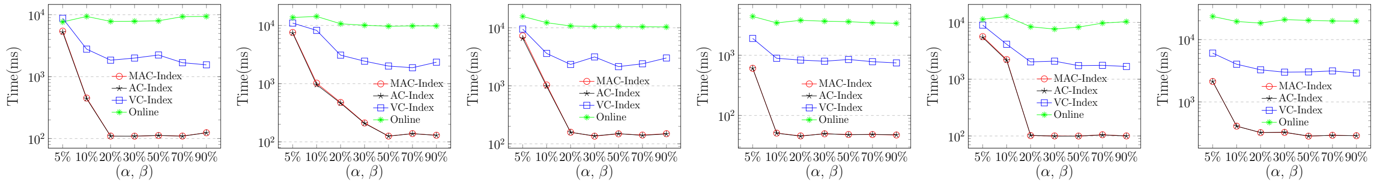
2) Algorithm: Our empirical studies are conducted based on following algorithm. There are two comparative algorithms in our work. The first is an online approach introduced in Section II-B, denoted as **Online**. The second is a SOTA solution, Bicore-Index, that is adapted to our problem (Section II-C), 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 ABi-Index cannot be constructed within 72 hours over all datasets, thus, the related experiment 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 integer $\alpha = 0.1 \cdot d^U_{max}$ 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. Query Efficiency

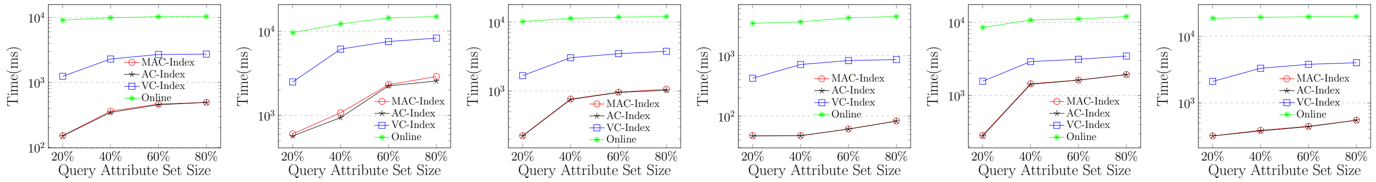
1) Varying α and β: We evaluate the query performance by varying the value of α and β over all datasets. Figure 8 shows that all index-based query algorithms significantly outperform the online algorithm in all settings. Also, we can observe that both AC-Index and MAC-Index demonstrate superior query performance compared to the VC-Index. Additionally, the query efficiency of AC-Index and MAC-Index is nearly identical since λ_{ac} and λ_{mac} are comparable (see Table II). 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, regardless of the value of α and β. For the VC-Index, it exhibits a downward trend in running time as the values of α and β increase, since the number of MASs for a vertex u and (α, β) will evidently decrease, resulting in lower time costs to check whether a vertex is in the (α, β)-core. Regarding the AC-Index and MAC-Index, they also show better query efficiency when the values of α and β get larger, which is primarily attributed to the size of the query result decreasing significantly as α and β grow, allowing them to quickly retrieve relevant vertices.

2) Varying the size of query attribute set: Fig. 9 presents the query performance by varying the size of the query attribute set. We can see that, our all index-based query algorithms consistently outperform online algorithm by 1∼2 orders of magnitude. In addition, both MAC-Index and AC-Index show



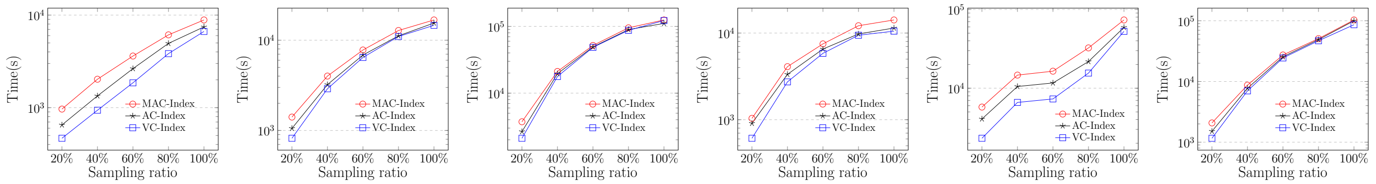
(a) IMDB-5M (b) IMDB-7M (c) IMDB-10M (d) Tmall-3M (e) Tmall-10M (f) DBLP

Fig. 8: Query time varying α and β



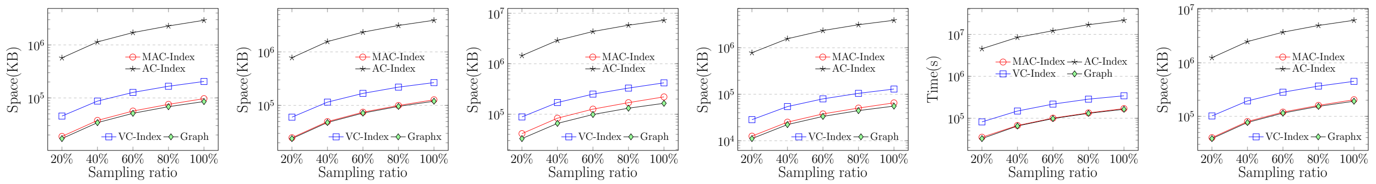
(a) IMDB-5M (b) IMDB-7M (c) IMDB-10M (d) Tmall-3M (e) Tmall-10M (f) DBLP

Fig. 9: Query time varying query attribute set size



(a) IMDB-5M (b) IMDB-7M (c) IMDB-10M (d) Tmall-3M (e) Tmall-10M (f) DBLP

Fig. 10: Construction performance for different sampling ratio



(a) IMDB-5M (b) IMDB-7M (c) IMDB-10M (d) Tmall-3M (e) Tmall-10M (f) DBLP

Fig. 11: Index size for different sampling ratio

better query efficiency than the VC-Index. When the size of the query attribute set increases, all index-based query algorithms show a slight upward trend on time cost, which is primarily due to the fact that the size of the intermediate subgraph increases, allowing more time overhead for the peeling process. In contrast, online query is not sensitive to the size of the query attribute set due to the peeling of the entire original graph from scratch.

C. Index Construction & Maintenance

1) Construction performance on all datasets: In Fig. 12, we evaluate the construction performance for AC-Index and MAC-Index. 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. 10. For each dataset, we randomly sample 20%, 40%, 60%, 80%, 100% of the edges from the original graph to perform the construction algorithm. We can see that the running time of all construction methods shows an increasing trend as the graph size grows, since these methods rely on the graph size and the number of attribute types in the datasets. In addition, the construction time of all methods is comparable, which corresponds to the results shown in Fig. 12.

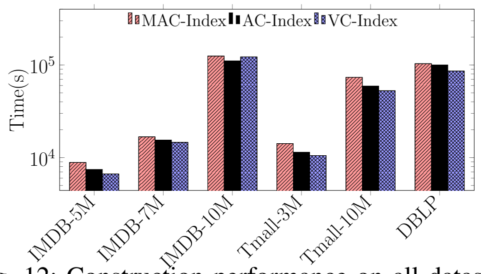


Fig. 12: Construction performance on all datasets

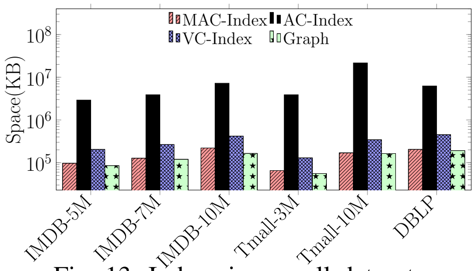


Fig. 13: Index size on all datasets

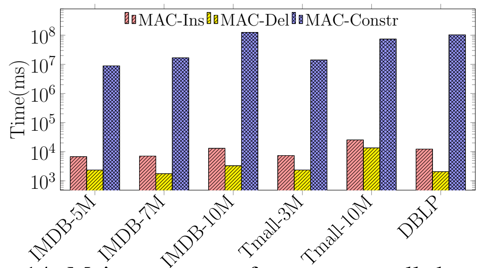


Fig. 14: Maintenance performance on all datasets

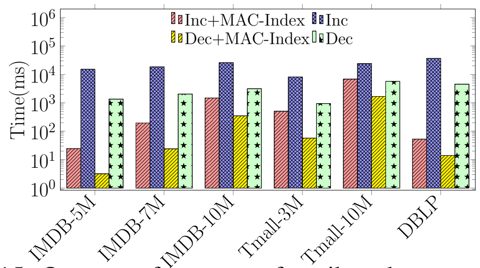


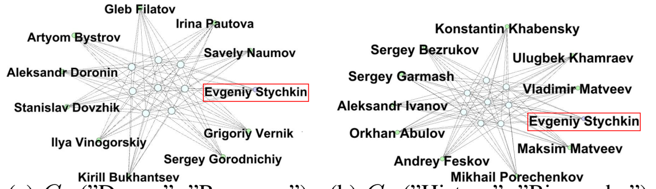
Fig. 15: Query performance of attributed communities

3) Maintenance performance on all datasets: We report the maintenance performance of MAC-Index in Fig. 14. Note that there are three algorithms for the maintenance of MAC-Index. The first is our proposed maintenance algorithm for edge insertion, denoted as **MAC-Ins**. The second is our proposed maintenance algorithm for edge deletion, denoted as **MAC-Del**. The last is the index construction algorithm adapted for the maintenance of MAC-Index, denoted as **MAC-Constr**. When evaluating the efficiency of edge deletion, we randomly remove 1000 edges from the input graph and report the average time required for each edge deletion. Then, for testing the edge insertion, we insert these removed edges into the graph and report the average time required for each edge insertion. From the result shown in Fig. 14, we can see that the running time of our proposed maintenance algorithm is significantly faster than the basic solution. The primary reason is that we only perform a localized update on the index during the edge insertion/deletion happens, thus avoiding unnecessary time overhead.

D. Index Size

1) Index size on all datasets: We compare the space overhead of different indexes and choose the size of the original graph as baseline. We can see from the results in Fig. 13 that the space efficiency of MAC-Index significantly outperforms the comparative indexes and is comparable to that of the original graph, since the factor ρ is a small integer, as shown in Table II. Apparently, the size of AC-Index is the largest among all indexes since it almost stores all query results. In addition, the space efficiency of VC-Index is better than that of AC-Index, as its size primarily depends on the graph size and the average number of MASs.

2) Index size varying graph size: We also evaluate the scalability of the index size by varying the graph size. For each dataset, we adopt the same sampling strategy as presented in Fig. 10. We can see from Fig. 11 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.



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

Fig. 16: Case study on IMDB graph ((7, 11)-core)

E. 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 set: {“History”, ”Biography”} and {”Drama”, ”Romance”}. Fig. 16 presents (7, 11)-cores of ”Evgeniy Stychkin”’s ego network on the IMDB graph under two different query attribute sets. We can see that the resulting subgraphs show significant differences between the two query attribute sets.
2) Accelerate the querying of the attributed communities. We evaluate the query performance of the attributed (α, β)-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 over attributed bipartite graphs. The Inc method has been thoroughly discussed in Section I. For the Dec method, the main idea is to start from a larger attribute-induced subgraph and check 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 in the result of the SQAC under the query attribute sets, 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 query attribute set is configured as the set of attribute types among the neighbors of q. The experiment result is presented in Fig. 15. We can see that the time efficiency of the two algorithms has improved significantly when incorporating our proposed SQAC, demonstrating the effectiveness of pruning.

VII. Related Work

In this section, we review several closely related works to our SQAC problem as follows:

Dense Subgraph models Over Bipartite Graphs. There are many dense subgraph models in bipartite graphs, including (α, β)-core [6]–[10], [18], k-bitruss [23]–[25], and biclique [26]–[31], which have attracted much attention in recent years.

The (α, β)-core is the extension of the k-core [32]–[35] in general graph. In [36], Ahmed et al. 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, which iteratively removes the vertices that violate the degree constraints over bipartite graphs. The SOTA work is proposed by [10], which utilizes the Bicore-Index to efficiently retrieve the (α, β)-core. In addition, Huang et al. [17] propose a parallel version and Liu et al. [9] design a distributed algorithm. Luo et al. [18] propose the SOTA algorithm for the maintenance of (α, β)-core over dynamic bipartite graphs. As another significant dense subgraph structure on bipartite graphs, the k-bitruss is derived by extending the k-truss [37]–[40] 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 the cohesiveness of vertex. Zou et al. [25] are the first to formulate the k-bitruss model in bipartite graphs. Wang et al. [24] propose the SOTA algorithm for the computation of k-bitruss, which utilizes the advance index to accelerate the query efficiency. Like clique-like structure [41]–[44] in general graphs, biclique is a complete subgraph in a bipartite graph. Chen et al. [28] propose a SOTA algorithm for maximal biclique enumeration over bipartite graphs, which has been proven to be NP-Hard [29]. Deng et al. [31] propose an efficient algorithm for the maintenance of top k (p, q)-bicliques over streaming bipartite graphs. However, all of these existing works do not consider the attributes of vertices and cannot be effectively applied to our problem.

Attributed Core-like Subgraph Query. Recently, many different types of attributed core-like subgraph models have been extensively studied over the past decade. In general graph, Fang et al. [45] propose a community model based on k-core over vertex-attributed graphs, which aims to find a connected subgraph that meets both structure cohesiveness and attribute cohesiveness (i.e., its vertices share the most attributes of given query attribute set). Deng et al. [46] propose an efficient algorithm to identify a k-core subgraph that incorporates edge-labeled constraints. While, these studies over k-core structure are inherently difficult to adapt to our problem of (α, β)-core.

In the attributed bipartite graphs, several variations of (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. And, our SQAC can serve as a building block in the computing process of such a community model. Li et al. [47] study a more densely (α, β)-attributed weight community model ((α, β)-AWC, for short), which considers the attribute score to measure the attribute cohesiveness of bipartite graph. Given a query attribute set Q, the attribute score of a vertex u is defined as the size of the intersection between the attribute set of u and Q, i.e., $|A(u) \cap Q|$. Also, the attribute score of a graph H is defined as the minimize value among the attribute scores of all vertices in H. The (α, β)-AWC is a maximal connected (α, β)-core that has maximal attribute scores. Under the same query attribute set, it is easy to determine that (α, β)-AWC is also a subset of our SQAC results due to their strict attribute and weight constraints on (α, β)-AWC. Wu et al. [48] propose a rational community model in edge-attributed bipartite graphs, which aims to retrieve the connected (α, β)-core with the largest rational score. While, the computation of rational score is heavily dependent on the edge attributes, thus, their method can hardly contribute to vertex-attributed graphs we focus on.

We can see that, none of the related work could efficiently solve our problem, which strengthens the novelty and importance of our work.

VIII. Conclusions

In this paper, we investigated the problem of querying the (α, β)-core with attribute constraints over attributed bipartite graphs. We are the first to propose the problem and existing studies over (α, β)-core disregard the vertex attributes, making it difficult for previous methods to adapt well to our problem. We proposed two 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 efficiency. 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.

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

[20] "Sqac: Scalable querying of attribute-constrained (α, β)-cores over large bipartite graphs [technical report]," 2025. [Online]. Available: https://github.com/XueQingSBQ/SQAC/blob/master/full.pdf

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

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

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

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

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

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

[46] 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.

[47] 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.

[48] 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.