Distributed (α, β)-Core Decomposition over Bipartite Graphs

Qing Liu[†§1], Xuankun Liao[†2], Xin Huang[†3], Jianliang Xu[†4], Yunjun Gao[§5]

[†]*Department of Computer Science, Hong Kong Baptist University, Hong Kong, China*
[§]*College of Computer Science, Zhejiang University, Hangzhou, China*
{[1]qingliu, [2]xkliao, [3]xinhuang, [4]xujl}@comp.hkbu.edu.hk, [5]gaoyj@zju.edu.cn

***Abstract*—(α, β)-core is an important cohesive subgraph model for bipartite graphs. Given a bipartite graph G, the problem of (α, β)-core decomposition is to compute non-empty (α, β)-cores for all possible values of α and β. The state-of-the-art (α, β)-core decomposition algorithm is a peeling-based algorithm, which iteratively deletes the vertex from high degree to low degree. However, as the peeling-based algorithm is designed for centralized environments, it cannot be applied to distributed environments, where graphs are partitioned and stored in different machines. Motivated by this, in this paper, we study the distributed (α, β)-core decomposition problem, aiming to develop new algorithms to support (α, β)-core decomposition in distributed environments. To this end, first, we analyze the local properties of (α, β)-core, and devise n-order Bi-indexes for the vertex, which are iteratively defined using the vertex neighbors' $(n-1)$-order Bi-indexes. Next, we propose an algorithm for (α, β)-core decomposition through iteratively calculating n-order Bi-indexes for every vertex. To further improve the efficiency of the algorithm, we propose two optimizations. Then, we extend our proposed algorithms to different distributed graph processing frameworks to make them run in distributed environments. Finally, extensive experimental results on both real and synthetic bipartite graphs demonstrate the efficiency of our proposed algorithms.**

I. Introduction

Bipartite graphs [1], [2], denoted by $G(U_G, V_G, E_G)$, consist of two different types of vertex sets U_G and V_G, and all edges of E_G connect one vertex type with the other. Many real-world applications can be modeled as bipartite graphs. For example, the movies rating graph [3] is a typical bipartite graph, where the vertices represent users and movies, respectively, and each edge represents a user's rating of a movie. Recently, a lot of research efforts have been devoted to the study of cohesive subgraph discovery over bipartite graphs, and many cohesive subgraph models have been proposed, such as (α, β)-core [3]–[5], biclique [6]–[10], and bitruss [11], [12]. In this paper, we mainly focus on the study of (α, β)-core due to its wide applications. Specifically, a (α, β)-core is the maximal subgraph H of the given bipartite graph G such that the vertices of U_H and V_H have at least α and β neighbors, respectively. For example, in Figure 1, the subgraph H induced by vertex set $\{u_1, u_2, u_4, v_1, v_2, v_3, v_4\}$ is a (3, 2)-core since the vertices of U_H and V_H have at least 3 and 2 neighbors, respectively. The (α, β)-core is widely used in applications of critical relationship identification [13], fault-tolerant group recommendation [14], community discovery [15], [16], and fraudsters detection [17].

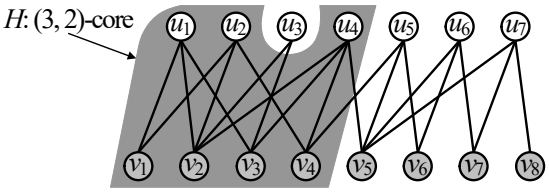


Fig. 1: A motivating example

Given a bipartite graph G, it may contain (α, β)-cores with different values of α and β. The problem of (α, β)-core decomposition aims to compute non-empty (α, β)-cores for all possible values of α and β. The (α, β)-core decomposition can be used to facilitate (α, β)-core-related applications. For example, to compute (α, β)-core for specific α and β values, a straightforward method is to traverse the whole graph to delete the vertices whose degrees do not satisfy the (α, β)-core constraints. This process could be time consuming. Instead, [17] and [15] use the (α, β)-core decomposition results to build indexes and perform the (α, β)-core computation through traversing the corresponding indexes, thus improving the efficiency. The state-of-the-art algorithm for (α, β)-core decomposition is a peeling-based algorithm [17]. The basic idea is to compute the $\beta_{\max,\alpha}(u)$ (resp. $\alpha_{\max,\beta}(v)$) for each possible value of α (resp. β) by iteratively deleting the vertices that violate the degree constraints. Here, $\beta_{\max,\alpha}(u)$ denotes the maximum value of β regarding the specific value of α such that u is in the corresponding (α, β)-core.

Motivations. Distributed graph processing has recently received considerable attention due to two facts [18]–[23]. First, in the real world, the graphs may be fragmented and distributed across multiple data centers. Second, the scale of graphs is growing exponentially, which raises two issues over a single machine: 1) the large-scale graph may be too large to fit into the memory; 2) the complexity of large-scale graph processing is high. For example, the time complexity of the peeling-based (α, β)-core decomposition algorithm is $O(m^{\frac{3}{2}})$ [17]. When m is large, the algorithm may not be scalable. In contrast, the distributed graph processing techniques can well handle the above problems. Therefore, we explore the (α, β)-core decomposition in distributed environments.

However, the existing peeling-based algorithm of (α, β)-core decomposition is designed for centralized environ-

ments [17], which cannot support distributed processing on partitioned graphs. Specifically, the peeling-based algorithm works in a sequential way, that is to maintain the degree information of the whole graph and iteratively delete the vertex from high degree to low degree until the graph is empty. However, in distributed environments, each machine has only local graph information. It is costly in terms of both time and communication to get the vertex with the minimum degree among different machines. In addition, many works have proposed different methods for distributed core decomposition, e.g., k-core [24]–[27] and D-core [28]. But, these methods also cannot be used to solve the distributed (α, β)-core decomposition problem since different core models have different semantics, which require specialized processing techniques. Hence, there is a need to propose new algorithms for distributed (α, β)-core decomposition.

Challenges and Our Solutions. When developing the algorithms for distributed (α, β)-core decomposition, we face two challenges. First, in distributed environments, communications occur among blocks (in a block-centric framework) or vertices (in a vertex-centric framework). Thus, we can only get the information of the vertices in the same block or from a vertex's neighbors. It is critical for us to leverage the local information to compute $\beta_{\max,\alpha}(u)$ or $\alpha_{\max,\beta}(v)$. Second, for many distributed graph processing frameworks, the operations are usually executed on vertices. Since a bipartite graph has two different types of vertices, there does not exist a unified operation for both types of vertices. Different operations should be designed for different types of vertices.

To tackle these challenges, we first reveal two relationships within (α, β)-core: (1) the relationship between $\beta_{\max,\alpha}(u)$ of u and $\alpha_{\max,\beta}(v)$ of u's neighbor v; and (2) the relationship between $\alpha_{\max,\beta}(v)$ of v and $\beta_{\max,\alpha}(u)$ of v's neighbor u. Based on that, we devise n-order Bi-indexes for vertices $u \in U_G$ and $v \in V_G$, respectively, which are iteratively defined using their neighbors' $(n-1)$-order Bi-indexes. The n-order Bi-indexes have a good property of convergence, i.e., the vertex's n-order Bi-indexes finally converge to $\beta_{\max,\alpha}(u)$ and $\alpha_{\max,\beta}(v)$ for u and v, respectively. Then, we propose an algorithm for (α, β)-core decomposition through iteratively calculating n-order Bi-indexes for all vertices. Moreover, we devise two optimizations to improve the efficiency of the proposed algorithm by introducing lower and upper bounds and leveraging intermediate results. It is worth mentioning that the Bi-indexes-based (α, β)-core decomposition algorithms only utilize the information of vertices' neighbors, which matches with the mechanism of distributed graph processing frameworks mentioned in the first challenge. On the basis of that, we implement the proposed algorithms using different distributed graph processing frameworks to show its flexibility.

Contributions. To be specific, we make the following contributions:

1) We propose the problem of distributed (α, β)-core decomposition, which is studied for the first time to the best of our knowledge.
2) We analyze the local property of (α, β)-core, based on which we propose n-order Bi-indexes for vertices and demonstrate the convergence of n-order Bi-indexes.
3) We develop an algorithm for (α, β)-core decomposition using n-order Bi-indexes and propose two optimizations to further improve the efficiency.
4) We extend the proposed algorithms to distributed graph processing frameworks and conduct extensive experiments on both real and synthetic bipartite graphs to demonstrate the efficiency of the proposed algorithms.

Paper Organization. Section II defines the problem. Section III presents the proposed algorithms and their distributed implementations. Section IV reports and analyzes experimental results. Finally, Section V reviews related work and Section VI concludes the paper.

II. Problem Formulation

A bipartite graph consists of two disjoint sets of vertices such that every edge only joins two vertices from different vertex sets. In this paper, we denote a bipartite graph by $G = (U_G, V_G, E_G)$, where U_G and V_G are two disjoint vertex sets and $E_G \subseteq U_G \times V_G$. We use u, v, and w to denote the vertices of U_G, V_G, and $U_G \cup V_G$, respectively. For a vertex w, we denote the neighbor of w by $\mathsf{N}_G(w) = \{w' | w' \in U_G \cup V_G, (w', w) \in E_G\}$. The degree of w is denoted by $\mathsf{deg}_G(w) = |\mathsf{N}_G(w)|$.

Definition 1. ((α, β)-core [14]). *Given a bipartite graph G, two integers α and β, the (α, β)-core of G is a maximal connected subgraph $H = (U_H, V_H, E_H) \subseteq G$ such that $\forall u \in U_H$, $\mathsf{deg}_H(u) \geq \alpha$, and $\forall v \in V_H$, $\mathsf{deg}_H(v) \geq \beta$.*

Take the subgraph H in Figure 1 as an example, $\forall u \in U_H$, $\mathsf{deg}_H(u) \geq 3$, and $\forall v \in V_H$, $\mathsf{deg}_H(v) \geq 2$. Thus, H is a (3, 2)-core. For a vertex w, it might be in multiple (α, β)-cores. It is cumbersome to record all the (α, β) pairs for w such that the corresponding (α, β)-core contains w. To this end, we introduce $\beta_{\max,\alpha}(u)$ and $\alpha_{\max,\beta}(v)$ [17].

Definition 2. ($\beta_{\max,\alpha}(u)$, $\alpha_{\max,\beta}(v)$). *Given a bipartite graph G and integers α and β,*
(1) $\beta_{\max,\alpha}(u) = \max\{\beta' \mid u \text{ is in a non-empty } (\alpha, \beta')\text{-core}\}$.
(2) $\alpha_{\max,\beta}(v) = \max\{\alpha' \mid v \text{ is in a non-empty } (\alpha', \beta)\text{-core}\}$.

In other words, $\beta_{\max,\alpha}(u)$ is the maximum value of β' w.r.t. the given integer α such that u is in the corresponding (α, β')-core. $\alpha_{\max,\beta}(v)$ is the maximum value of α w.r.t. the given integer β such that v is in the corresponding (α', β)-core. For example, in Figure 1, (1) for vertex u_1, let $\alpha = 3$, as $u_1 \in (3, 2)$-core but $u_1 \notin (3, 3)$-core, $\beta_{\max,3}(u_1) = 2$; (2) for vertex v_1, let $\beta = 2$, as $v_1 \in (3, 2)$-core but $v_1 \notin (4, 2)$-core, $\alpha_{\max,2}(v_1) = 3$. It is worth mentioning that $\beta_{\max,\alpha}(u)$ and $\alpha_{\max,\beta}(v)$ also can be extended to v and u, respectively.

According to Definition 2, for a vertex u and an integer α, u is contained in all (α, β)-cores with $\beta \leq \beta_{\max,\alpha}(u)$. Similarly, for a vertex v and an integer β, v is contained in all

TABLE I: (α, β)-core decomposition results

ID	$\mathcal{B}(u)$	ID	$\mathcal{A}(v)$
u_1	$\{(1,4),(2,3),(3,2)\}$	v_1	$\{(3,1),(3,2)\}$
u_2	$\{(1,4),(2,2),(3,2)\}$	v_2	$\{(4,1),(3,2),(2,3),(1,4)\}$
u_3	$\{(1,4),(2,3)\}$	v_3	$\{(4,1),(3,2),(2,3)\}$
u_4	$\{(1,4),(2,3),(3,2),(4,1)\}$	v_4	$\{(4,1),(3,2),(1,3)\}$
u_5	$\{(1,4),(2,2),(3,1)\}$	v_5	$\{(4,1),(2,2),(1,3),(1,4)\}$
u_6	$\{(1,4),(2,2),(3,1)\}$	v_6	$\{(3,1),(2,2)\}$
u_7	$\{(1,4),(2,2),(3,1)\}$	v_7	$\{(3,1),(2,2)\}$
		v_8	$\{(3,1)\}$

(α, β)-cores with $\alpha \leq \alpha_{\max,\beta}(v)$. Given a bipartite graph G, if we can compute the $\beta_{\max,\alpha}(u)$/$\alpha_{\max,\beta}(v)$ for all vertices and values of α/β, we can directly get (α, β)-cores of all possible (α, β) pairs for G. To this end, we formally define our studied problem as follows.

Problem 1. (Distributed (α, β)-core Decomposition). *Given a bipartite graph G, which has been partitioned into multiple subgraphs and distributed in different machines, the problem of distributed (α, β)-core decomposition is to collectively compute $\mathcal{B}(u) = \{(\alpha, \beta_{\max,\alpha}(u))|\forall \alpha \in [1, \mathsf{deg}_G(u)]\}$ and $\mathcal{A}(v) = \{(\alpha_{\max,\alpha}(v), \beta)|\forall \beta \in [1, \mathsf{deg}_G(v)]\}$ for every u and v, respectively.*

In Problem 1, $\mathcal{B}(u)$ and $\mathcal{A}(v)$ denote all possible $(\alpha, \beta_{\max,\alpha}(u))$ and $(\alpha_{\max,\alpha}(v), \beta)$ pairs for vertices u and v, respectively. For example, Table I shows the (α, β)-core decomposition results of bipartite graph in Figure 1. Note that for vertices u and v, the ranges of α and β are $[1, \mathsf{deg}_G(u)]$ and $[1, \mathsf{deg}_G(v)]$, respectively. It is because (1) vertex u can form $(1,1)$-core with any one of its neighbors, and can form $(\mathsf{deg}_G(u), 1)$-core with all of its neighbors, but cannot form $(\mathsf{deg}_G(u)+1, 1)$-core with any other vertices; (2) vertex v can form $(1,1)$-core with any one of its neighbors, and form $(1, \mathsf{deg}_G(v))$-core with all of its neighbors, but cannot form $(1, \mathsf{deg}_G(v)+1)$-core with any other vertices. In addition, u and v should be connected to the (α, β)-core. Hence, both α and β cannot be 0. Unless otherwise specified, we assume $\alpha \in [1, \mathsf{deg}_G(u)]$ and $\beta \in [1, \mathsf{deg}_G(v)]$ in the rest of the paper.

III. Distributed (α, β)-core Decomposition Algorithms

This section introduces our proposed distributed (α, β)-core decomposition algorithms. First, we analyze the relationships of $\beta_{\max,\alpha}(u)$ and $\alpha_{\max,\beta}(v)$ among neighboring vertices, based on which a new concept called n-order Bi-index is proposed (Section III-A). Then, we present an algorithm for (α, β)-core decomposition through n-order Bi-index computation (Section III-B), and corresponding optimizations (Section III-C). Since the computation of a vertex's n-order Bi-index only requires its neighbors' information, which matches with the mechanism of existing distributed graph systems, we introduce the distributed implementation of (α, β)-core decomposition algorithms (Section III-D).

A. n-order Bi-indexes

In distributed environments, we can only get local information directly. It is important to explore local information to facilitate graph analysis. Thus, we try to explore the relationships of $\beta_{\max,\alpha}(u)$ and $\alpha_{\max,\beta}(v)$ among neighboring vertices within a (α, β)-core. First, we introduce the concept of dominance. Given two pairs (α_1, β_1) and (α_2, β_2), and a set of (α, β) pairs P: (1) (α_1, β_1) dominates (α_2, β_2), denoted by $(\alpha_1, \beta_1) \succ (\alpha_2, \beta_2)$, iff (i) $\alpha_1 \geq \alpha_2$ and $\beta_1 \geq \beta_2$; and (ii) $\alpha_1 > \alpha_2$ or $\beta_1 > \beta_2$. (2) If $(\alpha_1, \beta_1) \succ (\alpha_2, \beta_2)$ or $(\alpha_1, \beta_1) = (\alpha_2, \beta_2)$, we denote it by $(\alpha_1, \beta_1) \succeq (\alpha_2, \beta_2)$. (3) If $\exists (\alpha, \beta) \in P$ such that $(\alpha, \beta) \succeq (\alpha_1, \beta_1)$, we denote it by $P \succeq (\alpha_1, \beta_1)$.

Theorem 1. *Given a bipartite graph G, two vertices u and v, two integers α and β.*

(1) *The vertex u has at least α neighbors v' such that $\exists(\alpha_{\max,\beta'}(v'), \beta') \in \mathcal{A}(v')$ satisfying $(\alpha_{\max,\beta'}(v'), \beta') \succeq (\alpha, \beta_{\max,\alpha}(u))$.*

(2) *The vertex v has at least β neighbors u' such that $\exists(\alpha', \beta_{\max,\alpha'}(u')) \in \mathcal{B}(u')$ satisfying $(\alpha', \beta_{\max,\alpha'}(u')) \succeq (\alpha_{\max,\beta'}(v), \beta)$.*

Proof. We first prove Theorem 1 (1). According to the given condition, u is in $(\alpha, \beta_{\max,\alpha}(u))$-core, denoted by H_1. In H_1, u has at least α neighbors v' whose degree is not less than $\beta_{\max,\alpha}(u)$. Let $\beta' = \beta_{\max,\alpha}(u)$, for these α neighbors v' of u, it must satisfy that $\alpha \leq \alpha_{\max,\beta'}(v')$. Hence, $(\alpha, \beta_{\max,\alpha}(u)) \preceq (\alpha_{\max,\beta'}(v'), \beta_{\max,\alpha}(u)) = (\alpha_{\max,\beta'}(v'), \beta')$. In the same way, we can prove Theorem 1 (2). Due to the space limitation, the details are omitted. □

For example, let $\alpha = 2$ for vertex u_1 in Figure 1. According to Table I, $\beta_{\max,2}(u_1) = 3$, and we can find two neighbors of u_1, i.e., v_2 and v_3, with $\mathcal{A}(v_2) \succeq (2,3)$ and $\mathcal{A}(v_3) \succeq (2,3)$. According to Theorem 1, given an integer α (resp. β), we can compute corresponding $\beta_{\max,\alpha}(u)$ (resp. $\alpha_{\max,\beta}(v)$) from $\mathcal{A}(v')$ of u's neighbors v' (resp. $\mathcal{B}(u')$ of v's neighbors u'). To this end, we define two operations as follows.

Definition 3. (α and β Operations). *Given a bipartite graph G, two vertices u and v, two integers α and β, let $S(u) = \{\mathcal{A}(v_i)|\forall v_i \in \mathsf{N}_G(u)\}$ and $S(v) = \{\mathcal{B}(u_i)|\forall u_i \in \mathsf{N}_G(v)\}$.*

(1) *The β operation over pairs set $S(u)$ w.r.t., α is defined as $\beta(\alpha, S(u)) = \max\{\beta' |$ there are at least α $\mathcal{A}(v_i) \in S(u)$ with $\mathcal{A}(v_i) \succeq (\alpha, \beta')\}$.*

(2) *The α operation over pairs set $S(v)$ w.r.t., β is defined as $\alpha(\beta, S(v)) = \max\{\alpha' |$ there are at least β $\mathcal{B}(u_i) \in S(v)$ with $\mathcal{B}(u_i) \succeq (\alpha', \beta)\}$.*

Obviously, $\beta(\alpha, S(u)) = \beta_{\max,\alpha}(u)$ and $\alpha(\beta, S(v)) = \alpha_{\max,\beta}(v)$. For example, let $\alpha = 3$ for u_1 in Figure 1. $S(u_1) = \{\mathcal{A}(v_1), \mathcal{A}(v_2), \mathcal{A}(v_3)\}$. According to Table I, $\mathcal{A}(v_1) \succeq (3,2)$ but $\mathcal{A}(v_1) \not\succeq (3,3)$, which also hold for $\mathcal{A}(v_2)$ and $\mathcal{A}(v_3)$. Therefore, $\beta(3, S(u_1)) = 2 = \beta_{\max,3}(u_1)$.

The above analysis is under the assumption that we have known the (α, β)-core decomposition results. However, given a bipartite graph, we only have the degree information of every

vertex and our goal is to compute the (α, β)-core decomposition results. Thus, combining Theorem 1, Definition 3 and the degree information of vertices, we propose a novel concept called n-order Bi-indexes for u and v as follows, which are defined in an iterative way.

Definition 4. (n-order Bi-indexes). *Given a bipartite graph G, two vertices u and v, two integers α and β, the n-order Bi-index of u w.r.t.,α is defined as follows:*

$$\mathsf{B}_U^{(n)}(\alpha, u) = \begin{cases} \max\limits_{\forall v_i \in \mathsf{N}_G(u)} \mathsf{deg}_G(v_i) & n = 0 \\ \beta(\alpha, S^{(n-1)}(u)) & n > 0 \end{cases} \tag{1}$$

where $S^{(n-1)}(u) = \{\mathcal{A}^{(n-1)}(v_i) | \forall v_i \in \mathsf{N}_G(u)\}$ and $\mathcal{A}^{(n-1)}(v_i) = \{(\mathsf{B}_V^{(n-1)}(\beta', v_i), \beta') | \forall \beta' \in [1, \mathsf{deg}_G(v_i)]\}$.

Similarly, the n-order Bi-index of v w.r.t., β is defined as follows:

$$\mathsf{B}_V^{(n)}(\beta, v) = \begin{cases} \max\limits_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i) & n = 0 \\ \alpha(\beta, S^{(n-1)}(v)) & n > 0 \end{cases} \tag{2}$$

where $S^{(n-1)}(v) = \{\mathcal{B}^{(n-1)}(u_i) | \forall u_i \in \mathsf{N}_G(v)\}$ and $\mathcal{B}^{(n-1)}(u_i) = \{(\alpha', \mathsf{B}_U^{(n-1)}(\alpha', u_i)) | \forall \alpha' \in [1, \mathsf{deg}_G(u_i)]\}$.

If the context is clear, we call n-order Bi-index of u and v for simplicity. The n-order Bi-indexes of u and v have following properties.

Theorem 2. *Given a bipartite graph G, the n-order and $(n+1)$-order Bi-indexes of vertices u and v satisfy,*

$$\mathsf{B}_U^{(n)}(\alpha, u) \geq \mathsf{B}_U^{(n+1)}(\alpha, u) \tag{3}$$

$$\mathsf{B}_V^{(n)}(\beta, v) \geq \mathsf{B}_V^{(n+1)}(\beta, v) \tag{4}$$

Proof. We prove Theorem 2 through mathematical induction. Since the proofs for u and v follow the same idea, we provide the proof for u in the following.

(1) $n = 0$. $\mathsf{B}_U^{(0)}(\alpha, u) = \max\limits_{\forall v_i \in \mathsf{N}_G(u)} \mathsf{deg}_G(v_i)$. $\mathsf{B}_U^{(1)}(\alpha, u) = \beta(\alpha, S^{(0)}(u))$, where $S^{(0)}(u) = \{\mathcal{A}^{(0)}(v_i) | \forall v_i \in \mathsf{N}_G(u)\}$ and $\mathcal{A}^{(0)}(v_i) = \{(\max\limits_{\forall u_j \in \mathsf{N}_G(v_i)} \mathsf{deg}_G(u_j), \beta) | \forall \beta\}$. If there exists α neighbors v' of u with $\mathcal{A}^{(0)}(v') \succeq (\alpha, \max\limits_{\forall v_i \in \mathsf{N}_G(u)} \mathsf{deg}_G(v_i))$, $\beta(\alpha, S^{(0)}(u)) = \max\limits_{\forall v_i \in \mathsf{N}_G(u)} \mathsf{deg}_G(v_i)$. Otherwise, $\beta(\alpha, S^{(0)}(u)) < \max\limits_{\forall v_i \in \mathsf{N}_G(u)} \mathsf{deg}_G(v_i)$. Thus, $\mathsf{B}_U^{(1)}(\alpha, u) = \beta(\alpha, S^{(0)}(u)) \leq \max\limits_{\forall v_i \in \mathsf{N}_G(u)} \mathsf{deg}_G(v_i) = \mathsf{B}_U^{(0)}(\alpha, u)$.

(2) Assume Theorem 2 holds when $n = m$, i.e., $\mathsf{B}_U^{(m)}(\alpha, u) \geq \mathsf{B}_U^{(m+1)}(\alpha, u)$ and $\mathsf{B}_V^{(m)}(\beta, v) \geq \mathsf{B}_V^{(m+1)}(\beta, v)$. $\mathsf{B}_U^{(m+1)}(\alpha, u)$ means u has at least α neighbors v_i with $\mathcal{A}^{(m)}(v_i) \succeq (\alpha, \mathsf{B}_U^{(m+1)}(\alpha, u))$. Since $\mathsf{B}_V^{(m)}(\beta, v) \geq \mathsf{B}_V^{(m+1)}(\beta, v)$, we consider two cases. (i) There are at least α neighbors v_i of u with $\mathcal{A}^{(m+1)}(v_i) \succeq (\alpha, \mathsf{B}_U^{(m+1)}(\alpha, u))$. Then, $\mathsf{B}_U^{(m+2)}(\alpha, u) = \beta(\alpha, S^{(m+1)}(u)) = \mathsf{B}_U^{(m+1)}(\alpha, u)$. (ii) There are less than α neighbors v_i of u with $\mathcal{A}^{(m+1)}(v_i) \succeq (\alpha, \mathsf{B}_U^{(m+1)}(\alpha, u))$. In order to find at least α neighbors v_i of u with $\mathcal{A}^{(m+1)}(v_i) \succeq (\alpha, \beta')$, β' should be less than $\mathsf{B}_U^{(m+1)}(\alpha, u)$. In other words, $\mathsf{B}_U^{(m+2)}(\alpha, u) = \beta' < \mathsf{B}_U^{(m+1)}(\alpha, u)$. Hence, $\mathsf{B}_U^{(m+1)}(\alpha, u) \geq \mathsf{B}_U^{(m+2)}(\alpha, u)$.

Combining (1) and (2), we arrive at $\mathsf{B}_U^{(n)}(\alpha, u) \geq \mathsf{B}_U^{(n+1)}(\alpha, u)$. □

Theorem 2 indicates that $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\beta, v)$ are non-increasing with the increase of n. Moreover, since $\mathsf{B}_U^{(n)}(\alpha, u) > 0$ and $\mathsf{B}_V^{(n)}(\beta, v) > 0$, $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\beta, v)$ can finally converge to an integer when n is big enough.

Theorem 3. *Given a bipartite graph G, the n-order Bi-indexes of vertices u and v satisfy:*

$$\lim_{n \to \infty} \mathsf{B}_U^{(n)}(\alpha, u) = \beta_{\max,\alpha}(u) \tag{5}$$

$$\lim_{n \to \infty} \mathsf{B}_V^{(n)}(\beta, v) = \alpha_{\max,\alpha}(v) \tag{6}$$

Proof. We prove Theorem 3 from two aspects.

(1) Let $U' = \{u' | \forall u' \in U_G, \mathcal{B}^{(\infty)}(u') \succeq (\alpha, \mathsf{B}_U^{(\infty)}(\alpha, u))\}$, $V' = \{v' | \forall v' \in V_G, \mathcal{A}^{(\infty)}(v') \succeq (\alpha, \mathsf{B}_U^{(\infty)}(\alpha, u))\}$, and $G' \subseteq G$ be the subgraph induced by vertices $U' \cup V'$. According to Definitions 3 and 4, G' is a $(\alpha, \mathsf{B}_U^{(\infty)}(\alpha, u))$-core. Thus, $\beta_{\max,\alpha}(u) \geq \mathsf{B}_U^{(\infty)}(\alpha, u)$.

(2) Let $U'' = \{u'' | \forall u'' \in U_G, \mathcal{B}(u'') \succeq \beta_{\max,\alpha}(u)\}$, $V'' = \{v'' | \forall v'' \in V_G, \mathcal{A}(v'') \succeq (\alpha, \beta_{\max,\alpha}(u)\}$, and $G'' \subseteq G$ be the subgraph induced by vertices $U'' \cup V''$. Obviously, for subgraph G'', (i) $\min\limits_{v'' \in V''} \mathsf{deg}_{G''}(v'') = \beta_{\max,\alpha}(u)$, and (ii) $\mathsf{B}_U^{(n)}(\alpha, u) \geq \min\limits_{v'' \in V''} \mathsf{deg}_{G''}(v'')$ for any n. Hence, $\mathsf{B}_U^{(\infty)}(\alpha, u) \geq \beta_{\max,\alpha}(u)$.

Therefore, $\mathsf{B}_U^{(\infty)}(\alpha, u) = \beta_{\max,\alpha}(u)$. In the same way, we can also prove that $\mathsf{B}_V^{(\infty)}(\beta, v) = \alpha_{\max,\alpha}(v)$. □

B. (α, β)-core Decomposition Algorithm

Based on Theorem 3, $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\beta, v)$ eventually converge to $\beta_{\max,\alpha}(u)$ and $\alpha_{\max,\alpha}(v)$, respectively. Moreover, the computation of $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\beta, v)$ only require $\mathcal{A}^{(n-1)}(v_i)$ of u's neighbors v_i and $\mathcal{B}^{(n-1)}(u_i)$ of v's neighbors u_i, respectively, which is easy to realize in distributed environments. Motivated by it, we propose a new algorithm for (α, β)-core decomposition. The basic idea is to iteratively compute the n-order Bi-indexes of u and v for all values of α and β, respectively, i.e., $\mathcal{B}^{(n)}(u)$ and $\mathcal{A}^{(n)}(v)$. When all n-order Bi-indexes converge, we get the final results.

Algorithm 1 shows the pseudo-code of (α, β)-core Decomposition. First, Algorithm 1 initializes $\mathcal{B}(u)$ and $\mathcal{A}(v)$ using 0-order Bi-indexes for every u and v (lines 1-6). Then, Algorithm 1 iteratively computes the n-order Bi-indexes for each vertex until it converges (lines 7-17). Note that the condition for judging convergence is $\mathcal{B}(u) = \mathcal{B}'(u)$ (lines 12-13) and $\mathcal{A}(v) = \mathcal{A}'(v)$ (lines 16-17). Next, we present the details of the n-order Bi-indexes computation algorithm.

Recall that, in Definition 3, $\beta(\alpha, S(u))$ is defined as the maximum of β' such that there are at least α $\mathcal{A}(v_i) \in S(u)$

Algorithm 1: (α, β)-core Decomposition

Input: bipartite graph G
Output: $\forall u$, $\mathcal{B}(u)$; $\forall v$, $\mathcal{A}(v)$

1 **for** $\forall u \in U_G$ **do**
2 **for** $\forall \alpha \in [1, \mathsf{deg}_G(u)]$ **do**
3 $\mathcal{B}(u) \leftarrow \mathcal{B}(u) \cup (\alpha, \max_{\forall v_i \in \mathsf{N}_G(u)} \mathsf{deg}_G(v_i))$;
4 **for** $\forall v \in V_G$ **do**
5 **for** $\forall \beta \in [1, \mathsf{deg}_G(v)]$ **do**
6 $\mathcal{A}(v) \leftarrow \mathcal{A}(v) \cup (\max_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i), \beta)$;
7 $Tag \leftarrow$ True;
8 **while** $Tag = True$ **do**
9 $Tag \leftarrow$ False;
10 **for** $\forall u \in U_G$ **do**
11 $\mathcal{B}'(u) \leftarrow$ compute n-order Bi-indexes of u w.r.t., $\forall \alpha \in [1, \mathsf{deg}_G(u)]$ using Algorithm 2;
12 **if** $\mathcal{B}(u) \neq \mathcal{B}'(u)$ **then**
13 $Tag \leftarrow$ True; $\mathcal{B}(u) \leftarrow \mathcal{B}'(u)$;
14 **for** $\forall v \in V_G$ **do**
15 $\mathcal{A}'(v) \leftarrow$ compute n-order Bi-indexes of v w.r.t., $\forall \beta \in [1, \mathsf{deg}_G(v)]$ using Algorithm 2;
16 **if** $\mathcal{A}(v) \neq \mathcal{A}'(v)$ **then**
17 $Tag \leftarrow$ True; $\mathcal{A}(v) \leftarrow \mathcal{A}'(v)$;
18 **return** $\mathcal{B}(u)$ and $\mathcal{A}(v)$ for all u and v;

Algorithm 2: n-order Bi-indexes Computation

Input: vertex w, $\mathcal{B}(\cdot)$, $\mathcal{A}(\cdot)$
Output: $\mathcal{C}(w)$

1 $\mathcal{C}(w) \leftarrow \varnothing$;
2 **for** $i \leftarrow \mathsf{deg}_G(w)$ **to** 1 **do**
3 **for** $j \leftarrow \max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w')$ **to** 1 **do**
4 **if** w *belongs to* U_G **then**
5 **for** $\forall w' \in \mathsf{N}_G(w)$ **do**
6 **if** $\mathcal{B}(w') \succeq (i, j)$ **then**
7 $V \leftarrow V \cup w'$;
8 **if** $|V| \geq i$ **then**
9 $\mathcal{C}(w) \leftarrow \mathcal{C}(w) \cup (i, j)$; **Break**;
10 **else**
11 **for** $\forall w' \in \mathsf{N}_G(w)$ **do**
12 **if** $\mathcal{A}(w') \succeq (j, i)$ **then**
13 $V \leftarrow V \cup w'$;
14 **if** $|V| \geq i$ **then**
15 $\mathcal{C}(w) \leftarrow \mathcal{C}(w) \cup (j, i)$; **Break**;
16 **return** $\mathcal{C}(w)$;

with $\mathcal{A}(v_i) \succeq (\alpha, \beta')$. A straightforward way to compute such qualified β' is to enumerate possible β' values in descending order. The first qualified β' value is just the result of $\beta(\alpha, S(u))$. Moreover, in Algorithm 1, $\mathcal{B}'(u)$ contains n-order Bi-indexes of u w.r.t., $\forall \alpha \in [1, \mathsf{deg}_G(u)]$. To compute $\beta(\alpha, S(u))$, we can first fix α, and then compute the n-order Bi-indexes of u through enumerating method mentioned above. Following this idea, we propose an algorithm for n-order Bi-indexes computation, whose pseudo-code is shown in Algorithm 2. Specifically, Algorithm 2 enumerates all possible values of i and j (lines 2-3). Then, Algorithm 2 checks whether the j is the n-order Bi-indexes of u (lines 4-7) or v (lines 10-13) w.r.t., i. If yes, (i, j) is added to the final pair set for u (lines 8-9), or (j, i) is added to the final pair set for v (lines 14-15).

We use the bipartite graph in Figure 1 to illustrate Algorithms 1 and 2. Table II shows $\mathcal{B}^{(n)}(u)$ and $\mathcal{A}^{(n)}(v)$ for each round. We take vertex u_1 as an example. In Figure 1, we can observe that $\mathsf{deg}_G(u_1) = 3$ and $\max_{\forall v \in \mathsf{N}_G(u_1)} \mathsf{deg}_G(v') = \mathsf{deg}_G(v_2) = 4$. So, $\mathsf{B}_U^{(0)}(3, u_1) = \mathsf{B}_U^{(0)}(2, u_1) = \mathsf{B}_U^{(0)}(1, u_1) = 4$ and $\mathcal{B}^{(0)}(u_1) = \{(1, 4), (2, 4), (3, 4)\}$. In the first iteration, we first check the pair $(3, 4)$ for u_1 and find that $\mathcal{A}(v_1) \not\succeq (3, 4)$, $\mathcal{A}(v_2) \succeq (3, 4)$, and $\mathcal{A}(v_3) \succeq (3, 4)$. Thus, $\mathsf{B}_U^{(1)}(3, u_1) \neq 4$. Then, we check pairs $(3, 3)$ and $(3, 2)$, and find that $\mathsf{B}_U^{(1)}(3, u_1) = 2$. $(3, 2)$ is added into $\mathcal{B}^{(1)}(u_1)$. In the same way, we can check the remaining pairs and get $\mathcal{B}^{(1)}(u_1) = \{(1, 4), (2, 3), (3, 2)\}$. In the next three rounds, we can compute $\mathcal{B}^{(2)}(u_1)$, $\mathcal{B}^{(3)}(u_1)$, and $\mathcal{B}^{(4)}(u_1)$, similarly. Since $\mathcal{B}^{(3)}(u) = \mathcal{B}^{(4)}(u)$ and $\mathcal{A}^{(3)}(v) = \mathcal{A}^{(4)}(v)$ for all u and v, respectively, the algorithm terminates. The final results are consistent with Table I.

Next, we analyze the bound of the number of iterations required for convergence. Given a bipartite graph G, we first introduce the concept of hierarchical sets, which is defined as follows. Specifically, the 0-th hierarchical set S_0 consists of the vertices with the minimum degree in G. The i-th hierarchical set S_i consists of the vertices with the minimum degree in the graph induced by vertex $(U_G \cup V_G) - \cup_{j<i} S_j$. Based on this concept, we have the following theorem.

Theorem 4. *The number of iterations needed for convergence is bounded by the number of hierarchical sets.*

Proof. Given a bipartite graph G, it is obvious that the vertices with the minimum degree converge within the first iteration according to the definition of Bi-indexes. Assume that $\forall j < i$, the hierarchical set S_j has converged. Then, we can delete the vertices $\cup_{j<i} S_j$ from G and the remaining vertices with the minimum degree, i.e., S_i, will converge in the next iteration. Therefore, in each iteration, at least one hierarchical set converges and the number of iterations needed for convergence is bounded by the number of hierarchical sets. □

Then, we give the time and space complexity of Algorithm 1. Let r be the number of iterations required for convergence, $n_U = |U_G|$, $n_V = |V_G|$, and $m = |E_G|$.

TABLE II: Illustrations of (α, β)-core decomposition algorithms on bipartite graph in Figure 1

ID	$\mathcal{B}^{(0)}(u)$	$\mathcal{B}^{(1)}(u)$	$\mathcal{B}^{(2)}(u)$	$\mathcal{B}^{(3)}(u)$	$\mathcal{B}^{(4)}(u)$
u_1	$\{(1,4),(2,4),(3,4)\}$	$\{(1,4),(2,3),(3,2)\}$	$\{(1,4),(2,3),(3,2)\}$	$\{(1,4),(2,3),(3,2)\}$	$\{(1,4),(2,3),(3,2)\}$
u_2	$\{(1,4),(2,4),(3,4)\}$	$\{(1,4),(2,3),(3,2)\}$	$\{(1,4),(2,3),(3,2)\}$	$\{(1,4),(2,2),(3,2)\}$	$\{(1,4),(2,2),(3,2)\}$
u_3	$\{(1,4),(2,4)\}$	$\{(1,4),(2,3)\}$	$\{(1,4),(2,3)\}$	$\{(1,4),(2,3)\}$	$\{(1,4),(2,3)\}$
u_4	$\{(1,4),(2,4),(3,4),(4,4)\}$	$\{(1,4),(2,4),(3,3),(4,1)\}$	$\{(1,4),(2,3),(3,2),(4,1)\}$	$\{(1,4),(2,3),(3,2),(4,1)\}$	$\{(1,4),(2,3),(3,2),(4,1)\}$
u_5	$\{(1,4),(2,4),(3,4)\}$	$\{(1,4),(2,3),(3,2)\}$	$\{(1,4),(2,2),(3,1)\}$	$\{(1,4),(2,2),(3,1)\}$	$\{(1,4),(2,2),(3,1)\}$
u_6	$\{(1,4),(2,4),(3,4)\}$	$\{(1,4),(2,2),(3,1)\}$	$\{(1,4),(2,2),(3,1)\}$	$\{(1,4),(2,2),(3,1)\}$	$\{(1,4),(2,2),(3,1)\}$
u_7	$\{(1,4),(2,4),(3,4)\}$	$\{(1,4),(2,2),(3,1)\}$	$\{(1,4),(2,2),(3,1)\}$	$\{(1,4),(2,2),(3,1)\}$	$\{(1,4),(2,2),(3,1)\}$
ID	$\mathcal{A}^{(0)}(v)$	$\mathcal{A}^{(1)}(v)$	$\mathcal{A}^{(2)}(v)$	$\mathcal{A}^{(3)}(v)$	$\mathcal{A}^{(4)}(v)$
v_1	$\{(3,1),(3,2)\}$	$\{(3,1),(3,2)\}$	$\{(3,1),(3,2)\}$	$\{(3,1),(3,2)\}$	$\{(3,1),(3,2)\}$
v_2	$\{(4,1),(4,2),(4,3),(4,4)\}$	$\{(4,1),(3,2),(2,3),(1,4)\}$	$\{(4,1),(3,2),(2,3),(1,4)\}$	$\{(4,1),(3,2),(2,3),(1,4)\}$	$\{(4,1),(3,2),(2,3),(1,4)\}$
v_3	$\{(4,1),(4,2),(4,3)\}$	$\{(4,1),(3,2),(2,3)\}$	$\{(4,1),(3,2),(2,3)\}$	$\{(4,1),(3,2),(2,3)\}$	$\{(4,1),(3,2),(2,3)\}$
v_4	$\{(4,1),(4,2),(4,3)\}$	$\{(4,1),(3,2),(3,3)\}$	$\{(4,1),(3,2),(1,3)\}$	$\{(4,1),(3,2),(1,3)\}$	$\{(4,1),(3,2),(1,3)\}$
v_5	$\{(4,1),(4,2),(4,3),(4,4)\}$	$\{(4,1),(3,2),(1,3),(1,4)\}$	$\{(4,1),(2,2),(1,3),(1,4)\}$	$\{(4,1),(2,2),(1,3),(1,4)\}$	$\{(4,1),(2,2),(1,3),(1,4)\}$
v_6	$\{(3,1),(3,2)\}$	$\{(3,1),(3,2)\}$	$\{(3,1),(2,2)\}$	$\{(3,1),(2,2)\}$	$\{(3,1),(2,2)\}$
v_7	$\{(3,1),(3,2)\}$	$\{(3,1),(2,2)\}$	$\{(3,1),(2,2)\}$	$\{(3,1),(2,2)\}$	$\{(3,1),(2,2)\}$
v_8	$\{(3,1)\}$	$\{(3,1)\}$	$\{(3,1)\}$	$\{(3,1)\}$	$\{(3,1)\}$
ID	$\mathcal{B}^{(0)}(v)$	$\mathcal{B}^{(1)}(v)$	$\mathcal{B}^{(2)}(v)$	$\mathcal{B}^{(3)}(v)$	$\mathcal{B}^{(4)}(v)$
v_1	$\{(1,2),(2,2),(3,2)\}$	$\{(1,2),(2,2),(3,2)\}$	$\{(1,2),(2,2),(3,2)\}$	$\{(1,2),(2,2),(3,2)\}$	$\{(1,2),(2,2),(3,2)\}$
v_2	$\{(1,4),(2,4),(3,4),(4,4)\}$	$\{(1,4),(2,3),(3,2),(4,1)\}$	$\{(1,4),(2,3),(3,2),(4,1)\}$	$\{(1,4),(2,3),(3,2),(4,1)\}$	$\{(1,4),(2,3),(3,2),(4,1)\}$
v_3	$\{(1,3),(2,3),(3,3),(4,3)\}$	$\{(1,3),(2,3),(3,2),(4,1)\}$	$\{(1,3),(2,3),(3,2),(4,1)\}$	$\{(1,3),(2,3),(3,2),(4,1)\}$	$\{(1,3),(2,3),(3,2),(4,1)\}$
v_4	$\{(1,3),(2,3),(3,3),(4,3)\}$	$\{(1,3),(2,3),(3,3),(4,1)\}$	$\{(1,3),(2,2),(3,2),(4,1)\}$	$\{(1,3),(2,2),(3,2),(4,1)\}$	$\{(1,3),(2,2),(3,2),(4,1)\}$
v_5	$\{(1,4),(2,4),(3,4),(4,4)\}$	$\{(1,4),(2,2),(3,2),(4,1)\}$	$\{(1,4),(2,2),(3,2),(4,1)\}$	$\{(1,4),(2,2),(3,1),(4,1)\}$	$\{(1,4),(2,2),(3,1),(4,1)\}$
v_6	$\{(1,2),(2,2),(3,2)\}$	$\{(1,2),(2,2),(3,2)\}$	$\{(1,2),(2,2),(3,2)\}$	$\{(1,2),(2,2),(3,1)\}$	$\{(1,2),(2,2),(3,1)\}$
v_7	$\{(1,2),(2,2),(3,2)\}$	$\{(1,2),(2,2),(3,1)\}$	$\{(1,2),(2,2),(3,1)\}$	$\{(1,2),(2,2),(3,1)\}$	$\{(1,2),(2,2),(3,1)\}$
v_8	$\{(1,1),(2,1),(3,1)\}$	$\{(1,1),(2,1),(3,1)\}$	$\{(1,1),(2,1),(3,1)\}$	$\{(1,1),(2,1),(3,1)\}$	$\{(1,1),(2,1),(3,1)\}$

Theorem 5. *The time and space complexity of Algorithm 1 is* $O(r \cdot m \cdot (\max\{n_U, n_V\})^2)$ *and* $O(m)$*, respectively.*

Proof. Algorithm 1 iteratively computes $\mathcal{B}^{(n)}(u)$ and $\mathcal{A}^{(n)}(v)$ until convergence. Thus, the time complexity of Algorithm 1 is determined by the number of iterations and the time required for each iteration. In each iteration, Algorithm 1 computes $\mathcal{B}^{(n)}(u)$ and $\mathcal{A}^{(n)}(v)$ for each u and v, respectively, using Algorithm 2. For each vertex $w \in U_G \cup V_G$, Algorithm 2 enumerates all possible pairs for examination, whose time complexity is $O(\mathsf{deg}_G(w) \cdot \max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w') \cdot \mathsf{deg}_G(w))$. Thus, the time for one iteration is $O(\sum_{w \in U_G \cup V_G} \mathsf{deg}_G(w) \cdot \max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w') \cdot \mathsf{deg}_G(w)) = O(m \cdot (\max\{n_U, n_V|\})^2)$. Therefore, the time complexity of Algorithm 1 is $O(r \cdot m \cdot (\max\{n_U, n_V\})^2)$.

Algorithm 1 should compute $\mathcal{B}(u)$ and $\mathcal{A}(v)$ for every u and v, respectively. $|\mathcal{B}(u)| = \mathsf{deg}_G(u)$ and $|\mathcal{A}(v)| = \mathsf{deg}_G(v)$. Hence, the space complexity of Algorithm 1 is $O(\sum_{w \in U_G \cup V_G} \mathsf{deg}_G(w)) = O(m)$. □

C. Optimized (α, β)-core Decomposition Algorithm

Theorem 5 shows that it is computationally expensive to enumerating all pairs for examination when computing n-order Bi-indexes. In this subsection, we propose two optimizations to improve the performance of (α, β)-core decomposition.

Optimization 1: Reducing Candidate Pairs Checking

Algorithm 2 should check $\mathsf{deg}_G(w) \times \max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w')$ candidate pairs for each vertex w in an iteration. Totally, for the entire bipartite graph, the algorithm should check $\sum_{w \in U_G \cup V_G} \mathsf{deg}_G(w) \times \max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w')$ candidate pairs in an iteration. However, we find that some pairs can be directly pruned without checking. To this end, we propose the first optimization via reducing candidate pairs checking. First, we present a theorem to support the first optimization.

Theorem 6. (1) $\mathsf{B}_U^{(n)}(\alpha+1, u) \leq \mathsf{B}_U^{(n)}(\alpha, u) \leq \mathsf{B}_U^{(n-1)}(\alpha, u)$*;*
(2) $\mathsf{B}_V^{(n)}(\beta+1, v) \leq \mathsf{B}_V^{(n)}(\beta, v) \leq \mathsf{B}_V^{(n-1)}(\beta, v)$*;*

Proof. We first prove Theorem 6 (1). $\mathsf{B}_U^{(n)}(\alpha, u) \leq \mathsf{B}_U^{(n-1)}(\alpha, u)$ has been proved in Theorem 2. We focus on the proof of $\mathsf{B}_U^{(n)}(\alpha+1, u) \leq \mathsf{B}_U^{(n)}(\alpha, u)$. According to Definition 4, $\mathsf{B}_U^{(n)}(\alpha+1, u)$ means that vertex u has at least $\alpha+1$ neighbors v_i with $\mathcal{A}^{(n)}(v_i) \succeq (\alpha+1, \mathsf{B}_U^{(n)}(\alpha+1, u))$. As $\alpha+1 > \alpha$, these neighbors v_i of u also satisfy $\mathcal{A}^{(n)}(v_i) \succeq (\alpha+1, \mathsf{B}_U^{(n)}(\alpha+1, u)) \succeq (\alpha, \mathsf{B}_U^{(n)}(\alpha+1, u))$. In other words, vertex u has at least α neighbors v_i with $\mathcal{A}^{(n)}(v_i) \succeq (\alpha, \mathsf{B}_U^{(n)}(\alpha+1, u))$. Thus, $\mathsf{B}_V^{(n)}(\alpha, v) \geq \mathsf{B}_V^{(n)}(\alpha+1, v)$. Theorem 6 (2) also can be proved in the same way. □

Theorem 6 points out the lower and upper bounds for $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\beta, v)$. Therefore, in line 3 of Algorithm 2, when enumerating j, we can reduce the range of

j from $[1, \max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w')]$ to lower and upper bounds. It is worth mentioning that the lower and upper bounds have already been calculated before computing $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\beta, v)$. So, Optimization 1 does not require other additional calculations and can have good improvement for Algorithm 1. Moreover, we would like to highlight two points. First, when $\alpha = \mathsf{deg}_G(w)$, the lower bound is set to $\max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w')$ since $\mathsf{B}_U^{(n)}(\alpha+1, u)$ does not exist. Second, if (i) the lower bound equals to upper bound, or (2) j equals to the lower bound, the n-order Bi-indexes, i.e., $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\beta, v)$, equal to the lower bound.

Optimization 2: Computing $\mathcal{A}(v)$ via $\mathcal{B}(v)$

The reason for traversing j in Algorithm 2 is that $\mathcal{B}^{(n)}(u)$ is α-dimension-centric while $\mathcal{A}^{(n)}(v)$ is β-dimension-centric. To be specific, from Table I, we can observe that the (α, β) pairs in $\mathcal{B}^{(n)}(u)$ and $\mathcal{A}^{(n)}(v)$ are continuous in α-dimension and β-dimension, respectively. Thus, we cannot directly compute the value of j in Algorithm 2. Motivated by it, if we can make $\mathcal{B}^{(n)}(u)$ and $\mathcal{A}^{(n)}(v)$ to be the same dimension-centric, it will benefit the computation of j. Motivated by it, we propose the second optimization.

First, we define $\mathcal{B}(v)$, which is modified from $\mathcal{B}(u)$ and $\mathcal{A}(v)$ in Definition 1. Specifically, $\mathcal{B}(v) = \{(\alpha, \beta_{\max,\alpha}(v)) | \forall \alpha \in [1, \max_{\forall u' \in \mathsf{N}_G(v)} \mathsf{deg}_G(u')]\}$. For $\mathcal{A}(v)$ and $\mathcal{B}(v)$, we have the following theorem.

Theorem 7. (1) *If we have $\mathcal{B}(v)$, $\forall \beta \in [1, \mathsf{deg}_G(v)]$,*

$$\alpha_{\max,\beta}(v) = \max\{\alpha' | \beta_{\max,\alpha'}(v) \geq \beta\} \tag{7}$$

(2) *If we have $\mathcal{A}(v)$, $\forall \alpha \in [1, \max_{\forall u' \in \mathsf{N}_G(v)} \mathsf{deg}_G(u')]$,*

$$\beta_{\max,\alpha}(v) = \max\{\beta' | \alpha_{\max,\beta'}(v) \geq \alpha\} \tag{8}$$

Proof. If $v \in (\alpha, \beta)$-core, it holds that $v \in (\alpha', \beta')$-core with $\alpha' \leq \alpha$ and $\beta' < \beta$. Combing the definitions of $\alpha_{\max,\beta}(v)$ and $\beta_{\max,\alpha}(v)$, we easily can get Equations 7 and 8. □

Obviously, Theorem 7 indicates that $\mathcal{A}(v)$ and $\mathcal{B}(v)$ are equivalent, and they can be converted to each other. Therefore, for Problem 1, we can first compute $\mathcal{B}(u)$ and $\mathcal{B}(v)$ for each u and v, and then convert $\mathcal{B}(v)$ to $\mathcal{A}(v)$, which is the basic idea of Optimization 2. To iteratively compute $\mathcal{B}(u)$ and $\mathcal{B}(v)$, we re-define the n-order Bi-indexes as follows.

Definition 5. *Given a bipartite graph G, two vertices u and v, an integer α, the n-order Bi-index of u w.r.t., $\alpha \in [1, \mathsf{deg}_G(u)]$, is defined as follows:*

$$\mathsf{B}_U^{(n)}(\alpha, u) = \begin{cases} \max_{\forall v_i \in \mathsf{N}_G(u)} \mathsf{deg}_G(v_i) & n = 0 \\ \beta(\alpha, S^{(n-1)}(u)) & n > 0 \end{cases} \tag{9}$$

where $S^{(n-1)}(u) = \{\mathcal{B}^{(n-1)}(v_i) | \forall v_i \in \mathsf{N}_G(u)\}$ and $\mathcal{B}^{(n-1)}(v_i) = \{(\alpha', \mathsf{B}_V^{(n-1)}(\alpha', v_i)) | \forall \alpha' \in [1, \max_{\forall u' \in \mathsf{N}_G(v_i)} \mathsf{deg}_G(u')]\}$.

The n-order Bi-index of v w.r.t., $\alpha \in [1, \max_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i)]$ is defined as follows:

$$\mathsf{B}_V^{(n)}(\alpha, v) = \begin{cases} \mathsf{deg}_G(v) & n = 0 \\ \gamma(\alpha, S^{(n-1)}(v)) & n > 0 \end{cases} \tag{10}$$

where $\gamma(\alpha, S^{(n-1)}(v)) = \max\{\beta' |$ there are at least β' $\mathcal{B}^{(n-1)}(u_i) \in S^{(n-1)}(v)$ such that $\mathcal{B}^{(n-1)}(u_i) \succeq (\alpha, \beta')\}$, $S^{(n-1)}(v) = \{\mathcal{B}^{(n-1)}(u_i) | \forall u_i \in \mathsf{N}_G(v)\}$, and $\mathcal{B}^{(n-1)}(u_i) = \{(\alpha', \mathsf{B}_U^{(n-1)}(\alpha', u_i)) | \forall \alpha' \in [1, \mathsf{deg}_G(u_i)]\}$.

Like the original n-order Bi-indexes in Definition 4, the above re-defined n-order Bi-indexes are convergent as well, which are shown in the following theorem.

Theorem 8.

$$\forall \alpha \in [1, \mathsf{deg}_G(u)], \lim_{n \to \infty} \mathsf{B}_U^{(n)}(\alpha, u) = \beta_{\max,\alpha}(u) \tag{11}$$

$$\forall \alpha \in [1, \max_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i)], \lim_{n \to \infty} \mathsf{B}_V^{(n)}(\alpha, v) = \beta_{\max,\alpha}(v) \tag{12}$$

Proof. Theorem 8 can be proved in the same way as Theorem 3. Due to the space limitation, we omit the details. □

Theorem 8 shows that $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\alpha, v)$ can finally converge to $\beta_{\max,\alpha}(u)$ and $\beta_{\max,\alpha}(v)$, respectively, indicating that we can compute $\mathcal{B}(u)$ and $\mathcal{B}(v)$ in an iterative manner. What's more, when $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\alpha, v)$ are of the same dimension centric, the computation of $\mathsf{B}_U^{(n)}(\alpha, u)$ and $\mathsf{B}_V^{(n)}(\alpha, v)$ are more convenient, as illustrated in Theorem 9.

Theorem 9. *For the n-order Bi-indexes of u and v defined in Definition 5, when $n > 0$,*

$$\mathsf{B}_U^{(n)}(\alpha, u) = \beta(\alpha, S^{(n-1)}(u)) = \mathbf{TOP}(\alpha, I) \tag{13}$$

where $\mathbf{TOP}(\alpha, I)$ denotes the top α-th value of integer set $I = \{\mathsf{B}_V^{(n-1)}(\alpha, v_i) | \forall v_i \in \mathsf{N}_G(u) \wedge \max_{\forall u' \in \mathsf{N}_G(v_i)} \mathsf{deg}_G(u') \geq \alpha\}$.

$$\mathsf{B}_V^{(n)}(\alpha, v) = \gamma(\alpha, S^{(n-1)}(v)) = \mathbf{H}(I) \tag{14}$$

where $\mathbf{H}(I) = \max\{i | I$ has i integers, whose value is not less than $i\}$ denotes the H-index [29] of integer set $I = \{\mathsf{B}_V^{(n-1)}(\alpha, u_i) | \forall u_i \in \mathsf{N}_G(v) \wedge \mathsf{deg}_G(u_i) \geq \alpha\}$.

Proof. We first prove Equation 13. According to Definitions 3 and 5, $\beta(\alpha, S^{(n-1)}(u))$ is the maximum value of β' such that u has at least α neighbors v_i with $(\alpha, \beta') \preceq \mathcal{B}^{(n-1)}(v_i)$. Since $(\alpha, \beta') \preceq \mathcal{B}^{(n-1)}(v_i) \Leftrightarrow (\alpha, \beta') \preceq (\alpha, \mathsf{B}_V^{(n-1)}(\alpha, v_i))$, $\beta(\alpha, S^{(n-1)}(u))$ is equivalent to find the maximum value of β' such that u has at least α neighbors v_i with $(\alpha, \beta') \preceq (\alpha, \mathsf{B}_V^{(n-1)}(\alpha, v_i))$. Therefore, β' is the top α-th value of $\mathsf{B}_V^{(n-1)}(\alpha, v_i)$ of all u's neighbors v_i.

Then, we prove Equation 14. According to Definitions 5, $\gamma(\alpha, S^{(n-1)}(v))$ is the maximum value of β' such that v has at least β' neighbors u_i with $\mathcal{B}^{(n-1)}(u_i) \succeq (\alpha, \beta')$.

Algorithm 3: Optimized n-order Bi-indexes Computation

Input: vertex w, $\mathcal{B}(\cdot)$
Output: $\mathcal{C}(w)$

1 $\mathcal{C}(w) \leftarrow \varnothing$;
2 **if** $w \in U_G$ **then**
3 **for** $i \leftarrow \mathsf{deg}_G(w)$ **to** 1 **do**
4 **if** $i = \mathsf{deg}_G(w)$ *or* $\mathsf{B}_U^{(n)}(i+1, w) < \mathsf{B}_U^{(n-1)}(i, w)$ **then**
5 $I \leftarrow \{\mathsf{B}_V^{(n-1)}(i, w') | \forall w' \in \mathsf{N}_G(w)\}$;
6 $j \leftarrow \mathbf{TOP}(\alpha, I)$;
7 **else**
8 $j \leftarrow \mathsf{B}_U^{(n)}(i+1, w)$;
9 $\mathcal{C}(w) \leftarrow \mathcal{C}(w) \cup (i, j)$;
10 **else**
11 **for** $i \leftarrow \max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w')$ **to** 1 **do**
12 **if** $i = \max_{\forall w' \in \mathsf{N}_G(w)} \mathsf{deg}_G(w')$ *or* $\mathsf{B}_V^{(n)}(i+1, w) < \mathsf{B}_V^{(n-1)}(i, w)$ **then**
13 $I \leftarrow \{\mathsf{B}_U^{(n-1)}(i, w') | \forall w' \in \mathsf{N}_G(w)\}$;
14 $j \leftarrow \mathbf{H}(I)$;
15 **else**
16 $j \leftarrow \mathsf{B}_V^{(n)}(i+1, w)$;
17 $\mathcal{C}(w) \leftarrow \mathcal{C}(w) \cup (i, j)$;
18 **return** $\mathcal{C}(w)$;

Since $(\alpha, \beta') \preceq \mathcal{B}^{(n-1)}(u_i) \Leftrightarrow (\alpha, \beta') \preceq (\alpha, \mathsf{B}_U^{(n-1)}(\alpha, u_i))$, $\gamma(\alpha, S^{(n-1)}(v))$ is equivalent to find the maximum value of β' such that v has at least β' neighbors u_i with $(\alpha, \beta') \preceq (\alpha, \mathsf{B}_U^{(n-1)}(\alpha, u_i))$. In other words, β' is the H-index of integer set consisting of all $\mathsf{B}_U^{(n-1)}(\alpha, u_i)$.

Moreover, the integer sets I of Equations 13 and 14 contain degree constraints. It is because (1) for Equation 13, if $\max_{\forall u' \in \mathsf{N}_G(v_i)} \mathsf{deg}_G(u') < \alpha$, $\mathsf{B}_V^{(n-1)}(\alpha, v_i)$ does not exist; (2) for Equation 14, if $\mathsf{deg}_G(u_i) < \alpha$, $\mathsf{B}_V^{(n-1)}(\alpha, u_i)$ does not exist. □

Optimized (α, β)-core Decomposition Algorithm. Combining Optimizations 1 and 2, we propose an optimized (α, β)-core decomposition algorithm. Since the pseudo-code of optimized (α, β)-core decomposition algorithm is similar to that of Algorithm 1, we omit the pseudo-code and mainly discuss three differences compared with Algorithm 1.

First, for initialization of 0-order Bi-index for vertices $v \in V_G$ (lines 4-6 of Algorithm 1), we should use the degree of v according to Equation 10.

Second, the computation of n-order Bi-indexes is different (lines 11 and 15 of Algorithm 1). The pseudo-code of the optimized n-order Bi-indexes computation is shown in Algorithm 3, which integrates both Optimizations 1 and 2. Specifically, when computing n-order Bi-indexes for each value i, (1) if lower bound (i.e., $\mathsf{B}_U^{(n)}(i+1, w)$) equals to upper bound (i.e., $\mathsf{B}_U^{(n-1)}(i, w)$), the n-order Bi-indexes equal to the lower bound (lines 7-8, 15-16); (2) otherwise, Algorithm 3 computes the n-order Bi-indexes using $\mathbf{TOP}(\alpha, I)$ operation and $\mathbf{TOP}(\alpha, I)$ operation for u and v, respectively (lines 4-6, 12-14). Note that the range of i for u and v are different (lines 3 and 11).

Algorithm 4: Converting $\mathcal{B}(v)$ to $\mathcal{A}(v)$

Input: $\mathcal{B}(v)$,
Output: $\mathcal{A}(v)$

1 $j \leftarrow 1$;
2 **for** $i \leftarrow \mathsf{deg}_G(v)$ **to** 1 **do**
3 **while** $\beta_{\max,j}(v) \geq i$ **do**
4 $j \leftarrow j + 1$;
5 $\mathcal{A}(v) \leftarrow \mathcal{A}(v) \cup (j-1, i)$;
6 **return** $\mathcal{A}(v)$;

Third, after n-order Bi-indexes are converged, the optimized algorithm should convert $\mathcal{B}(v)$ to $\mathcal{A}(v)$ (between lines 17 and 18 of Algorithm 1). Algorithm 4 shows the corresponding pseudo-code. To be specific, for each $i \in [1, \mathsf{deg}_G(v)]$, Algorithm 4 finds the maximum j such that $\beta_{\max,j}(v) \geq i$ according to Equation 7, and then adds the pair $(j-1, i)$ into final result set $\mathcal{A}(v)$ (lines 2-5).

We use the bipartite graph in Figure 1 to illustrate the optimized (α, β)-core decomposition algorithm, and the details are shown in Table II. We take vertex v_1 as an example. For v_1, $\mathsf{N}_G(v_1) = \{u_1, u_2\}$, $\mathsf{deg}_G(v_1) = 2$, $\mathsf{deg}_G(u_1) = 3$, and $\mathsf{deg}_G(u_1) = 3$. Thus, $\mathcal{B}^{(0)}(v_1) = \{(1,2), (2,2), (3,2)\}$. Then, the algorithm computes $\mathcal{B}^{(1)}(v_1)$. Since $\mathsf{B}_V^{(1)}(3, v_1) = \mathbf{H}(\{\mathsf{B}_U^{(0)}(3, u_1), \mathsf{B}_U^{(0)}(3, u_2)\}) = \mathbf{H}(\{4, 4\}) = 2$, $\mathsf{B}_V^{(1)}(2, v_1) = \mathbf{H}(\{\mathsf{B}_U^{(0)}(2, u_1), \mathsf{B}_U^{(0)}(2, u_2)\}) = \mathbf{H}(\{4, 4\}) = 2$, and $\mathsf{B}_V^{(1)}(1, v_1) = \mathbf{H}(\{\mathsf{B}_U^{(0)}(1, u_1), \mathsf{B}_U^{(0)}(1, u_2)\}) = \mathbf{H}(\{4, 4\}) = 2$, $\mathcal{B}^{(1)}(v_1) = \{(1,2), (2,2), (3,2)\}$. Similarly, $\mathcal{B}^{(2)}(v_1)$, $\mathcal{B}^{(3)}(v_1)$, and $\mathcal{B}^{(4)}(v_1)$ are computed. Since $\mathcal{B}^{(3)}(v) = \mathcal{B}^{(4)}(v)$ and $\mathcal{B}^{(3)}(u) = \mathcal{B}^{(4)}(u)$, the algorithm terminates. Finally, the algorithm converts $\mathcal{B}^{(4)}(v)$ to $\mathcal{A}^{(4)}(v)$ for all vertices in V_G. It is worth mentioning that the $\mathcal{B}^{(n)}(u)$ computed in the optimized algorithm is the same as that of non-optimized algorithm, i.e., Algorithm 1.

Next, we analyze the time complexity of optimized (α, β)-core decomposition algorithm.

Theorem 10. *The optimized (α, β)-core decomposition algorithm and Algorithm 1 have the same number of iterations.*

Theorem 11. *The time and space complexity of the optimized (α, β)-core decomposition algorithm is $O(r \cdot m \cdot n_V)$ and $O(m + n_V \cdot \max \mathsf{deg}_G(u))$, respectively.*

Proof. The optimized algorithm iteratively computes $\mathcal{B}^{(n)}(u)$ and $\mathcal{B}^{(n)}(v)$ until convergence. Thus, the time complexity of the optimized algorithm is also determined by the number of iterations and the time required for each iteration. In each iteration, (1) for a vertex u, Algorithm 3 computes $\mathsf{B}_U^{(n)}(\alpha, u)$ for all $\alpha \in [1, \mathsf{deg}_G(u)]$ using $\mathbf{TOP}(\alpha, I)$, whose time complexity is $O(\mathsf{deg}_G(u) \cdot \mathsf{deg}_G(u))$; (2) for a vertex v, Algorithm 3

computes $\mathsf{B}_V^{(n)}(\alpha, v)$ for all $\alpha \in [1, \max_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i)]$ using $\mathbf{H}(I)$, whose time complexity is $O(\max_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i) \cdot \mathsf{deg}_G(v))$. Totally, the time complexity of a single round is $O(\sum_{u \in U_G} \mathsf{deg}_G(u) \cdot \mathsf{deg}_G(u) + \sum_{v \in V_G} \max_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i) \cdot \mathsf{deg}_G(v)) = O(m \cdot n_V)$. As a result, the time complexity of optimized (α, β)-core decomposition algorithm is $O(r \cdot m \cdot n_V)$.

The optimized algorithm should compute $\mathcal{B}(u)$ and $\mathcal{B}(v)$ for every u and v, respectively. $|\mathcal{B}(u)| = \mathsf{deg}_G(u)$ and $|\mathcal{B}(v)| = \max_{u \in \mathsf{N}_G(v)} \mathsf{deg}_G(u)$. Hence, the space complexity of the optimized algorithm is $O(\sum_{u \in U_G} \mathsf{deg}_G(u) + \sum_{v \in V_G} \max_{u \in \mathsf{N}_G(v)} \mathsf{deg}_G(u)) = O(m + n_V \cdot \max \mathsf{deg}_G(u))$. □

D. Distributed (α, β)-core Decomposition Algorithm

In this subsection, we extend (α, β)-core decomposition algorithms proposed in Sections III-B and III-C to distributed graph processing frameworks. We consider two types of well known frameworks, including vertex-centric framework and block-centric framework. We take the Algorithm 1 as an example. The optimized algorithm also can be extended in the same way.

Vertex-centric Distributed (α, β)-core Decomposition Algorithm.

For the vertex-centric framework, each vertex corresponds to one computing node, which has two states, i.e., *active* and *inactive*. To be specific, the vertex-centric framework works as follows. First, the vertex-centric framework does some initialization for each vertex and marks it as inactive. Then, when a vertex receives messages from its neighbors, it becomes active, executes a user-defined function, sends messages to its neighbors if necessary, and resets to inactive. The framework stops once all vertices become inactive. Pregel [30], GPS [31], and GraphLab [32] are typical vertex-centric frameworks.

According to the workflow of vertex-centric framework, we should consider the details of (1) initialization and (2) operations after receiving messages for each vertex. Specifically, in initialization step, the distributed algorithm should set $\mathcal{B}^{(0)}(u)/\mathcal{A}^{(0)}(v)$ for vertices, which corresponds to lines 1-6 of Algorithm 1, and then the $\mathcal{B}^{(0)}(u)/\mathcal{A}^{(0)}(v)$ should be broadcast to all neighbors of the vertex via messages, which is to trigger 1-order Bi-indexes computation. Next, we discuss the operations after receiving messages. Note that, in our distributed algorithms, the messages just contain the latest $\mathcal{B}^{(n)}(u)/\mathcal{A}^{(n)}(v)$ of sender. So, after receiving messages, the vertex should update the latest $\mathcal{B}^{(n)}(u)/\mathcal{A}^{(n)}(v)$ of its neighbors. It is because for distributed algorithm, each vertex should store the latest $\mathcal{B}^{(n)}(u)/\mathcal{A}^{(n)}(v)$ of its neighbors to compute its own $\mathcal{B}^{(n+1)}(u)/\mathcal{A}^{(n+1)}(v)$. When all messages are processed, the vertex should compute its own $\mathcal{B}^{(n+1)}(u)/\mathcal{A}^{(n+1)}(v)$, which corresponds to Algorithm 2. When the $\mathcal{B}^{(n+1)}(u)/\mathcal{A}^{(n+1)}(v)$ computation is finished, if vertex's $\mathcal{B}^{(n+1)}(u) \neq \mathcal{B}^{(n+1)}(u)/\mathcal{A}^{(n+1)}(v) \neq \mathcal{A}^{(n+1)}(v)$, the vertex should send $\mathcal{B}^{(n+1)}(u)/\mathcal{A}^{(n+1)}(v)$ to all its neighbors. Otherwise, the vertex does not need to broadcast the messages. When no vertex broadcasts messages, the distributed algorithm finished.

Block-centric Distributed (α, β)-core Decomposition Algorithm.

Another popular distributed graph processing framework is block-centric framework, such as Giraph++ [33], Blogel [34], and GRAPE [35]. For the block-centric framework, the graph is partitioned into many blocks and a single computing node stores one block of vertices. As for the communication of block-centric framework, (1) the communication occurs after the computation within a block reaches convergence; (2) the communication occurs between blocks. The block-centric framework stops when there is no communication between any two blocks.

Next, we extend Algorithm 1 to the block-centric framework. At initialization step, each block also should set $\mathcal{B}^{(0)}(u)/\mathcal{A}^{(0)}(v)$ for all vertices within the block. Then, $\mathcal{B}^{(0)}(u)/\mathcal{A}^{(0)}(v)$ of a block are broadcast to other blocks. When the block receives messages (i.e., the $\mathcal{B}^{(n)}(u)/\mathcal{A}^{(n)}(v)$ of vertices in other blocks), it first updates $\mathcal{B}^{(n)}(u)/\mathcal{A}^{(n)}(v)$, and then the block computes the $\mathcal{B}^{(n+1)}(u)/\mathcal{A}^{(n+1)}(v)$ of vertices in its own block. When $\mathcal{B}^{(n+1)}(u)/\mathcal{A}^{(n+1)}(v)$ is converged within the block, the block broadcasts $\mathcal{B}^{(n+1)}(u)/\mathcal{A}^{(n+1)}(v)$ to other blocks. If there are no messages broadcast between any two blocks, the distributed algorithm terminates.

In the following, we analyze the communication complexity for distributed (α, β)-core decomposition.

Theorem 12. *The total communication complexity of the distributed (α, β)-core decomposition algorithm is $O(r \cdot m)$.*

Proof. The distributed (α, β)-core decomposition algorithm iteratively computes the n-order Bi-indexes until convergence. Thus, the total communication complexity is determined by the number of iterations and the communication complexity of each iteration. For each iteration, at worst, every vertex w should send one message, whose size is $O(\mathsf{deg}_G(w))$. The communication complexity for each iteration is thus $O(\sum_{w \in U_G \cup V_G} \mathsf{deg}_G(w)) = O(m)$. Therefore, the total communication complexity for distributed (α, β)-core decomposition is $O(r \cdot m)$. □

IV. Empirical Evaluation

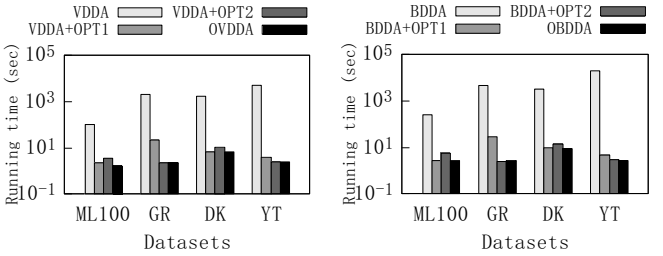
A. Experimental Settings

All experiments are conducted on a collection of Amazon EC2 r5.2x large instances, each powered by 8 vCPUs and 64GB memory. The network bandwidth is up to 10G Gb/s. All experiments are implemented in C++ on the Ubuntu 18.04 operating system.

Datasets. We use eleven datasets to evaluate the algorithms, including nine real bipartite graphs and two synthetic bipartite graphs. For the real bipartite graphs, MovieLens-100K and MovieLens-10M are the bipartite graphs of *user*'s rating of *movies*; Daily Kos, Enron Words, and PubMed are the bipartite graphs of *word*'s occurrence in different *documents*. YouTube

TABLE III: Dataset Statistics (**dmax** represents the maximum degree of a bipartite graph; $\mathbf{K} = 10^3$, $\mathbf{M} = 10^6$, and $\mathbf{B} = 10^9$)

Dataset	Abbr.	$\lvert U_V \rvert$	$\lvert V_G \rvert$	$\lvert E_G \rvert$	dmax
MovieLens-100K	ML100	943	1,682	100**K**	737
Daily Kos	DK	3,430	6,906	353**K**	2,123
YouTube	YT	94.2**K**	30.1**K**	293**K**	7,591
BookCrossing	BC	78**K**	186**K**	434**K**	8,524
Genres	GR	259**K**	7,783	463**K**	25**K**
Enron Words	EW	39.9**K**	28.1**K**	3.71**M**	7,190
MovieLens-10M	ML10	70**K**	11**K**	10**M**	35**K**
LiveJournal	LJ	3.20**M**	7.49**M**	11.3**M**	1.05**M**
PubMed	PM	8.2**M**	141**K**	483**M**	2.32**M**
Uniform	UD	5**M**	5**M**	1.06**B**	211
Power Law	PL	5**M**	5**M**	1.06**B**	38.1**K**



(a) Vertex-centric (b) Block-centric

Fig. 2: Efficiency of optimizations (Running time)

and LiveJournal are the bipartite graphs of *user*'s membership w.r.t., different *groups*. BookCrossing is a bipartite graph of *user*'s rating of *books*. Genres is a bipartite graph of relationship between *genres* and *works*. All the real datasets are downloaded from the website KONECT[1]. In addition, we also generate two synthetic bipartite graphs, i.e., Uniform and Power Law, whose degree distributions follow uniform distribution and power-law distribution, respectively. The dataset statistics are summarized in Table III.

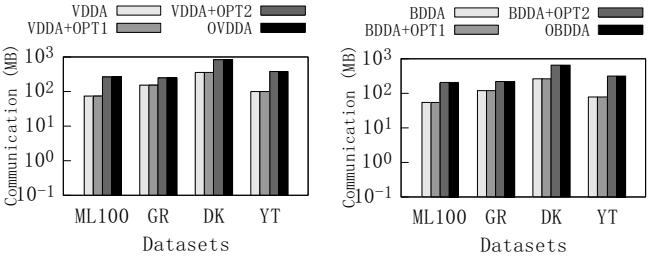
Algorithms. We compare a set of algorithms in our experiments. Specifically, VDDA and BDDA are the distributed (α, β)-core decomposition algorithms, which extend Algorithm 1 to vertex-centric and block-centric frameworks, respectively. VDDA+OPT1 and BDDA+OPT1 are VDDA and BDDA integrated with Optimization 1, respectively. VDDA+OPT2 and BDDA+OPT2 are VDDA and BDDA integrated with Optimization 2, respectively. OVDDA and OBDDA are VDDA and BDDA integrated with 2 optimizations, respectively.

In our experiments, we employ a popular block-centric framework GRAPE [35] to realize the block-centric distributed algorithms, and use the hash partitioner for graph partitioning by default. For the sake of fairness, we also employ GRAPE to simulate the vertex-centric framework. Specifically, at each round, all vertices execute computations only once and broadcast to their neighbors immediately.

B. Experimental Results

Exp-1: Evaluation of optimization efficiency. We start by evaluating the efficiency of two optimizations proposed

[1]http://konect.cc/networks/



(a) Vertex-centric (b) Block-centric

Fig. 3: Efficiency of optimizations (Communication)

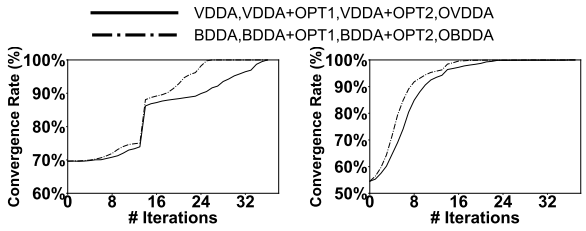
in Section III-C. Figures 2 and 3 report the running time and communication overhead of all algorithms, respectively. For the running time shown in Figure 2, within the same framework, the performance of algorithms with two optimizations are the best, following by the algorithms with only one optimization. The algorithms without any optimizations have the worst performance. This phenomenon demonstrates the effectiveness of Optimizations 1 and 2. Moreover, we can observe that the algorithms with optimizations consistently outperform the algorithms without optimizations by 1-3 orders of magnitude on all datasets. For example, OBDDA has three orders of magnitude improvement compared with BDDA. In addition, the two optimizations have different strengths on different datasets. For example, Optimization 1 outperforms Optimization 2 over datasets ML100 and DK while Optimization 2 works better over datasets GR and YT.

For the communication overhead reported in Figure 3, we have following observations: (1) VDDA and VDDA+OPT1 have the same communication overhead; (2) VDDA+2 and OVDDA have the same communication overhead; and (3) the communication overheads of VDDA and VDDA+OPT1 are smaller than that of VDDA+2 and OVDDA. The reasons behind are that (1) both VDDA and VDDA+OPT1 broadcast $\mathcal{B}^{(n)}(u)$ and $\mathcal{A}^{(n)}(v)$; (2) both VDDA+OPT2 and OVDDA broadcast $\mathcal{B}^{(n)}(u)$ and $\mathcal{B}^{(n)}(v)$; (3) according to Definitions 4 and 5, $|\mathcal{B}^{(n)}(u)| = \mathsf{deg}_G(u)$, $|\mathcal{A}^{(n)}(v)| = \mathsf{deg}_G(v)$, and $|\mathcal{B}^{(n)}(v)| = \max_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i)$. Obviously, $\sum_{\forall v \in V_G} \mathsf{deg}_G(v) < \sum_{\forall v \in V_G} \max_{\forall u_i \in \mathsf{N}_G(v)} \mathsf{deg}_G(u_i)$. Therefore, VDDA+2 and OVDDA have more communication overhead. However, the magnitude of the increase in communication is small. For example, in Figure3(a), OVDDA has 14% more communication overhead than that of VDDA on dataset GR. In addition, the increase of communication overhead is less than an order of magnitude for all datesets. Compared the significant improvement of running time, the increase of communication overhead is acceptable. Overall, our proposed optimizations are effective.

Exp-2: Evaluation on the number of iterations. In this experiments, we explore the number of iterations for all algorithms. Table IV reports the results on datasets ML100, GR, DK, and YT. We can observe that, first, the algorithms of the same framework have the same number of iterations. It is because Optimization 1 employs the lower and upper bounds to reduce the candidate pairs examination and Optimization 2

TABLE IV: # Iterations required for the algorithms

Algorithms		Dataset			
		ML100	GR	DK	YT
Vertex-centric	VDDA	38	39	76	39
	VDDA+OPT1	38	39	76	39
	VDDA+OPT2	38	39	76	39
	OVDDA	38	39	76	39
Block-centric	BDDA	27	29	54	26
	BDDA+OPT1	27	29	54	26
	BDDA+OPT2	27	29	54	26
	OBDDA	27	29	54	26



(a) ML100 (b) GR

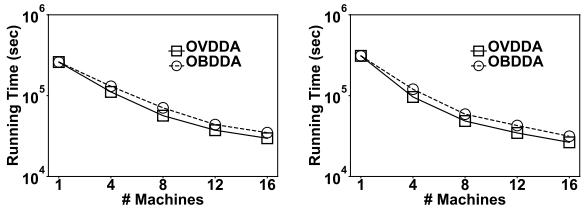
Fig. 4: Convergence Rate

speed up the $\mathcal{A}^{(n)}(v)$ computation via $\mathcal{B}^{(n)}(v)$. Both optimizations only accelerate the calculation of $\mathcal{A}^{(n)}(v)$ within a single iteration, and does not influence other iterations. Second, the number of iterations for block-centric algorithms are less than that of the vertex-centric algorithms. The reason behind is that in block-centric algorithms, the vertices have been locally converged before the blocks broadcast messages. Thus, block-centric algorithms converge faster globally. Third, for a given graph G, all of our proposed algorithms have few iterations. It also shows the efficiency of our proposed algorithms.

Exp-3: Evaluation of convergence rate. According to Table II, many vertices converge within few iterations. For example, vertex u_2 has converged in the first iteration. To this end, we evaluate the convergence rate for each algorithm, i.e., the percentage of vertices with $\mathcal{B}^{(n)}(u) = \mathcal{B}(u)$, or $\mathcal{B}^{(n)}(v) = \mathcal{B}(v)$, or $\mathcal{A}^{(n)}(v) = \mathcal{A}(v)$ after an iteration finishes. Figure 4 shows the results over four datasets. As expected, the algorithms of the same framework have the same convergence rate, and the block-centric algorithms converges faster, which are consistent with the results of **Exp-2**. Overall, all algorithms have fast convergence rate. For example, for the dataset GR in Figure 4(b), 99.7% and 96.8%vertices have been converged after 16 iterations for block-centric and vertex-centric algorithms, respectively.

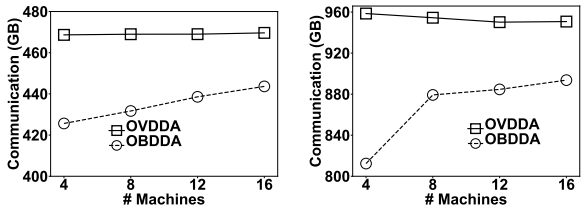
In the rest of experiments, we evaluate the efficiency of proposed algorithms. We only report the results of the OVDDA and OBDDA algorithms due to the inefficiency of the other algorithms over large datasets.

Exp-4: Effect of the number of machines. This set of experiments test the effect of the number of machines, whose range varies from 1 to 16, where 1 represents centralized setting and the others represent distributed settings. Figures 5 and 6 show the running time and communication overhead,



(a) PL (b) UD

Fig. 5: Effect of the number of machines (Running time)



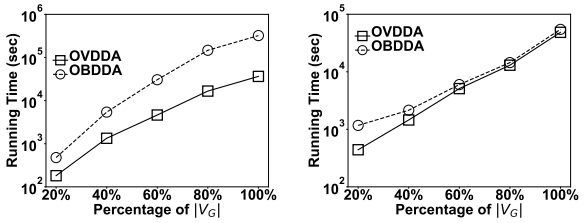
(a) PL (b) UD

Fig. 6: Effect of the number of machines (Communication)

respectively. In Figure 5, we can observe that (1) in distributed setting, our proposed algorithms have better performance, and (2) when the number of machines increases, OVDDA and OBDDA take less running time. This is because more machines render more computing power for distributed algorithms. Thus, both algorithms have better performance. In addition, OVDDA takes less time than OBDDA, which is caused by straggler [36]. Recall that in each iteration, OBDDA should iteratively compute $\mathcal{B}^{(n)}(u)$ and $\mathcal{B}^{(n)}(v)$ until convergence within the block. There may be some blocks that converge very slowly, and thus deteriorates the overall performance of the OBDDA. Moreover, the smaller the blocks, the weaker the effect of straggler. Hence, with the increase of the number of machines, the gap between the performance of OVDDA and OBDDA becomes smaller.

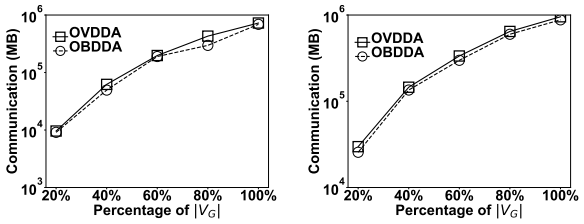
In Figure 6, we can observe that OBDDA has less communication overhead than OVDDA. Although the communication overhead of OBDDA is more than that of OVDDA in each iteration, OBDDA takes much less iterations than OVDDA. Totally, OBDDA has less communication overhead. With the increase of the number of machines, we have the following observations. (1) The communication overhead of OVDDA almost remain stable. This is because vertex-centric framework considers each vertex as a machine and the communication occurs between edges. Changes in the number of machines do not affect edges, and therefore the communication overhead. (2) The communication overhead of OBDDA increases. It is because, when the number of machines increases, each block becomes smaller and the block-centric framework is more similar to the vertex-centric framework, meaning that OBDDA takes more iterations. Thus, the communication overhead of OBDDA increases, and becomes closer to that of OVDDA.

Exp-5: Effect of dataset cardinality. We randomly select different fractions of vertices from the original graphs, which vary from 20% to 100%, and generate a set of induced subgraphs. We employ these subgraphs to test the effect of dataset cardinality over OVDDA and OBDDA. Figures 7 and



(a) PM (b) UD

Fig. 7: Effect of cardinality (Running time)



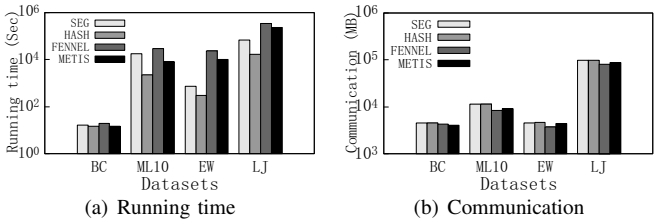
(a) PM (b) UD

Fig. 8: Effect of cardinality (Communication)

8 show the running time and communication overhead, respectively. It is within expected that both the running time and communication overhead increase with the dataset cardinality.

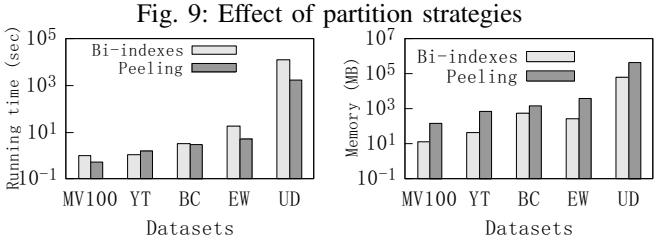
Exp-6: Effect of partition strategies. As block-centric framework partitions graph into blocks, finally, we test the effect of partition strategies for OBDDA. Four partition strategies are compared, including (1) SEG [35], which allocates vertex to the v_{id}/C-th subgraph (v_{id} and C denote vertex ID and the maximum cardinality of partitioned subgraphs, respectively); (2) HASH [35], which allocates vertex to the $v_{id}\%N$-th subgraph (N denotes the number of partitioned subgraphs); (3) FENNEL [37], which subsumes the folklore heuristic that places a vertex to the subgraph with the fewest non-neighbors and the degree-based heuristic that uses different heuristics to place a vertex based on its degree, to partition the graph; and (4) METIS [38], which partitions the graph into subgraphs with minimum crossing edges. The corresponding results are shown in Figure 9. We can observe that (1) HASH is the best in terms of running time , which is due to its balanced partitions, i.e., each partition has almost an equal number of vertices. (2) METIS and FENNEL are worse than HASH in terms of running time, sometimes by an order of magnitude. But, their communications are relatively less than HASH. This is because METIS and FENNEL have higher locality, leading to more prominent effect of straggler. Considering the running time advantage of HASH, and not much worse in communication overhead, we employ HASH as default partition strategy in experiments.

Exp-7: Bi-indexes-based algorithm vs. Peeling-based algorithm. Our proposed Bi-indexes-based algorithms can be implemented in both centralized setting and existing distributed graph frameworks. In this experiment, we compare our proposed algorithm and peeling-based algorithm in centralized settings. The result is shown in Figure 10. We can observe that the peeling-based algorithm is better in terms of running time while it takes more memory. Therefore, our proposed algorithms are more preferable for distributed settings.



(a) Running time (b) Communication

Fig. 9: Effect of partition strategies



(a) Running time (b) Memory

Fig. 10: Bi-indexes-based vs. Peeling-based algorithms

V. Related Work

Distributed Core Decomposition. The k-core is an important model for cohesive subgraph analysis [39]. Given an undirected graph, the problem of k-core decomposition aims at finding the k-cores for all possible values of k. By now, many efficient algorithms have been proposed to handle k-core decomposition [40]–[43]. Moreover, several parallel [44]–[46] and distributed [24]–[28] algorithms for k-core decomposition also have been proposed, which cannot be applied to handle our problem. It is because different core models require different core decomposition techniques, and these work do not involve (α, β)-core.

Cohesive Subgraphs Discovery over Bipartite Graphs. Apart from (α, β)-core, other cohesive subgraph models also have been proposed for bipartite graphs, such as biclique [6]–[10], bitruss [11], [12], [47], biplex [48], [49], and (α, β, η)-core [50]. However, these work also cannot be applied to our work. The reason is two folded. First, these cohesive subgraph models are different from (α, β)-core. Second, these work focus on centralized environments.

VI. Conclusion

In this paper, we explore the problem of distributed (α, β)-core decomposition. To solve this problem, we first analyze the relationship between vertex and its neighbors and propose n-order Bi-indexes for vertices. Then, we propose an algorithm of (α, β)-core decomposition through iteratively computing n-order Bi-indexes. We also present two optimizations to accelerate the algorithm and discuss how to implement the algorithm into different distributed graph processing frameworks. Finally, we conduct extensive experiments using real and synthetic graphs to evaluate the proposed algorithms. The results demonstrate the efficiency, scalability and effectiveness of proposed algorithms and optimizations.

Acknowledgments. This work is supported by the NSFC under Grants No. 62025206 and 61972338, Hong Kong RGC grants C2004-21GF, 12202221, 12201520, and 12200021. Jianliang Xu is the corresponding author.

REFERENCES

[1] S. P. Borgatti and M. G. Everett, "Network analysis of 2-mode data," *Social networks*, vol. 19, no. 3, pp. 243–269, 1997.

[2] A. S. Asratian, T. M. Denley, and R. Häggkvist, *Bipartite graphs and their applications*. Cambridge university press, 1998, vol. 131.

[3] A. Ahmed, V. Batagelj, X. Fu, S. Hong, D. Merrick, and A. Mrvar, "Visualisation and analysis of the internet movie database," in *APVIS*, 2007, pp. 17–24.

[4] M. Cerinsek and V. Batagelj, "Generalized two-mode cores," *Soc. Networks*, vol. 42, pp. 80–87, 2015.

[5] Y. Hao, M. Zhang, X. Wang, and C. Chen, "Cohesive subgraph detection in large bipartite networks," in *SSDBM*, 2020, pp. 22:1–22:4.

[6] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum biclique search at billion scale," *Proc. VLDB Endow.*, vol. 13, no. 9, pp. 1359–1372, 2020.

[7] A. Abidi, R. Zhou, L. Chen, and C. Liu, "Pivot-based maximal biclique enumeration," in *IJCAI*, 2020, pp. 3558–3564.

[8] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient exact algorithms for maximum balanced biclique search in bipartite graphs," in *SIGMOD*, 2021, pp. 248–260.

[9] S. Hirahara and N. Shimizu, "Nearly optimal average-case complexity of counting bicliques under SETH," in *SODA*, 2021, pp. 2346–2365.

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

[11] Z. Zou, "Bitruss decomposition of bipartite graphs," in *DASFAA*, 2016, pp. 218–233.

[12] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Efficient bitruss decomposition for large-scale bipartite graphs," in *ICDE*, 2020, pp. 661–672.

[13] C. Chen, Q. Zhu, Y. Wu, R. Sun, X. Wang, and X. Liu, "Efficient critical relationships identification in bipartite networks," *World Wide Web*, To appear.

[14] D. Ding, H. Li, Z. Huang, and N. Mamoulis, "Efficient fault-tolerant group recommendation using alpha-beta-core," in *CIKM*, 2017, pp. 2047–2050.

[15] K. Wang, W. Zhang, X. Lin, Y. Zhang, L. Qin, and Y. Zhang, "Efficient and effective community search on large-scale bipartite graphs," in *ICDE*, 2021, pp. 85–96.

[16] Y. He, K. Wang, W. Zhang, X. Lin, and Y. Zhang, "Exploring cohesive subgraphs with vertex engagement and tie strength in bipartite graphs," *Inf. Sci.*, vol. 572, pp. 277–296, 2021.

[17] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation in bipartite graphs," *VLDB J.*, vol. 29, no. 5, pp. 1075–1099, 2020.

[18] T. Zhang, Y. Gao, L. Chen, W. Guo, S. Pu, B. Zheng, and C. S. Jensen, "Efficient distributed reachability querying of massive temporal graphs," *VLDB J.*, vol. 28, no. 6, pp. 871–896, 2019.

[19] X. Ren, J. Wang, W. Han, and J. X. Yu, "Fast and robust distributed subgraph enumeration," *Proc. VLDB Endow.*, vol. 12, no. 11, pp. 1344–1356, 2019.

[20] Y. Zhuo, J. Chen, Q. Luo, Y. Wang, H. Yang, D. Qian, and X. Qian, "Symplegraph: distributed graph processing with precise loop-carried dependency guarantee," in *PLDI*, A. F. Donaldson and E. Torlak, Eds., 2020, pp. 592–607.

[21] K. Hao, L. Yuan, and W. Zhang, "Distributed hop-constrained s-t simple path enumeration at billion scale," *Proc. VLDB Endow.*, vol. 15, no. 2, pp. 169–182, 2021.

[22] S. Shahrivari and S. Jalili, "Efficient distributed k-clique mining for large networks using mapreduce," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 3, pp. 964–974, 2021.

[23] C. Rost, K. Gómez, M. Täschner, P. Fritzsche, L. Schons, L. Christ, T. Adameit, M. Junghanns, and E. Rahm, "Distributed temporal graph analytics with GRADOOP," *VLDB J.*, vol. 31, no. 2, pp. 375–401, 2022.

[24] A. Montresor, F. D. Pellegrini, and D. Miorandi, "Distributed k-core decomposition," *IEEE Trans. Parallel Distributed Syst.*, vol. 24, no. 2, pp. 288–300, 2013.

[25] N. S. Dasari, D. Ranjan, and M. Zubair, "Park: An efficient algorithm for k-core decomposition on multicore processors," in *Big Data*, 2014, pp. 9–16.

[26] A. Mandal and M. A. Hasan, "A distributed k-core decomposition algorithm on spark," in *Big Data*, 2017, pp. 976–981.

[27] T. H. Chan, M. Sozio, and B. Sun, "Distributed approximate k-core decomposition and min-max edge orientation: Breaking the diameter barrier," in *IPDPS*, 2019, pp. 345–354.

[28] X. Liao, Q. Liu, J. Jiang, X. Huang, J. Xu, and B. Choi, "Distributed d-core decomposition over large directed graphs," *Proc. VLDB Endow.*, vol. 15, no. 8, pp. 1546–1558, 2022.

[29] J. E. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National academy of Sciences*, vol. 102, no. 46, pp. 16 569–16 572, 2005.

[30] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *SIGMOD*, 2010, pp. 135–146.

[31] S. Salihoglu and J. Widom, "GPS: a graph processing system," in *SSDBM*, 2013, pp. 22:1–22:12.

[32] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning in the cloud," *PVLDB*, vol. 5, no. 8, pp. 716–727, 2012.

[33] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From "think like a vertex" to "think like a graph"," *PVLDB*, vol. 7, no. 3, pp. 193–204, 2013.

[34] D. Yan, J. Cheng, Y. Lu, and W. Ng, "Blogel: A block-centric framework for distributed computation on real-world graphs," *PVLDB*, vol. 7, no. 14, pp. 1981–1992, 2014.

[35] W. Fan, W. Yu, J. Xu, J. Zhou, X. Luo, Q. Yin, P. Lu, Y. Cao, and R. Xu, "Parallelizing sequential graph computations," *ACM Trans. Database Syst.*, vol. 43, no. 4, pp. 18:1–18:39, 2018.

[36] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *OSDI*, 2010, pp. 265–278.

[37] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "FENNEL: streaming graph partitioning for massive scale graphs," in *WSDM*, 2014, pp. 333–342.

[38] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs," *Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version*, vol. 4, no. 0, 1998.

[39] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.

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

[41] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *ICDE*, 2011, pp. 51–62.

[42] W. Khaouid, M. Barsky, S. Venkatesh, and A. Thomo, "K-core decomposition of large networks on a single PC," *PVLDB*, vol. 9, no. 1, pp. 13–23, 2015.

[43] F. Bonchi, A. Khan, and L. Severini, "Distance-generalized core decomposition," in *SIGMOD*, 2019, pp. 1006–1023.

[44] L. Lü, T. Zhou, Q.-M. Zhang, and H. E. Stanley, "The h-index of a network node and its relation to degree and coreness," *Nature communications*, vol. 7, no. 1, pp. 1–7, 2016.

[45] A. E. Sariyüce, C. Seshadhri, and A. Pinar, "Local algorithms for hierarchical dense subgraph discovery," *PVLDB*, vol. 12, no. 1, pp. 43–56, 2018.

[46] Q. Liu, X. Zhu, X. Huang, and J. Xu, "Local algorithms for distance-generalized core decomposition over large dynamic graphs," *PVLDB*, vol. 14, no. 9, pp. 1531–1543, 2021.

[47] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs," *VLDB J.*, vol. 31, no. 2, pp. 203–226, 2022.

[48] K. Yu, C. Long, S. Liu, and D. Yan, "Efficient algorithms for maximal k-biplex enumeration," in *SIGMOD*, 2022, pp. 860–873.

[49] W. Luo, K. Li, X. Zhou, Y. Gao, and K. Li, "Maximum biplex search over bipartite graphs," in *ICDE*, 2022, pp. 898–910.

[50] G. Zhao, K. Wang, W. Zhang, X. Lin, Y. Zhang, and Y. He, "Efficient computation of cohesive subgraphs in uncertain bipartite graphs," in *ICDE*, 2022, pp. 2334–2346.