Density Decomposition of Bipartite Graphs

YALONG ZHANG, Beijing Institute of Technology, China
RONG-HUA LI, Beijing Institute of Technology, China
QI ZHANG, Beijing Institute of Technology, China
HONGCHAO QIN, Beijing Institute of Technology, China
LU QIN, University of Technology Sydney, Australia
GUOREN WANG, Beijing Institute of Technology, China

Mining dense subgraphs in a bipartite graph is a fundamental task in bipartite graph analysis, with numerous applications in community detection, fraud detection, and e-commerce recommendation. Existing dense subgraph models, such as biclique, k-biplex, k-bitruss, and (α, β)-core, often face challenges due to their high computational complexity or limitations in effectively capturing the density of the graph. To overcome these issues, in this paper, we propose a new dense subgraph model for bipartite graphs, namely (α, β)-dense subgraph, designed to capture the density structure inherent in bipartite graphs. We show that all (α, β)-dense subgraphs are nested within each other, forming a hierarchical density decomposition of the bipartite graph. To efficiently compute the (α, β)-dense subgraph, we develop a novel network flow algorithm with a carefully-designed core pruning technique. The time complexity of our algorithm is $O(|E| + |E(R)|^{1.5})$, where $|E|$ denotes the number of edges and $|E(R)|$ is the number of edges of the pruned graph, often significantly smaller than $|E|$. Armed with this algorithm, we also propose a novel and efficient divide-and-conquer algorithm to compute the entire density decomposition of the bipartite graph within $O(p \cdot \log d_{\max} \cdot |E|^{1.5})$ time, where p is typically a small constant in real-world bipartite graphs and $d_{\max}$ is the maximum degree. Extensive experiments and case studies on 11 real-world datasets demonstrate the effectiveness of our (α, β)-dense subgraph model and the high efficiency and scalability of our proposed algorithms.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**.

Additional Key Words and Phrases: bipartite graph; dense subgraph; network flow

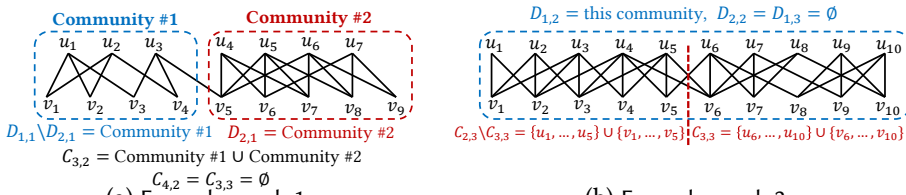
ACM Reference Format:
Yalong Zhang, Rong-Hua Li, Qi Zhang, Hongchao Qin, Lu Qin, and Guoren Wang. 2025. Density Decomposition of Bipartite Graphs. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 30 (February 2025), 25 pages. https://doi.org/10.1145/3709680

1 Introduction

A bipartite graph is a basic structure in which vertices are divided into two disjoint sets representing different types of entities, with edges connecting vertices in these sets to represent connections between entities. In the real world, bipartite graphs are ubiquitous, appearing in contexts such as

Authors' Contact Information: Yalong Zhang, Beijing Institute of Technology, Beijing, China, yalong-zhang@qq.com; Rong-Hua Li, Beijing Institute of Technology, Beijing, China, lironghuabit@126.com; Qi Zhang, Beijing Institute of Technology, Beijing, China, qizhangcs@bit.edu.cn; Hongchao Qin, Beijing Institute of Technology, Beijing, China, qhc.neu@gmail.com; Lu Qin, University of Technology Sydney, Sydney, Australia, lu.qin@uts.edu.au; Guoren Wang, Beijing Institute of Technology, Beijing, China, wanggrbit@126.com.

ACM 2836-6573/2025/2-ART30
https://doi.org/10.1145/3709680



(a) Example graph 1 (b) Example graph 2

Fig. 1. Example graphs with different models.

online user-item networks [14, 34], gene co-expression networks [40], social networks [6], and so on. Mining dense subgraphs in bipartite graphs is a fundamental technique in graph analysis with numerous applications across various fields. For instance, in user-item networks, dense subgraphs are crucial for e-commerce recommendations [3, 11, 18, 39], as customers with similar purchasing habits often belong to the same community and are grouped within dense subgraphs. In gene co-expression networks, dense subgraphs typically contain co-expressed gene modules, revealing regulatory networks and aiding in identifying functionally related gene groups [40]. In social networks, dense subgraphs play a key role in fraud detection [2, 7], as fraudulent activities often tend to form unusually dense subgraphs.

Various models have been proposed to characterize dense subgraphs in bipartite graphs. Notable examples include maximal biclique [1, 20, 23, 26, 40], k-biplex [15, 37, 38], k-bitruss [29, 30, 35, 42], and (α, β)-core [12, 17, 19, 22, 25]. However, all these existing models often face challenges due to their high computational complexity or limitations in effectively capturing the density of the graph. Specifically, the maximal biclique and k-biplex models do not have polynomial-time algorithms, making them intractable for large graphs. For k-bitruss, its computation requires identifying all butterflies (i.e., $(2, 2)$-cliques) in the graph. The number of butterflies can be extremely large (e.g., the Tracker graph in [35] has only 140 million edges but contains more than 2×10^{13} butterflies). Therefore, computing k-bitruss is also very costly for large real-world graphs.

Although (α, β)-core can be computed more efficiently, we observe that (α, β)-core, $C_{\alpha,\beta}$, often fails to accurately model the density structure of the graph, resulting in unsatisfactory outcomes, such as (1) failing to separate two different communities or (2) forcibly separating a densely-connected community. For example, as shown in Figure 1a, the $(3, 2)$-core contains two communities, i.e., $C_{3,2}$ = Community #1 ∪ Community #2, and $C_{4,2} = C_{3,3} = \emptyset$, indicating that (α, β)-core cannot separate the two communities even by adjusting the values of α and β. On the other hand, the graph in Figure 1b contains only one densely-connected community, while $C_{2,3}$ and $C_{3,3}$ forcibly split it. Furthermore, $C_{3,3}$ has a higher α value compared to $C_{2,3}$ and thus is supposed to be denser, but in fact, $C_{3,3}$ contains only 15 edges, whereas $C_{2,3} \setminus C_{3,3}$ contains 16 edges. This indicates that $C_{3,3}$ unreasonably splits the community and incorrectly identifies a sparser part as denser.

To address the shortcomings of existing models, we propose a novel dense subgraph model for bipartite graphs, called the (α, β)-dense subgraph. Compared to existing models, the advantages of our (α, β)-dense subgraph model are twofold. (1) **Density-based definition.** Unlike (α, β)-core, our (α, β)-dense subgraph can accurately reflect the density structure of bipartite graphs. For instance, in the example graphs shown in Figure 1, the (α, β)-dense subgraph, denoted as $D_{\alpha,\beta}$, correctly partitions the first graph into two distinct communities and does not split the community in the second graph, providing a more accurate and meaningful decomposition than the (α, β)-core. (2) **Efficient computation and high scalability.** Unlike maximal biclique, k-biplex, and k-bitruss, which are computationally costly, the proposed (α, β)-dense subgraph model can be computed in near-linear time and is scalable to large graphs with billions of edges.

These advantages make the (α, β)-dense subgraph a powerful tool for analyzing bipartite graphs. It combines computational efficiency with accurate density modeling, enabling its application to a wide range of real-world problems. Building on the advantages of the (α, β)-dense subgraph, this paper makes the following key contributions:

Novel model. We present a novel (α, β)-dense subgraph model for bipartite graphs, based on the indegree and reachability of nodes within the orientation of the graph. Specifically, we partition nodes into two sets $S = \{u \in U | \vec{d}_u < \alpha\} \cup \{v \in V | \vec{d}_v < \beta\}$ and $T = \{u \in U | \vec{d}_u > \alpha\} \cup \{v \in V | \vec{d}_v > \beta\}$. When there is no path from a node in S to a node in T, the (α, β)-dense subgraph contains all the nodes within T or those can reach a node in T. We show that our model is *density-based* (see Theorem 2), and it is unique for a given pair of (α, β). Moreover, we demonstrate that all the (α, β)-dense subgraphs are nested within each other, forming a hierarchical density decomposition of the bipartite graph. Additionally, we prove a Sandwich Theorem stating that the (α, β)-dense subgraphs and (α, β)-cores are interlaced with each other, based on which powerful core-based pruning techniques are carefully designed for computing the (α, β)-dense subgraphs.

Efficient algorithms. Equipped with our model, we study the dense subgraph search (DSS) problem which aims to find the (α, β)-dense subgraph for a given pair of (α, β); and the density decomposition (DD) problem which computes all non-empty (α, β)-dense subgraphs. For the DSS problem, we first devise a basic algorithm DSS based on the definition of (α, β)-dense subgraph with a complexity of $O(|E|^2)$. Then, we develop a novel and more efficient network flow algorithm with a time complexity of $O(|E|^{1.5})$. To further improve the efficiency, we propose DSS++ which employs a carefully-designed core-based pruning technique to narrow the graph size to a much smaller subgraph R before the network flow computation, reducing the complexity to $O(|E| + |E(R)|^{1.5})$.

For the DD problem, we first present a basic algorithm DD that utilizes the DSS++ algorithm to compute the dense subgraphs layer by layer, with a complexity of $O(layer \cdot |E|^{1.5})$, where $layer$ is the number of layers of the density decomposition. Then, we propose a novel and more efficient network-flow algorithm DD+ along with a newly-developed divide-and-conquer technique, reducing the time complexity to $O(p \cdot \log d_{\max} \cdot |E|^{1.5})$, where p is typically a small constant in real-world graphs, and $d_{\max}$ is the maximum degree.

Extensive experiments. We use 9 large real-world bipartite graphs to evaluate the efficiency of the proposed algorithms. The results show that: (1) The DSS++ algorithm for (α, β)-dense subgraph search is significantly faster than DSS and DSS+, with up to 3674x and 14.8x speedup, respectively; (2) The core-based pruning technique used in DSS++ can considerably narrow the scale of the graph and thus minimize the network flow computation time, making the time cost of the (α, β)-dense subgraph search comparable to that of (α, β)-core subgraph; (3) For density decomposition computation, the improved DD+ algorithm is up to 244.5x faster than DD, while consuming almost the same amount of memory as DD. We also conduct case studies on 3 additional real-world bipartite graphs to demonstrate the effectiveness of our (α, β)-dense subgraph model. The results confirm that (1) compared to the other models, the (α, β)-dense subgraph is denser and more reasonable in modeling dense subgraphs in bipartite graphs; and (2) the density decomposition is superior to core decomposition for characterizing hierarchical dense subgraphs in bipartite graphs.

Reproducibility. The source code of this paper can be found at https://github.com/Flydragonet/ab-dense-subgraph.

2 Preliminaries

Consider an undirected and unweighted bipartite graph $G = (U, V, E)$, where U and V are two disjoint node sets representing the nodes on the upper and lower sides respectively, and $E \subseteq U \times V$

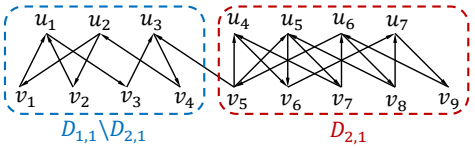


Fig. 2. Illustration of the orientation of a bipartite graph.

denotes the set of edges. For a node $x \in U \cup V$, its degree in G, denoted by $d_x(G)$, is the number of nodes linked to x, i.e., $d_x(G) = |\{y|(x,y) \in E\}|$. Alternatively, we denote the degree by d_x for brevity. Given a node subset $X \subseteq U \cup V$, let X^U and X^V be the upper and lower nodes of X, defined as $X^U \triangleq X \cap U$ and $X^V \triangleq X \cap V$, respectively. The subgraph induced by X is $G(X) = (X^U, X^V, E(X))$, where $E(X)$ includes all edges whose both endpoints are contained within X. For two node sets X and Y satisfying $X \cap Y = \emptyset$, we define the *cross edge* between X and Y as $E_\times(X,Y) = \{(x,y) \in E | x \in X, y \in Y\}$.

Assigning a direction to each edge of a graph $G = (U,V,E)$ transforms it into a directed graph termed as an *orientation* of G, denoted by $\vec{G} = (U,V,\vec{E})$, where $\vec{E}$ comprises the directed edges. For example, the directed graph depicted in Figure 2 is an orientation of the undirected graph shown in Figure 1a. We use angle brackets $\langle x,y \rangle$ to denote a directed edge, and round brackets (x,y) to denote an undirected edge. In the orientation $\vec{G}$, the indegree of a node $x \in U \cup V$ is denoted by $\vec{d}_x(\vec{G}) = |\{y|\langle y,x \rangle \in \vec{E}\}|$ or, more succinctly, $\vec{d}_x$. A *path* $s \rightsquigarrow t$ is a sequence of nodes $s = x_0, x_1, \cdots, x_{l-1}, t = x_l$, where $\langle x_{i-1}, x_i \rangle \in \vec{E}$ for $i = 1, \cdots, l$, and the length of the path is l. If a path $s \rightsquigarrow t$ exists, then we say that s can *reach* t.

Before introducing the problem of density decomposition of bipartite graphs, we first propose a novel dense subgraph model on bipartite graphs, called (α,β)-dense subgraph.

Definition 1. **((α,β)-dense subgraph)** *Given an undirected and unweighted bipartite graph $G = (U,V,E)$, two non-negative integers α and β, let $\vec{G}$ be an orientation of G, and let $S = \{u \in U | \vec{d}_u < \alpha\} \cup \{v \in V | \vec{d}_v < \beta\}$ and $T = \{u \in U | \vec{d}_u > \alpha\} \cup \{v \in V | \vec{d}_v > \beta\}$. If there is no path $s \rightsquigarrow t$ in the orientation $\vec{G}$ such that $s \in S$ and $t \in T$, then the (α,β)-dense subgraph $D_{\alpha,\beta}$ is the subgraph induced by the node set $T \cup \{x | x$ can reach a node in T in $\vec{G}\}$.*

In Definition 1, the orientation is used only to define the concept of $D_{\alpha,\beta}$. Therefore, regardless of which orientation $\vec{G}$ is used, as long as it satisfies the condition that there is no path from s to t, the resulting $D_{\alpha,\beta}$ is the same. That is, $D_{\alpha,\beta}$ is independent of the orientation, as shown by the following theorem demonstrating the uniqueness of $D_{\alpha,\beta}$.

Theorem 1. *Given a graph G and two non-negative integers α and β, the $D_{\alpha,\beta}$ is unique.*

Proof. Assume for contradiction that there are two different $D_{\alpha,\beta}$, denoted as D_1 and D_2, and let $D = D_1 \setminus D_2 \neq \emptyset$. Since $D \subseteq D_1$ and $D \subseteq (U \cup V) \setminus D_2$, we have $|E(D)| + |E_\times(D, D_1 \setminus D)| > \alpha \cdot |D^U| + \beta \cdot |D^V|$ and $|E(D)| + |E_\times(D, D_2)| \leq \alpha \cdot |D^U| + \beta \cdot |D^V|$ according to Theorem 2, which we will prove later. Given that $|E_\times(D, D_2)| \geq |E_\times(D, D_1 \setminus D)|$, we derive a contradiction: $\alpha \cdot |D^U| + \beta \cdot |D^V| < |E(D)| + |E_\times(D, D_1 \setminus D)| \leq |E(D)| + |E_\times(D, D_2)| \leq \alpha \cdot |D^U| + \beta \cdot |D^V|$. Thus, $D_{\alpha,\beta}$ is unique. □

Note that in Definition 1, the parameters α and β are introduced to constrain the indegrees of vertices on the two sides of bipartite graphs respectively, as these sides are typically asymmetric. For convenience, in the following, we use $D_{\alpha,\beta}$ to denote the (α,β)-dense subgraph or the set of nodes corresponding to that subgraph. By Definition 1, we can easily obtain the following results.

Lemma 1. *Given an orientation $\vec{G}$ that does not contain a path $s \rightsquigarrow t$ as defined in Definition 1. Then, we have: (1) the edges in $E_\times(D_{\alpha,\beta}, (U \cup V) \setminus D_{\alpha,\beta})$ are oriented toward $(U \cup V) \setminus D_{\alpha,\beta}$; (2) for any $u \in D^U_{\alpha,\beta}$ and $v \in D^V_{\alpha,\beta}$, we have $\vec{d}_u(\vec{G}) = \vec{d}_u(D_{\alpha,\beta}) \geq \alpha$ and $\vec{d}_v(\vec{G}) = \vec{d}_v(D_{\alpha,\beta}) \geq \beta$; (3) for any $u \in U \setminus D^U_{\alpha,\beta}$ and $v \in V \setminus D^V_{\alpha,\beta}$, we have $\vec{d}_u(\vec{G}) \leq \alpha$ and $\vec{d}_v(\vec{G}) \leq \beta$.*

Example 1. *Consider the orientation $\vec{G}$ shown in Figure 2. Let $\alpha = 2$ and $\beta = 1$, and then, according to Definition 1, we have $S = \{u_2\}$ and $T = \{v_6\}$. It can be observed that no node in S can reach any node in T. Consequently, the $(2,1)$-dense subgraph $D_{2,1}$ is the subgraph induced by the node set $T \cup \{x | x$ can reach a node in $T \in \vec{G}\} = \{u_4, \cdots, u_7\} \cup \{v_5, \cdots, v_9\}$ (i.e., the red cycle area in Figure 2).*

The intuition behind (α, β)-dense subgraph model. The following theorem demonstrates that our (α, β)-dense subgraph is density-based and can serve as an effective dense subgraph model for bipartite subgraphs.

Theorem 2. *The (α, β)-dense subgraph $D_{\alpha,\beta}$ satisfies: (1) **(inside dense)** for any non-empty $X \subseteq D_{\alpha,\beta}$, the removal of X from $D_{\alpha,\beta}$ results in a deletion of more than $\alpha \cdot |X^U| + \beta \cdot |X^V|$ edges, i.e., $|E(X)| + |E_\times(X, D_{\alpha,\beta} \setminus X)| > \alpha \cdot |X^U| + \beta \cdot |X^V|$; (2) **(outside sparse)** for any subset $Y \subseteq (U \cup V) \setminus D_{\alpha,\beta}$, the inclusion of Y into $D_{\alpha,\beta}$ leads to an increase of at most $\alpha \cdot |Y^U| + \beta \cdot |Y^V|$ edges, i.e., $|E(Y)| + |E_\times(Y, D_{\alpha,\beta})| \leq \alpha \cdot |Y^U| + \beta \cdot |Y^V|$.*

Proof. Given an orientation $\vec{G}$ that does not contain a path $s \rightsquigarrow t$ as defined in Definition 1, it is clear that $\vec{G}$ satisfies the three conditions given in Lemma 1.

For case (1), since the edges in $E_\times(D_{\alpha,\beta}, (U \cup V) \setminus D_{\alpha,\beta})$ are oriented toward $(U \cup V) \setminus D_{\alpha,\beta}$ (condition (1) in Lemma 1), the indegree of the nodes in X must come from the edges within $E(D_{\alpha,\beta})$. Thus, we have $\sum_{x \in X} \vec{d}_x(\vec{G}) = \sum_{x \in X} \vec{d}_x(D_{\alpha,\beta}) \leq |E(X)| + |E_\times(X, D_{\alpha,\beta} \setminus X)|$. Combining this with condition (2) in Lemma 1, we further get $|E(X)| + |E_\times(X, D_{\alpha,\beta} \setminus X)| \geq \sum_{x \in X} \vec{d}_x(\vec{G}) \geq \alpha \cdot |X^U| + \beta \cdot |X^V|$. These two inequalities cannot both hold as equalities; otherwise, X would not satisfy the definition of $D_{\alpha,\beta}$ and should not be included in $D_{\alpha,\beta}$. Thus, we have $|E(X)| + |E_\times(X, D_{\alpha,\beta} \setminus X)| > \alpha \cdot |X^U| + \beta \cdot |X^V|$.

Regarding case (2), we have $\alpha \cdot |Y^U| + \beta \cdot |Y^V| \geq \sum_{y \in Y} \vec{d}_y(\vec{G}) \geq \sum_{y \in Y} \vec{d}_y(D_{\alpha,\beta} \cup Y) = |E(Y)| + |E_\times(Y, D_{\alpha,\beta})|$, where the first inequality comes from condition (3) in Lemma 1 and the last equality holds because the edges in $E_\times(D_{\alpha,\beta}, Y)$ are oriented toward Y. □

With Theorem 2, the removal of any subset $X \subseteq D_{\alpha,\beta}$ results in the loss of more than $\alpha \cdot |X^U| + \beta \cdot |X^V|$ edges, while merging a node set Y located outside $D_{\alpha,\beta}$ into $D_{\alpha,\beta}$ only increases the number of edges by at most $\alpha \cdot |Y^U| + \beta \cdot |Y^V|$. These two aspects show that every subgraph in $D_{\alpha,\beta}$ is *locally* dense and well-connected, and the subgraph outside $D_{\alpha,\beta}$ is sparse and not closely connected to $D_{\alpha,\beta}$. Thus, $D_{\alpha,\beta}$ can be used to efficiently extract dense subgraphs, demonstrating why $D_{\alpha,\beta}$ is considered as *density-based* dense subgraph model. In contrast, the degree-based (α, β)-core does not exhibit this density property.

Example 2. *For $D_{2,1}$ in Figure 1a, according to Theorem 2, $D_{2,1}$ is an inside dense and outside sparse subgraph. For example, if we remove the four nodes $\{u_4, u_5, v_5, v_6\}$ **inside** of $D_{2,1}$ and their adjacent edges, $D_{2,1}$ will lose up to 9 edges. In contrast, if we add the four nodes $\{u_2, u_3, v_3, v_4\}$ **outside** of $D_{2,1}$ into $D_{2,1}$, $D_{2,1}$ will only gain 4 edges. For $D_{1,2}$ in Figure 1b, all parts of $D_{1,2}$ are relatively dense, so it cannot extract a subgraph like $C_{3,3}$.*

Furthermore, the definition of the (α, β)-dense subgraph is essentially analogous to a layer of the *density decomposition* defined on general graphs [10]. Since each layer of the density decomposition

is inherently a density-based dense subgraph [10], the (α, β)-dense subgraph should also be considered as a density-based dense subgraph. This conceptual alignment highlights the density-based nature of the (α, β)-dense subgraph.

The hierarchical property of (α, β)-dense subgraph. Below, we prove that the (α, β)-dense subgraph $D_{\alpha,\beta}$ is *hierarchical*.

Theorem 3. *Given a graph G, for $\alpha^+ \geq \alpha$ and $\beta^+ \geq \beta$, we have $D_{\alpha^+,\beta^+} \subseteq D_{\alpha,\beta}$.*

Proof. We assume for contradiction that $D = D_{\alpha^+,\beta^+} \setminus D_{\alpha,\beta} \neq \emptyset$. With Theorem 2, $|E(D)| + |E_\times(D, D_{\alpha^+,\beta^+} \setminus D)| > \alpha^+ \cdot |D^U| + \beta^+ \cdot |D^V|$ and $|E(D)| + |E_\times(D, D_{\alpha,\beta})| \leq \alpha \cdot |D^U| + \beta \cdot |D^V|$ hold. Due to $|E_\times(D, D_{\alpha,\beta})| \geq |E_\times(D, D_{\alpha^+,\beta^+} \setminus D)|$, we can derive the contradiction $\alpha^+ \cdot |D^U| + \beta^+ \cdot |D^V| < |E(D)| + |E_\times(D, D_{\alpha^+,\beta^+} \setminus D)| \leq |E(D)| + |E_\times(D, D_{\alpha,\beta})| \leq \alpha \cdot |D^U| + \beta \cdot |D^V|$. Thus, the hierarchy of $D_{\alpha,\beta}$ is established. □

By Theorem 3, all (α, β)-dense subgraphs with different parameters α and β form a hierarchy within a bipartite graph. Specifically, each parameter pair (α, β) defines a distinct layer in this hierarchy. We refer to the task of identifying a single layer in this hierarchy as the problem of finding a dense subgraph and computing all non-empty layers as the problem of density decomposition. Formally, we define our problems as follows:

Problem 1: Dense Subgraph Search (DSS). Given an undirected and unweighted bipartite graph $G = (U, V, E)$ and two integers α and β, the DSS problem is to compute the (α, β)-dense subgraph.

Problem 2: Density Decomposition (DD). Given an undirected and unweighted bipartite graph $G = (U, V, E)$, the DD problem is to find all non-empty (α, β)-dense subgraphs.

Discussions and challenges. For Problem 1, a potential approach to compute the (α, β)-dense subgraph is to iteratively find the $s \rightsquigarrow t$ paths in Definition 1 and subsequently eliminate them by reversing the direction of all edges along these paths. This process continues until no $s \rightsquigarrow t$ paths remain. This approach is similar to the *path reversal* algorithm for density decomposition on unipartite graphs [10]. However, such an approach incurs a high time complexity of $O(|E|^2)$ [10], making it not scalable to large graphs. In this paper, we will develop a novel network flow-based algorithm to accelerate the computation. An inherent challenge is how to design the flow network structure so that the maximum flow algorithm can compute the (α, β)-dense subgraph. Moreover, for real-world graphs with billions of edges, merely traversing the edge set is time-consuming. Thus, it becomes essential to devise reduction techniques that can scale down the graph data, enabling computation of the dense subgraph on a reduced graph and thereby enhancing scalability.

For Problem 2, a straightforward approach to compute the complete density decomposition is to sequentially compute each layer using the single-layer algorithm from Problem 1. However, this approach fails to leverage the hierarchical characteristics of the dense subgraph, resulting in redundant computations and inefficiencies. Therefore, it is important to develop new computation strategies that capitalize on these characteristics. In summary, the key challenges to address are as follows: (1) How to design a flow network to compute the (α, β)-dense subgraph using network flow technique? (2) How to design an effective pruning strategy to reduce the computation of the dense subgraph to a smaller data scale? (3) How to utilize the properties of density decomposition to design efficient computational strategies?

3 Theoretical Relation to (α, β)-core

In this section, we first present a "Sandwich Theorem" to establish the inclusion relation between the (α, β)-dense subgraph and (α, β)-core, which will be used for graph reduction in solving both

the DSS and DD problems. We then highlight the advantage of the (α, β)-dense subgraph over (α, β)-core by the density metric defined on bipartite graphs. Below, we give the definition of (α, β)-core.

Definition 2. **((α, β)-core)** *[17] Given an undirected, unweighted bipartite graph $G = (U, V, E)$ and two positive integers α and β, the (α, β)-core, denoted by $C_{\alpha,\beta}$, is the maximal subgraph where each node $u \in C^U_{\alpha,\beta}$ has a degree $d_u(C_{\alpha,\beta}) \geq \alpha$, and each node $v \in C^U_{\alpha,\beta}$ has a degree $d_v(C_{\alpha,\beta}) \geq \beta$.*

Note that the parameters α and β are *positive* integers for $C_{\alpha,\beta}$, whereas they are *non-negative* integers for $D_{\alpha,\beta}$. These constraints are designed to exclude the trivial case of isolated nodes, as in the previous studies [22, 25].

The Sandwich Theorem. Theorem 4 shows that the (α, β)-dense subgraphs and the (α, β)-cores are sandwiched by each other.

Theorem 4. **(Sandwich Theorem)** *Given an undirected and unweighted bipartite graph $G = (U, V, E)$, we have: $C_{2\alpha+1,2\beta+1} \subseteq D_{\alpha,\beta} \subseteq C_{\alpha+1,\beta+1} \subseteq D_{\lceil \frac{\alpha-1}{2} \rceil, \lceil \frac{\beta-1}{2} \rceil}$.*

Proof. Given an orientation $\vec{G}$ in Definition 1, it satisfies the three properties in Lemma 1.

We first prove that $D_{\alpha,\beta} \subseteq C_{\alpha+1,\beta+1}$. This trivially holds when $D_{\alpha,\beta} = \emptyset$; thus, we focus on the case of $D_{\alpha,\beta} \neq \emptyset$. According to Lemma 1, for any $u \in D^U_{\alpha,\beta}$, we have $\vec{d}_u(D_{\alpha,\beta}) \geq \alpha$. In particular, if $\vec{d}_u(D_{\alpha,\beta}) = \alpha$, we can conclude that u has at least one outgoing edge in $\vec{G}(D_{\alpha,\beta})$, otherwise, by Definition 1, u should not be included in $D_{\alpha,\beta}$. Since the degree of a node in G is equal to the sum of its indegree and outdegree in the orientation $\vec{G}$, we have $d_u(D_{\alpha,\beta}) \geq \alpha + 1$. Similarly, for any $v \in D^V_{\alpha,\beta}$, $d_v(D_{\alpha,\beta}) \geq \beta + 1$ holds. According to Definition 2, we can conclude $D_{\alpha,\beta} \subseteq C_{\alpha+1,\beta+1}$.

Then, we show $C_{\alpha+1,\beta+1} \subseteq D_{\lceil \frac{\alpha-1}{2} \rceil, \lceil \frac{\beta-1}{2} \rceil}$ by analyzing the non-trivial case when $C_{\alpha+1,\beta+1} \neq \emptyset$. Given x be any node in $C_{\alpha+1,\beta+1}$, we aim to prove $x \in D_{\lceil \frac{\alpha-1}{2} \rceil, \lceil \frac{\beta-1}{2} \rceil}$. Let $R = \{y \in C_{\alpha+1,\beta+1} \mid x \text{ can reach } y \text{ in } \vec{G}\}$, and then all edges in $E_\times(R, C_{\alpha+1,\beta+1} \setminus R)$ are oriented toward R in $\vec{G}$. We can further derive $\sum_{y \in R} \vec{d}_y(\vec{G}) \geq |E_\times(R, C_{\alpha+1,\beta+1} \setminus R)| + |E(R)|$. According to the definition of $C_{\alpha+1,\beta+1}$, we have $|E_\times(R, C_{\alpha+1,\beta+1} \setminus R)| + |E(R)| \geq \sum_{y \in R} d_y(C_{\alpha+1,\beta+1})/2 \geq (\alpha+1)|R^U|/2 + (\beta+1)|R^V|/2$. Thus, $\sum_{y \in R} \vec{d}_y(\vec{G}) \geq (\alpha+1)|R^U|/2 + (\beta+1)|R^V|/2$ holds. This implies that at least one node $u \in R^U$ has $\vec{d}_u \geq \lceil \frac{\alpha-1}{2} \rceil + 1$, or at least one node $v \in R^V$ has $\vec{d}_v \geq \lceil \frac{\beta-1}{2} \rceil + 1$, which can be reached by x. Therefore, according to Definition 1, we have $x \in D_{\lceil \frac{\alpha-1}{2} \rceil, \lceil \frac{\beta-1}{2} \rceil}$.

Since $C_{\alpha+1,\beta+1} \subseteq D_{\lceil \frac{\alpha-1}{2} \rceil, \lceil \frac{\beta-1}{2} \rceil}$ holds, we can equivalently derive $C_{2\alpha+1,2\beta+1} \subseteq D_{\alpha,\beta}$. □

Based on the Sandwich Theorem, we can efficiently compute (α, β)-dense subgraphs by pruning the graph with the (α, β)-cores. Such a pruning technique will be used in our proposed algorithms (see Sections 4 and 5).

Advantages of the (α, β)-dense subgraph model. Recall that a good dense subgraph model should be densely connected internally and sparsely connected externally. Given $D_{\alpha,\beta} \neq \emptyset$ and $D_{\alpha,\beta} \neq C_{\alpha+1,\beta+1}$, $D_{\alpha,\beta}$ guarantees that the number of edges lost after deleting any subgraph of $D_{\alpha,\beta}$ is not less than the number of edges gained by adding the external nodes to $D_{\alpha,\beta}$ (Theorem 2). While $C_{\alpha+1,\beta+1}$ intuitively contains both $D_{\alpha,\beta}$ and a sparse redundant part $C_{\alpha+1,\beta+1} \setminus D_{\alpha,\beta}$ (Theorem 4), it does not satisfy the well-behaved property described in Theorem 2. Thus, for $C_{\alpha+1,\beta+1}$, we can typically find a denser (and hence better) subgraph $D_{\alpha,\beta}$ instead of $C_{\alpha+1,\beta+1}$.

Theorem 5 shows another advantage of the (α, β)-dense subgraph model over the (α, β)-core subgraph: it provides a better lower bound for the degree-based density, as defined in the following.

Algorithm 1: DSS(G, α, β)

Input: A bipartite graph G, two non-negative integers α and β.
Output: $D_{\alpha,\beta}$.

```
Arbitrarily obtain an orientation G⃗ of G;
while True do
    S ← {u ∈ U | d⃗_u(G⃗) < α} ∪ {v ∈ V | d⃗_v(G⃗) < β};
    T ← {u ∈ U | d⃗_u(G⃗) > α} ∪ {v ∈ V | d⃗_v(G⃗) > β};
    if there is a path s ⇝ t with s ∈ S, t ∈ T in G⃗ then
        reverse s ⇝ t;
    else break the loop;
D_{α,β} ← T ∪ {x | x can reach a node in T};
return D_{α,β};
```

Definition 3. **(Density)** *[4] Given an undirected and unweighted bipartite graph $G = (U, V, E)$ and a subgraph H, the density of H is defined as $\rho(H) = \frac{|E(H)|}{\sqrt{|H^U| \cdot |H^V|}}$.*

Theorem 5. *Given an undirected and unweighted bipartite graph G and two non-negative integers α and β, we have: (1) $\rho(D_{\alpha,\beta}) > 2\sqrt{\alpha\beta}$ if $D_{\alpha,\beta} \neq \emptyset$; (2) $\rho(C_{\alpha,\beta}) \geq \sqrt{\alpha\beta}$ if $C_{\alpha,\beta} \neq \emptyset$, and this density lower bound is tight.*

Proof. In Theorem 2, let $X = D_{\alpha,\beta}$, we have $E(D_{\alpha,\beta}) > \alpha|D^U_{\alpha,\beta}| + \beta|D^V_{\alpha,\beta}|$. By Definition 3, we can derive $\rho(D_{\alpha,\beta}) = \frac{|E(D_{\alpha,\beta})|}{2\sqrt{|D^U_{\alpha,\beta}||D^V_{\alpha,\beta}|}} > \frac{\alpha|D^U_{\alpha,\beta}|+\beta|D^V_{\alpha,\beta}|}{2\sqrt{|D^U_{\alpha,\beta}||D^V_{\alpha,\beta}|}} \geq 2\sqrt{\alpha\beta}$. In terms of $C_{\alpha,\beta}$, according to Definition 3, we have $\rho(C_{\alpha,\beta}) = \frac{|E(C_{\alpha,\beta})|}{2\sqrt{|C^U_{\alpha,\beta}||C^V_{\alpha,\beta}|}} \geq \frac{\alpha|C^U_{\alpha,\beta}|/2+\beta|C^V_{\alpha,\beta}|/2}{2\sqrt{|C^U_{\alpha,\beta}||C^V_{\alpha,\beta}|}} \geq \sqrt{\alpha\beta}$. When a $C_{\alpha,\beta}$ satisfies that nodes in both $C^U_{\alpha,\beta}$ and $C^V_{\alpha,\beta}$ have degrees α and β respectively, the above inequality becomes an equality. Therefore, the lower-bound is tight. □

According to Theorem 5, to achieve a density lower bound nearly as good as $D_{\alpha,\beta}$, we need to compute $C_{2\alpha,2\beta}$ within the core-based subgraph model. However, for sufficiently large values of α and β satisfying $D_{\alpha,\beta} \neq \emptyset$, the subgraph $C_{2\alpha,2\beta}$ is often empty, making it meaningless. Taking the real-world graph in Figure 11 as an example, when $\alpha = \beta = 185$, $D_{185,185} \neq \emptyset$, but $C_{2\alpha,2\beta} = C_{370,370} = \emptyset$. In fact, the largest integer δ such that $C_{\delta,\delta} \neq \emptyset$ is only 221. This indicates that the density lower bound of $C_{221,221}$ is merely $\sqrt{221 \times 221} = 221$, whereas the density lower bound of $D_{185,185}$ is as high as $2\sqrt{185 \times 185} = 370$. Therefore, our proposed (α, β)-dense subgraph model has the advantage over (α, β)-core in better guaranteeing the density of subgraphs.

4 Dense Subgraph Search Algorithms

This section presents three novel algorithms for finding the (α, β)-dense subgraph. The idea of them is closely based on the definition of dense subgraph, that is, eliminating all $s \rightsquigarrow t$ paths in Definition 1 and then obtaining $D_{\alpha,\beta}$. The difference between these algorithms lies in how the paths are found and eliminated. In the following, we first propose the DSS algorithm, which eliminates the $s \rightsquigarrow t$ paths through multiple rounds of Breadth-First Search (BFS). As each round of BFS can eliminate one $s \rightsquigarrow t$ path and the number of such paths can be quite large, the algorithm is relatively inefficient with time complexity of $O(|E|^2)$. Then we present the DSS+ algorithm, which performs a maximum flow computation to eliminate all $s \rightsquigarrow t$ paths at once, resulting in a time complexity of

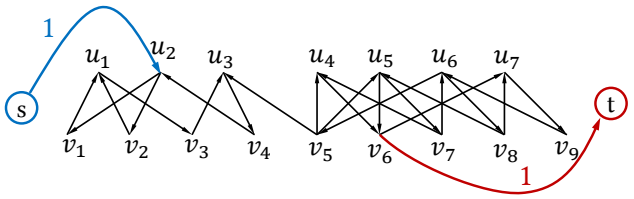


Fig. 3. An example of the re-orientation network.

$O(|E|^{3/2})$; To further improve efficiency, the DSS++ algorithm is developed by utilizing the sandwich theorem (Theorem 4) to perform *core-reduction* pruning strategy on DSS+. The DSS++ algorithm has a time complexity of $O(|E| + |E(R)|^{3/2})$, where $R = C_{\alpha+1,\beta+1} \setminus C_{2\alpha+1,2\beta+1}$ is the reduced graph, which is typically much smaller than the entire graph G.

4.1 The BFS-based Algorithm: DSS

The DSS algorithm uses BFS to find the $s \rightsquigarrow t$ path and then eliminate it based on the *reverse* operation, defined as follows. If a directed edge $\langle x, y\rangle$ is *reversed*, it becomes $\langle y, x\rangle$; If a path is *reversed*, then all its edges are reversed. With the reverse operation, the pseudo-code of the DSS algorithm is shown in Algorithm 1. First, it obtains an orientation by arbitrarily orienting all edges of G (line 1). Then, in the while loop (lines 2-7), the algorithm finds (line 5) and reverses (line 6) all $s \rightsquigarrow t$ paths in Definition 1 through the BFS algorithm. Each iteration of the loop invokes one BFS, removing one $s \rightsquigarrow t$ path. The loop terminates when no $s \rightsquigarrow t$ path can be found (line 7). Finally, the algorithm obtains the (α, β)-dense subgraph $D_{\alpha,\beta}$ according to the definition (line 8).

According to Definition 1, the DSS algorithm correctly outputs the (α, β)-dense subgraph $D_{\alpha,\beta}$. Below, we analyze the complexity of the DSS algorithm.

Theorem 6. *The time and space complexity of the* DSS *algorithm is* $O(|E|^2)$ *and* $O(|E|)$ *respectively.*

Proof. For the arbitrary orientation $\vec{G}$ obtained by line 1 of Algorithm 1, let $\bar{T} = \{u \in U | \vec{d}_u(\vec{G}) > \alpha\} \cup \{v \in V | \vec{d}_v(\vec{G}) > \beta\}$ be the initial set T. The reversal of a $s \rightsquigarrow t$ path cannot add any new nodes to the set T, thus every node t in the $s \rightsquigarrow t$ path must be in $\bar{T}$. As each reversal decreases the indegree of a node t in $\bar{T}$ by 1 and the indegree of a node is non-negative, the number of reversals is clearly bounded by $\sum_{x \in \bar{T}} \vec{d}_x \leq |E|$. Since each $s \rightsquigarrow t$ path can be found and reversed using a BFS with a time complexity of $O(|E|)$, the total time required by DSS is $O(|E|^2)$. Clearly, the space complexity is the input size $O(|E|)$. □

4.2 The Flow-Based Algorithm: DSS+

As mentioned, the DSS algorithm is relatively inefficient because it needs to reverse paths up to $O(|E|)$ times. Here, we introduce the DSS+ algorithm, which accelerates the process by reversing all paths at once using the network flow technique. The rationale behind the DSS+ algorithm is as follows: there is a class of algorithms in network flow known as augmenting path algorithms which eliminate all paths from s to t using the maximum flow computation. Intuitively, by designing the flow network structure appropriately, this path elimination technique can also be used to remove paths in the definition of (α, β)-dense subgraph. Therefore, inspired by the re-orientation network of unipartite graphs [8], we propose a re-orientation network on bipartite graphs.

Definition 4. **(Re-orientation network)** *Given an orientation* $\vec{G} = (U, V, \vec{E})$ *of a bipartite graph and two integers* α *and* β*, the re-orientation network is a triplet* $(V \cup U \cup \{s, t\}, A, c(\cdot))$*, where* s *and* t *are the source and sink nodes, respectively,* A *is the arc set, and* $c(\cdot)$ *is the capacity function, satisfying:*

Algorithm 2: DSS+(G, α, β)

Input: A bipartite graph G, two non-negative integers α and β.
Output: $D_{\alpha,\beta}$.

1 Arbitrarily obtain an orientation $\vec{G}$ of G;
2 **foreach** $x \in \vec{G}$ **do** $\vec{d}_x \leftarrow$ the indegree of x in $\vec{G}$;
3 $S \leftarrow \{u \in U | \vec{d}_u(\vec{G}) < \alpha\} \cup \{v \in V | \vec{d}_v(\vec{G}) < \beta\}$;
4 $T \leftarrow \{u \in U | \vec{d}_u(\vec{G}) > \alpha\} \cup \{v \in V | \vec{d}_v(\vec{G}) > \beta\}$;
5 **foreach** $\langle x, y\rangle \in \vec{G}$ **do**
6 Add arc $\langle x, y\rangle$ to A; $c(\langle x, y\rangle) \leftarrow 1$;
7 **foreach** $x \in S$ **do**
8 Add arc $\langle s, x\rangle$ to A;
9 **if** $x \in U$ **then** $c(\langle s, x\rangle) \leftarrow \alpha - \vec{d}_x$;
10 **else** $c(\langle s, x\rangle) \leftarrow \beta - \vec{d}_x$;
11 **foreach** $x \in T$ **do**
12 Add arc $\langle x, t\rangle$ to A;
13 **if** $x \in U$ **then** $c(\langle x, t\rangle) \leftarrow \vec{d}_x - \alpha$;
14 **else** $c(\langle x, t\rangle) \leftarrow \vec{d}_x - \beta$;
15 Compute the maximum flow of network A with capacity $c(\cdot)$;
16 **return** $D_{\alpha,\beta} = \{$*all nodes can reach t in the residual network*$\}$;

(1) $e = \langle x, y\rangle \in A$, $c(e) = 1$, *if* $e \in \vec{E}$; *(2)* $e = \langle s, u\rangle \in A$, $c(e) = \alpha - \vec{d}_u$, *if* $u \in U$ *and* $\vec{d}_u < \alpha$; *(3)* $e = \langle s, v\rangle \in A$, $c(e) = \beta - \vec{d}_v$, *if* $v \in V$ *and* $\vec{d}_v < \beta$; *(4)* $e = \langle u, t\rangle \in A$, $c(e) = \vec{d}_u - \alpha$, *if* $u \in U$ *and* $\vec{d}_u > \alpha$; *(5)* $e = \langle v, t\rangle \in A$, $c(e) = \vec{d}_v - \beta$, *if* $v \in V$ *and* $\vec{d}_v > \beta$.

Let $S = \{u \in U | \vec{d}_u(\vec{G}) < \alpha\} \cup \{v \in V | \vec{d}_v(\vec{G}) < \beta\}$ and $T = \{u \in U | \vec{d}_u(\vec{G}) > \alpha\} \cup \{v \in V | \vec{d}_v(\vec{G}) > \beta\}$, then in the re-orientation network, the source s is connected to all nodes in S, and the sink t is connected to all nodes in T. When the network flow reaches its maximum value, there is no augmenting path from s to t, i.e., there should be no path from S to T, which is consistent with the definition of (α, β)-dense subgraph. Thus, the re-orientation network flow technique can be used to eliminate paths in Definition 1 and search the (α, β)-dense subgraph.

Example 3. *Given the orientation in Figure 2 and* $\alpha = 2$, $\beta = 1$, *the corresponding re-orientation network is shown in Figure 3, where the capacity of each arc is 1. For nodes in* U, α *is the pivot of indegree, and for nodes in* V, β *is the pivot of indegree. Source* s *is connected to the nodes whose indegree does not reach the pivot, thus it is connected to* u_2 *with a capacity of* $\alpha - \vec{d}_{u_2} = 1$. *Sink* t *is connected to the nodes whose indegree exceeds the pivot, thus it is connected to* v_6 *with a capacity of* $\vec{d}_{v_6} - \beta = 1$. *Currently,* s *cannot reach* t, *which corresponds to "there is no path* $s \rightsquigarrow t$*" in Definition 1, and the set* $T \cup \{x | x$ *can reach a node in* T *in* $\vec{G}\}$ *contains all nodes that can reach the sink* t, *i.e.,* $D_{\alpha,\beta} = \{u_4, \cdots, u_7\} \cup \{v_5, \cdots, v_9\}$.

Based on the re-orientation network, we propose the DSS+ algorithm as outlined in Algorithm 2. The algorithm first arbitrarily obtains an orientation (line 1) and then constructs the re-orientation network (lines 5-14). After that, it computes the maximum flow on it (line 15). Finally, it obtains $D_{\alpha,\beta}$ by computing all nodes that can reach t and returns it as the (α, β)-dense subgraph $D_{\alpha,\beta}$ (line 16). Below we prove the correctness and complexity of DSS+.

Theorem 7. *The* DSS+ *algorithm can output* $D_{\alpha,\beta}$ *correctly with a time complexity of* $O(|E|^{1.5})$ *and a space complexity of* $O(|E|)$.

Algorithm 3: DSS++$(G, \alpha, \beta, D_l, D_u)$

Input: A bipartite graph G, two non-negative integers α and β, two subgraph D_l and D_u satisfying $D_u \subseteq D_{\alpha,\beta} \subseteq D_l$.
Output: $D_{\alpha,\beta}$.

```
1 Let R be the induced subgraph of D_l \ D_u;      // Reduction
2 Arbitrarily obtain an orientation R⃗ of R;
3 foreach x ∈ R⃗ do d⃗_x ← the indegree of x in R⃗;
4 foreach (x, y) ∈ E×(R, D_u), x ∈ D_u, y ∈ R do d⃗_y ← d⃗_y + 1;
5 S ← {u ∈ R^U | d⃗_u(R⃗) < α} ∪ {v ∈ R^V | d⃗_v(R⃗) < β};
6 T ← {u ∈ R^U | d⃗_u(R⃗) > α} ∪ {v ∈ R^V | d⃗_v(R⃗) > β};
7 Same as lines 5-15 of Algorithm 2;
8 return {all nodes can reach t in the residual network} ∪ D_u;
```

Proof. After the maximum flow algorithm is completed, suppose for each *saturated* edge $\langle x, y\rangle \in \vec{E}$ in A, it is reversed in $\vec{G}$, then in $\vec{G}$, the set $S = \{u \in U | \vec{d}_u(\vec{G}) < \alpha\} \cup \{v \in V | \vec{d}_v(\vec{G}) < \beta\}$ will contain only the nodes in the residual network connected to s by unsaturated edges, and the set $T = \{u \in U | \vec{d}_u(\vec{G}) > \alpha\} \cup \{v \in V | \vec{d}_v(\vec{G}) > \beta\}$ will contain only the nodes in the residual network connected to t by unsaturated edges. Because there is no path $s \rightsquigarrow t$ in the residual network, there is also no path from the set S to the set T in $\vec{G}$, which conforms to the definition of a (α, β)-dense subgraph. Furthermore, all nodes that can reach t in the residual network are exactly those that can reach T in the (α, β)-dense subgraph definition plus T, thus the set output by the DSS+ algorithm is $D_{\alpha,\beta}$.

For the time complexity, it can be derived from [9] that the re-orientation network belongs to the so-called AUC-2 network (i.e., nearly all weights of the edges in the flow network are 1 except the weights of the edges connected to the source and sink nodes). The time and space complexity of the maximum flow computation on such an AUC-2 network is $O(|E|^{1.5})$ and $O(|E|)$ respectively [9]. Therefore, the time and space complexity of DSS+ is $O(|E|^{1.5})$ and $O(|E|)$ respectively. □

4.3 The Improved Flow-Based Algorithm: DSS++

Although DSS+ reduces the time complexity to $O(|E|^{1.5})$, it is still relatively time-consuming for large real-world graphs. Further acceleration can be achieved by seeking pruning strategies to reduce the computation scale from the entire graph to a small part of the graph. To this end, we develop the **core reduction** pruning strategy based on the Sandwich Theorem and propose an improved flow-based algorithm DSS++, which computes the $D_{\alpha,\beta}$ within the subgraph $C_{\alpha+1,\beta+1} \setminus C_{2\alpha+1,2\beta+1}$ rather than the entire graph.

The core reduction pruning strategy. Without loss of generality, we assume that there are two subgraphs D_l and D_u satisfying $D_u \subseteq D_{\alpha,\beta} \subseteq D_l$. Given that $D_{\alpha,\beta} \subseteq D_l$, we can safely disregard nodes outside of D_l and compute $D_{\alpha,\beta}$ only in the induced subgraph of D_l. On the other hand, directly ignoring D_u and computing $D_{\alpha,\beta}$ in the induced subgraph of $D_l \setminus D_u$ using the re-orientation network is incorrect because this approach does not consider the influence of $E_\times(D_l \setminus D_u, D_u)$ on computing $D_{\alpha,\beta}$. To resolve this issue, we design the following approach: for each $(x, y) \in E_\times(D_l \setminus D_u, D_u)$ where $x \in D_u$ and $y \in D_l \setminus D_u$, we assign y an additional unit indegree in the re-orientation network. In other words, all the edges in $E_\times(D_l \setminus D_u, D_u)$ are treated as oriented toward $D_l \setminus D_u$. With such additional indegree assignment, we can compute $D_{\alpha,\beta}$ correctly in the induced subgraph of $D_l \setminus D_u$ using the re-orientation network.

Based on the above idea and the sandwich theorem, we can set $D_l = C_{\alpha+1,\beta+1}$, $D_u = C_{2\alpha+1,2\beta+1}$, thus reducing the computation of $D_{\alpha,\beta}$ from the entire graph to $C_{\alpha+1,\beta+1} \setminus C_{2\alpha+1,2\beta+1}$. The DSS++ algorithm equipped with such core reduction technique is depicted in Algorithm 3. Note that the algorithm requires two additional subgraphs $D_l = C_{\alpha+1,\beta+1}$ and $D_u = C_{2\alpha+1,2\beta+1}$ as inputs. In line 1, the algorithm derives the reduced subgraph R, then arbitrarily obtains its orientation $\vec{R}$ (line 2). To handle the edges in $E_\times(D_l \setminus D_u, D_u)$, the algorithm assigns an additional unit of indegree to y for each edge (x, y) (line 4). Then, it constructs the re-orientation network and computes the network flow on the induced subgraph of R based on the current indegree (lines 5-7). Finally, the returned subgraph is united with D_u (line 8). Below, we prove the correctness and complexity of DSS++.

Theorem 8. *The* DSS++ *algorithm can correctly output $D_{\alpha,\beta}$ within $O(|E| + |E(R)|^{1.5})$ time using $O(|E|)$ space, where $R = D_l \setminus D_u$ with $D_l = C_{\alpha+1,\beta+1}$ and $D_u = C_{2\alpha+1,2\beta+1}$ respectively.*

Proof. To prove the correctness, we need to prove that assigning the indegree of edges in $E_\times(D_l \setminus D_u, D_u)$ to $D_l \setminus D_u$ is correct, that is, to prove (1) it allows the nodes in $D_{\alpha,\beta}$ to output correctly, and (2) it does not cause nodes outside $D_{\alpha,\beta}$ to output incorrectly. For (1), assume for contradiction that some nodes $H \subseteq R \cap D_{\alpha,\beta}$ belongs to $D_{\alpha,\beta}$ but is not output correctly, then the indegree of nodes in H in the re-orientation network is $\sum_{x\in H} \vec{d}_x \le \alpha \cdot |H^U| + \beta \cdot |H^V|$, thus $|E(H)| + |E_\times(H, D_{\alpha,\beta} \setminus H)| \le \alpha \cdot |H^U| + \beta \cdot |H^V|$. However, according to Theorem 2, we have $|E(H)| + |E_\times(H, D_{\alpha,\beta} \setminus H)| > \alpha \cdot |H^U| + \beta \cdot |H^V|$, which is a contradiction. For (2), we can use a similar method assuming for contradiction that some node $H \in (U \cup V) \setminus D_{\alpha,\beta}$ is output to prove.

To prove the complexity, it takes $O(|E|)$ time complexity to compute $C_{\alpha+1,\beta+1}$ and $C_{2\alpha+1,2\beta+1}$ [17] and handle the edges in $E_\times(R, D_u)$, and since other calculations are conducted in the induced subgraph of R, the network flow only requires $O(|E(R)|^{1.5})$ time. Summing up, the time complexity is $O(|E| + |E(R)|^{1.5})$, and the space complexity is graph size $O(|E|)$. □

We can first calculate $C_{\alpha+1,\beta+1}$ and $C_{2\alpha+1,2\beta+1}$ and then call DSS++$(G, \alpha, \beta, C_{\alpha+1,\beta+1}, C_{2\alpha+1,2\beta+1})$ to compute $D_{\alpha,\beta}$. Since $|E(R)|$ is often much smaller than $|E|$, this core reduction method can significantly reduce the computation scale and time, as confirmed by our experiments.

5 Density Decomposition Algorithms

In this section, we propose two algorithms for density decomposition computation, namely DD and DD+, to find all non-empty (α, β)-dense subgraphs. The DD algorithm uses the dense subgraph search algorithms to compute the density decomposition layer by layer. However, this basic algorithm involves a large number of redundant computations, making it inefficient. Therefore, a novel and improved algorithm DD+ based on the hierarchical characteristics of dense subgraphs is presented, which greatly reduces redundant computations by using a carefully-designed divide-and-conquer technique to continuously reduce the graph.

5.1 The Basic Algorithm: DD

Using the dense subgraph search algorithm DSS++, the density subgraph decomposition can be computed layer by layer. Based on this idea, we propose the DD algorithm as shown in Algorithm 4. Specifically, DD enumerates all combinations of α and β through two nested loops (lines 1-2). After fixing α and β, the algorithm prepares the subgraphs D_l and D_u as the two inputs for DSS++ (lines 3-5), and then invokes DSS++$(G, \alpha, \beta, D_l, D_u)$ (line 6). If $D_{\alpha,\beta} = \emptyset$, it indicates that for the current α, the maximum value of β has been enumerated, and the algorithm should proceed to the next α (lines 7-8). If for some enumeration of β, even when $\beta = 0$, $D_{\alpha,0}$ is empty, it means that the current

Algorithm 4: DD(G)

Input: A bipartite graph G.
Output: All non-empty $D_{\alpha,\beta}$.

1 **for** $\alpha = 0, 1, \cdots$ **do**
2 **for** $\beta = 0, 1, \cdots$ **do**
3 $D_l \leftarrow C_{\alpha+1,\beta+1}$; $D_u \leftarrow C_{2\alpha+1,2\beta+1}$;
4 **if** $D_{\alpha-1,\beta}$ *has been computed* **then** $D_l \leftarrow D_l \cup D_{\alpha-1,\beta}$;
5 **if** $D_{\alpha,\beta-1}$ *has been computed* **then** $D_l \leftarrow D_l \cup D_{\alpha,\beta-1}$;
6 $D_{\alpha,\beta} \leftarrow$ DSS++$(G, \alpha, \beta, D_l, D_u)$;
7 **if** $D_{\alpha,\beta} \neq \varnothing$ **then** output $D_{\alpha,\beta}$;
8 **else** break the loop;
9 **if** $D_{\alpha,0} = \varnothing$ **then return**;

α is sufficiently large, and all non-empty $D_{\alpha,\beta}$ have been enumerated, so the algorithm returns (line 9).

The correctness of DD can be directly derived from the correctness of DSS++, thus we omit its correctness proof. For the complexity of DD, its time and space complexity can be bounded by $O(layer \cdot |E|^{1.5})$ and $O(|E|)$ respectively, where $layer$ is the number of layers in the dense subgraph decomposition, i.e., the number of combinations of α and β that make $D_{\alpha,\beta}$ non-empty. Below, we give a lower bound of $layer$.

Theorem 9. *Given G, let $d^U_{\max} = \max_{u\in U} d_u$, $d^V_{\max} = \max_{v\in V} d_v$. We have the maximum $\alpha_{\max}$ (resp., $\beta_{\max}$) such that $D_{\alpha_{\max},0} \neq \varnothing$ (resp., $D_{0,\beta_{\max}} \neq \varnothing$) equals to $d^U_{\max} - 1$ (resp., $d^V_{\max} - 1$), and the number of layers in the density decomposition satisfies $layer \geq d^U_{\max} + d^V_{\max} + p^2 - 1$, where p is the maximum number such that $D_{p,p} \neq \varnothing$.*

Proof. Construct such an orientation: all edges are oriented toward the set U, then for any α and $\beta = 0$, the orientation meets the definition of a dense subgraph. Clearly, according to the definition, $D_{\alpha,0}$ consists of all nodes in the set U with a degree greater than α and their neighbors. Therefore, $D_{\alpha,0}$ is non-empty only when $\alpha \leq d^U_{\max} - 1$, and similarly, $D_{0,\beta}$ is non-empty only when $\beta \leq d^V_{\max} - 1$. In summary, $D_{0,0}, \cdots, D_{d^U_{\max}-1,0}$, these $d^U_{\max}$ subgraphs are non-empty; $D_{0,0}, \cdots, D_{0,d^V_{\max}-1}$, these $d^V_{\max}$ subgraphs are non-empty; and $D_{\alpha,\beta}$ is also non-empty for any $\alpha = 1, \cdots, p$ and $\beta = 1, \cdots, p$. Putting it all together, we have $layer \geq d^U_{\max} + d^V_{\max} + p^2 - 1$, where the subtraction of 1 is due to the repeated calculation of one $D_{0,0}$. □

Real-world large graphs may have a large $d^U_{\max}$ and $d^V_{\max}$ (e.g., in our experiments, the $d^V_{\max}$ of Twitter is nearly 3 million), thus $layer$ can also be very large, impacting the scalability of algorithm DD. The inefficiency arises due to DD requiring numerous repetitive computations. Specifically, prior to enumerating $D_{\alpha,\beta}$, the algorithm computes $D_{\alpha,0}, D_{\alpha,1}, \cdots, D_{\alpha,\beta-1}$, with these calculations often involving $D_{\alpha,\beta}$ itself. Consequently, $D_{\alpha,\beta}$ ends up being recalculated up to β times. This redundancy becomes pronounced when β is large, significantly hampering efficiency.

5.2 The Improved Algorithm: DD+

To address the issue of redundant calculations in DD, we propose a novel divide-and-conquer technique. Assuming that α is now fixed, we need to calculate all $D_{\alpha,0}, D_{\alpha,1}, D_{\alpha,2}, \cdots$. According to the idea of DD, it calculates $D_{\alpha,0}, D_{\alpha,1}, \cdots$ sequentially. However, using the divide-and-conquer technique, we can find a β_m and compute D_{α,β_m} and D_{α,β_m+1} first. Then, based on the hierarchy

Algorithm 5: DD+(G)

Input: A bipartite graph G.
Output: All non-empty $D_{\alpha,\beta}$.

1 $p \leftarrow$ the maximum integral such that $D_{p,p} \neq \varnothing$;
2 $d^U_{\max} \leftarrow \max_{u\in U} d_u(G), d^V_{\max} \leftarrow \max_{v\in V} d_v(G)$;
3 **for** $\alpha = 0, 1, \cdots, p$ **do**
4 $\quad T = \{u \in U | d_u(G) > \alpha\}$;
5 $\quad D_{\alpha,0} \leftarrow T \cup \{$the neighbor nodes of $T\}$; $D_{\alpha,d^V_{\max}} \leftarrow \varnothing$;
6 $\quad$ Divide-a$(D_{\alpha,0}, D_{\alpha,d^V_{\max}})$;
7 **for** $\beta = 0, 1, \cdots, p$ **do**
8 $\quad T = \{v \in V | d_v(G) > \beta\}$;
9 $\quad D_{0,\beta} \leftarrow T \cup \{$the neighbor nodes of $T\}$; $D_{d^U_{\max},\beta} \leftarrow \varnothing$;
10 $\quad$ Divide-b$(D_{0,\beta}, D_{d^U_{\max},\beta})$;
11 **Function** Divide-a$(D_{\alpha,\beta_l}, D_{\alpha,\beta_u})$
12 $\quad$ **if** $\beta_u - \beta_l \leq 1$ *or* $D_{\alpha,\beta_l} = D_{\alpha,\beta_u}$ **then return**;
13 $\quad$ Perform binary search to find the maximum integral β_m such that $|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_m})| < |E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})|/2$;
14 $\quad D_{\alpha,\beta_m} \leftarrow$ DSS++$(G, \alpha, \beta_m, D_{\alpha,\beta_l}, D_{\alpha,\beta_u})$;
15 $\quad D_{\alpha,\beta_m+1} \leftarrow$ DSS++$(G, \alpha, \beta_m + 1, D_{\alpha,\beta_l}, D_{\alpha,\beta_u})$;
16 $\quad$ Output D_{α,β_m} and D_{α,β_m+1};
17 $\quad$ Divide-a$(D_{\alpha,\beta_l}, D_{\alpha,\beta_m})$;
18 $\quad$ Divide-a$(D_{\alpha,\beta_m+1}, D_{\alpha,\beta_u})$;
19 **Function** Divide-b$(D_{\alpha_l,\beta}, D_{\alpha_u,\beta})$
20 $\quad$ Same as lines 6-12 but interchanging α with β;

of the dense subgraphs, we can continue to calculate $D_{\alpha,0}, D_{\alpha,1}, \cdots, D_{\alpha,\beta_m-1}$ in $D_{\alpha,0} \setminus D_{\alpha,\beta_m}$ instead of the whole graph, thus reducing the computation scale. On the other hand, we can continue to calculate $D_{\alpha,\beta_m+2}, D_{\alpha,\beta_m+3}, \cdots$ in D_{α,β_m+1} instead of the whole graph. The advantage of this approach is that when calculating in $D_{\alpha,0} \setminus D_{\alpha,\beta_m}$, the computation scale does not involve D_{α,β_m}, ensuring that the nodes in D_{α,β_m} are not calculated redundantly.

Therefore, a Divide-a$(D_{\alpha,\beta_l}, D_{\alpha,\beta_u})$ function can be designed, which inputs subgraphs D_{α,β_l} and D_{α,β_u}, and then computes all subgraphs between them, $D_{\alpha,\beta_l+1}, D_{\alpha,\beta_l+2}, \cdots, D_{\alpha,\beta_u-1}$. The computation method is divide-and-conquer, that is, first determining a pivot number β_m between β_l and β_u, then calling the DSS++ algorithm to compute D_{α,β_m} and D_{α,β_m+1}, and then recursively calling Divide-a$(D_{\alpha,\beta_l}, D_{\alpha,\beta_m})$ and Divide-a$(D_{\alpha,\beta_m+1}, D_{\alpha,\beta_u})$. The recursion ends when $\beta_u - \beta_l \leq 1$ or $D_{\alpha,\beta_l} = D_{\alpha,\beta_u}$. Clearly, in these two cases, there are no finer subgraphs between D_{α,β_l} and D_{α,β_u}. Through this recursive divide-and-conquer strategy, calling Divide-a$(D_{\alpha,\beta_l}, D_{\alpha,\beta_u})$ can compute all subgraphs $D_{\alpha,\beta_l+1}, \cdots, D_{\alpha,\beta_u-1}$.

The remaining question is how to select β_m. Since β_m serves as the pivot number to divide the entire graph, we need to ensure that the divided two subgraphs are as balanced as possible. Otherwise, computing the larger part will be time-consuming. Therefore, β_m should aim to split the current edge set into two roughly equal parts, i.e., ensuring that both $|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_m})|$ and $|E(D_{\alpha,\beta_m+1} \setminus D_{\alpha,\beta_u})|$ are less than half of $|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})|$. This can be achieved by selecting β_m as the maximum integer such that $|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_m})| < \frac{1}{2}|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})|$, guaranteeing that the two subdivided subgraphs are both smaller than half of the original graph.

Based on the above analysis, the density decomposition can be calculated as follows: starting from 0, incrementally enumerate α. For each α, start the recursion by calling Divide-a$(D_{\alpha,0}, D_{\alpha,d^V_{\max}})$,

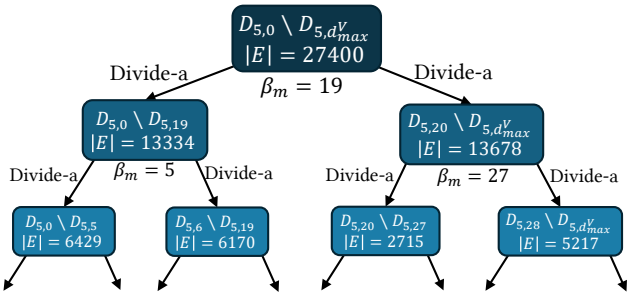


Fig. 4. An example of Divide-a($D_{5,0}, D_{5,d^V_{\max}}$) on DBpedia dataset.

where $D_{\alpha,0}$ can be obtained by the method in the proof of Theorem 9, and $D_{\alpha,d^V_{\max}} = \emptyset$. After that, all non-empty $D_{\alpha,0}, D_{\alpha,1}, \cdots$ for the current α can be obtained. Once α is enumerated to $d^U_{\max}$, the density decomposition can be obtained. However, because $d^U_{\max}$ can be very large, such an enumeration method will call Divide-a too many times, causing inefficiency. A Divide-b function can be symmetrically designed, and then the enumeration strategy can be changed as follows: similarly, first enumerate α, but only enumerate it to p, where p is the maximum number such that $D_{p,p} \neq \emptyset$. This way, Divide-a can calculate all $D_{\alpha,\beta}$ with α less than or equal to p. Then, we enumerate β, from 0 to p. This way, Divide-b can symmetrically calculate all $D_{\alpha,\beta}$ with β less than or equal to p. Since p is the maximum integer such that $D_{p,p} \neq \emptyset$, this enumeration method can calculate all layers in the density decomposition with only calling Divide-a or Divide-b for $2p + 2$ times, which is typically much smaller than $d^U_{\max}$, as confirmed in our experiments.

We develop the DD+ algorithm (Algorithm 5) based on this above idea. First, the algorithm calculates p (line 1), which can be achieved through binary search. Then, the algorithm enumerates α (line 3). For a specific α, it first calculates $D_{\alpha,0}$ and $D_{\alpha,d^V_{\max}}$, then calls Divide-a($D_{\alpha,0}, D_{\alpha,d^V_{\max}}$) to compute dense subgraphs (lines 5-6). When calling Divide-a, the algorithm first checks whether the recursion endpoint is reached (line 12), then uses binary search to determine the pivot number β_m (line 13). After calculating D_{α,β_m} and D_{α,β_m+1}, it begins deeper recursion (lines 14-18). After α is enumerated to p, the algorithm symmetrically enumerates β (lines 7-10). Once β is also enumerated to p, the algorithm has calculated all layers in the density decomposition. Below, we give an illustrative example of Divide-a.

Example 4. *Assume that the dense subgraph is computed by calling* Divide-a($D_{5,0}, D_{5,d^V_{\max}}$) *on the dataset* DBpedia *(details in the experiments section), as shown in Figure 4. The root node in the figure represents the starting point of the divide-and-conquer process, where the subgraph $D_{5,0} \setminus D_{5,d^V_{\max}}$ is divided, and the number of edges in the induced subgraph is 27400. By using binary search, $\beta_m = 19$ is determined. Then, the algorithm calls* Divide-a($D_{5,0}, D_{5,19}$) *and* Divide-a($D_{5,20}, D_{5,d^V_{\max}}$)*, respectively calculating the subgraphs $D_{5,0} \setminus D_{5,19}$ and $D_{5,20} \setminus D_{5,d^V_{\max}}$, with the induced subgraphs having 13334 and 13678 edges respectively, both less than half of 27400. The process is repeated to further divide the subgraphs, reducing the edge set by half each time, which rapidly reduces the computation scale, significantly enhancing efficiency.*

The following theorem shows the correctness of Algorithm 5.

Theorem 10. DD+ *(Algorithm 5) can correctly compute the density decomposition of the input bipartite graph.*

Proof. For Divide-a($D_{\alpha,\beta_l}, D_{\alpha,\beta_u}$), since the divide-and-conquer strategy only limits the scale of the algorithm's computation, each layer still needs to call DSS++ to compute. Furthermore,

line 12 of DD+ ensures that every non-empty $D_{\alpha,\beta}$ can be computed. Therefore, based on the hierarchy of dense subgraph and the correctness of DSS++, Divide-a($D_{\alpha,\beta_l}, D_{\alpha,\beta_u}$) can output all non-empty $D_{\alpha,\beta_l+1}, \cdots, D_{\alpha,\beta_u-1}$. Similarly, Divide-b($D_{\alpha_l,\beta}, D_{\alpha_u,\beta}$) can also output all non-empty $D_{\alpha_l+1,\beta}, \cdots, D_{\alpha_u-1,\beta}$. Since p is the maximum integer satisfying $D_{p,p} \neq \emptyset$, enumerating α and β to p respectively can compute all layers of the density decomposition. □

We analyze the time and space complexity of DD+ in the following theorem.

Theorem 11. *The time and space complexity of* DD+ *is* $O(p \cdot |E|^{1.5} \cdot \log d_{\max})$ *and* $O(|E| + |U \cup V| \log |E|)$, *respectively, where* $d_{\max} = \max_{x \in U \cup V} d_x(G)$.

Proof. It is easy to show that the time complexity for DD+ to obtain p in line 1 through binary search is $O(|E|^{1.5} \log d_{\max})$. Next, we analyze the complexity of Divide-a($D_{\alpha,\beta_l}, D_{\alpha,\beta_u}$). Since all computations in Divide-a can be completed within the induced subgraph of $D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u}$, the binary search in line 13 can be completed in $O(|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})|^{1.5} \log d_{\max})$ time. Since β_m is the maximum integer such that $|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_m})| < |E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})|/2$, we have $|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_m+1})| \geq |E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})|/2$. Furthermore, since $|E(D_{\alpha,\beta_m+1} \setminus D_{\alpha,\beta_u})| \leq |E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})| - |E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_m+1})|$, we get $|E(D_{\alpha,\beta_m+1} \setminus D_{\alpha,\beta_u})| \leq |E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})|/2$. This indicates that the Divide-a algorithm reduces the computational scale by half each recursion. According to the master theorem for divide-and-conquer [5], the complexity of deeper recursion in lines 17-18 does not exceed the binary search complexity in line 13. Thus, the time complexity of Divide-a($D_{\alpha,\beta_l}, D_{\alpha,\beta_u}$) is $O(|E(D_{\alpha,\beta_l} \setminus D_{\alpha,\beta_u})|^{1.5} \log d_{\max}) \leq O(|E|^{1.5} \log d_{\max})$. Since the DD+ algorithm calls Divide-a and Divide-b $2p + 2$ times with a time complexity of $O(|E|^{1.5} \log d_{\max})$, the total time complexity of the DD+ algorithm is $O(p|E|^{1.5} \log d_{\max})$.

For the space complexity, besides the graph size $O(|E|)$, each level of recursion in Divide-a and Divide-b needs to store the input subgraphs in memory, which requires $O(|U \cup V|)$ space. Since each level of recursion reduces the number of edges by at least half, the recursion depth is at most $O(\log |E|)$. Therefore, the space complexity of DD+ is bounded by $O(|E| + |U \cup V| \log |E|)$. □

If we consider a bipartite graph as a unipartite graph and set $\alpha = \beta$, then the problem of computing p can be converted into the problem of computing *pseudoarboricity* [8]. It can be shown that p equals pseudoarboricity minus 1 [8], where pseudoarboricity is a measure of the sparsity of a graph. As observed in [9], pseudoarboricity tends to be a small constant in real-world graphs, thus p is typically small. Therefore, compared to the $O(layer \cdot |E|^{1.5})$ time complexity of DD, the $O(p \cdot \log d_{\max} \cdot |E|^{1.5})$ time complexity of DD+ represents a significant improvement. This is because $layer \geq d_{\max}^U + d_{\max}^V + p^2 - 1$ is typically much larger than $p \cdot \log d_{\max}$ in real-world graphs. On the other hand, although the space complexity of DD+ is $O(|E| + |V| \log |E|)$, in large real-world graphs it is often the case that $|E| > |V| \log |E|$. Thus, the space complexity of DD+ is practically linear, as confirmed in our experiments.

6 Experiments

Algorithms. We have implemented five algorithms: three (α, β)-dense subgraph search algorithms, DSS (Algorithm 1), DSS+ (Algorithm 2), and DSS++ (Algorithm 3), along with two density decomposition algorithms, DD (Algorithm 4) and DD+ (Algorithm 5). Note that since our (α, β)-dense subgraph is a novel model specifically designed for bipartite graphs, there are no existing algorithms capable of computing (α, β)-dense subgraphs or density decomposition. Consequently, we employ the basic versions of our algorithms, DSS and DD, as baseline methods for addressing the (α, β)-dense subgraph search and density decomposition problems, respectively. All algorithms are

Table 1. Statistics of datasets

$d_{\max}^U = \max_{u \in U} d_u$, $d_{\max}^V = \max_{v \in V} d_v$, p is the maximum integer such that $D_{p,p} \neq \emptyset$. $1K = 1,000$, $1M = 1,000,000$, and $1B = 1,000,000,000$.

Name	$\lvert U \rvert$	$\lvert V \rvert$	$\lvert E \rvert$	$d_{\max}^U$	$d_{\max}^V$	p
DBpedia	258.8K	7.8K	463.5K	31	24,821	7
Digg	139.4K	3.6K	3.0M	10,526	24,099	186
Enron	39.9K	28.1K	3.7M	2,120	7,190	212
IMDB	303.6K	896.3K	3.8M	1,334	1,590	20
Livejournal	3.2M	7.5M	112.3M	300	1,053,676	104
Yahoo	1.0M	625.0K	256.8M	307,205	468,366	1,006
Orkut	2.8M	8.7M	327.0M	40,425	318,240	438
Twitter	35.7M	40.1M	1.5B	770,155	2,997,469	1,427

implemented in C++ with O3 optimization. All experiments were conducted on a Linux system PC with a 2.2GHz AMD 3990X 64-Core CPU and 256GB memory.

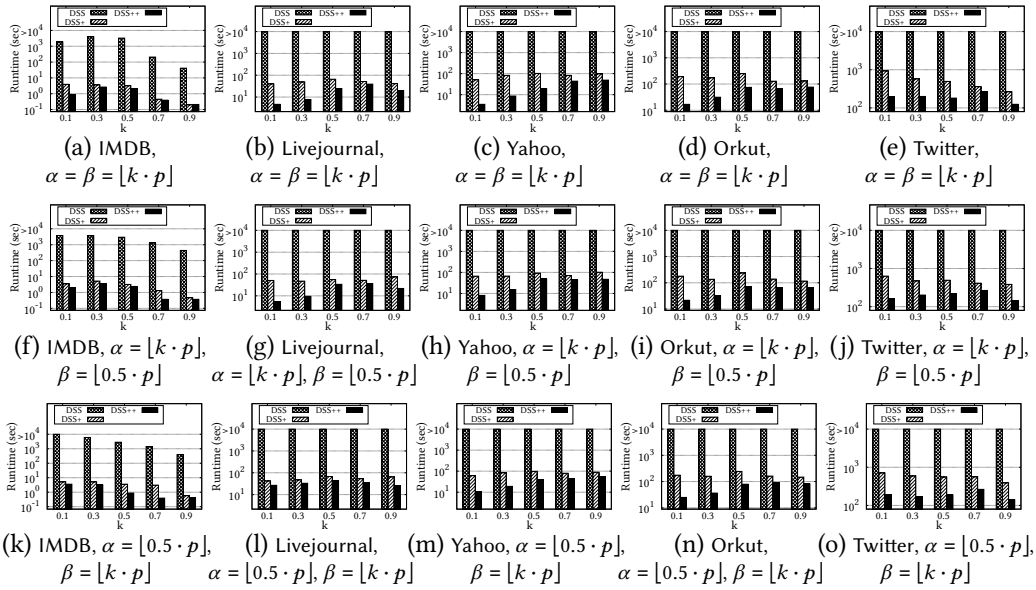
Datasets. We use 8 real-world datasets to evaluate the efficiency of different algorithms in our experiments. These datasets can be downloaded from the Koblenz Network Collection (http://konect.cc/). The detailed statistics are shown in Table 1.

6.1 Performance Studies

Exp-1: Runtime of different dense subgraph search algorithms. In this experiment, we compare the runtime of various dense subgraph search algorithms, with the results on 5 large datasets shown in Figure 5. For a comprehensive comparison, we use three methods for setting the parameters α and β: (1) $\alpha = \beta = \lfloor k \cdot p \rfloor$; (2) $\alpha = \lfloor k \cdot p \rfloor$, $\beta = \lfloor 0.5 \cdot p \rfloor$; (3) $\alpha = \lfloor 0.5 \cdot p \rfloor$, $\beta = \lfloor k \cdot p \rfloor$, where k is chosen from $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. As can be seen, DSS is relatively inefficient, with runtime exceeding 10^4 seconds across all datasets. In contrast, both the DSS+ and DSS++ algorithms can output the (α, β)-dense subgraph in less than 10^3 seconds within all parameter settings. Moreover, the DSS++ algorithm is significantly faster than DSS+ on large graphs with the benefit of the core reduction technique. For instance, on the Twitter dataset, when $\alpha = \beta = \lfloor 0.1 \cdot p \rfloor$, the runtimes of DSS+ and DSS++ are 936.25 seconds and 195.47 seconds, respectively, with the latter achieving a 4.7x speedup. Notably, even on super-large graphs with billions of edges like Twitter, DSS++ consistently completes the (α, β)-dense subgraph search in approximately 200 seconds. These results confirm the high efficiency of the proposed DSS++ algorithm.

To demonstrate the effectiveness of core reduction in DSS++, we record the time taken for core reduction and the size of the reduced graph on five large datasets where $\alpha = \beta = \lfloor 0.5 \cdot p \rfloor$. The results are depicted in Table 2, and similar results can also be observed for other values of α and β. It is evident that the size of the graph significantly decreases after core reduction, falling to less than 20% of its original size. Notably, for the Twitter dataset with billions of edges, the reduced graph constitutes only 2.8% of the original. The time required for network flow computation of the (α, β)-dense subgraph search on these reduced graphs is minimal, with most of the time spent on calculating $C_{\alpha+1,\beta+1}$ and $C_{2\alpha+1,2\beta+1}$. These findings suggest that the computational cost of DSS++ equipped with the core reduction strategy for (α, β)-dense subgraph search is comparable to that of computing (α, β)-core subgraph.

Exp-2: Memory overheads of various dense subgraph search algorithms. The memory consumption of our dense subgraph search algorithms is shown in Figure 6. Note that it is unnecessary to vary α and β, since the memory overheads of various algorithms do not change with these parameters. As can be seen, the memory occupancy of the DSS, DSS+, and DSS++ algorithms is almost identical, as their space complexity is the graph size $O(|E|)$. For example, on Twitter dataset,



(a) IMDB, $\alpha = \beta = \lfloor k \cdot p \rfloor$ (b) Livejournal, $\alpha = \beta = \lfloor k \cdot p \rfloor$ (c) Yahoo, $\alpha = \beta = \lfloor k \cdot p \rfloor$ (d) Orkut, $\alpha = \beta = \lfloor k \cdot p \rfloor$ (e) Twitter, $\alpha = \beta = \lfloor k \cdot p \rfloor$

(f) IMDB, $\alpha = \lfloor k \cdot p \rfloor$, $\beta = \lfloor 0.5 \cdot p \rfloor$ (g) Livejournal, $\alpha = \lfloor k \cdot p \rfloor$, $\beta = \lfloor 0.5 \cdot p \rfloor$ (h) Yahoo, $\alpha = \lfloor k \cdot p \rfloor$, $\beta = \lfloor 0.5 \cdot p \rfloor$ (i) Orkut, $\alpha = \lfloor k \cdot p \rfloor$, $\beta = \lfloor 0.5 \cdot p \rfloor$ (j) Twitter, $\alpha = \lfloor k \cdot p \rfloor$, $\beta = \lfloor 0.5 \cdot p \rfloor$

(k) IMDB, $\alpha = \lfloor 0.5 \cdot p \rfloor$, $\beta = \lfloor k \cdot p \rfloor$ (l) Livejournal, $\alpha = \lfloor 0.5 \cdot p \rfloor$, $\beta = \lfloor k \cdot p \rfloor$ (m) Yahoo, $\alpha = \lfloor 0.5 \cdot p \rfloor$, $\beta = \lfloor k \cdot p \rfloor$ (n) Orkut, $\alpha = \lfloor 0.5 \cdot p \rfloor$, $\beta = \lfloor k \cdot p \rfloor$ (o) Twitter, $\alpha = \lfloor 0.5 \cdot p \rfloor$, $\beta = \lfloor k \cdot p \rfloor$

Fig. 5. Runtime of different (α, β)-dense subgraph search algorithms, varying α and β

Table 2. The effectiveness of core reduction in DSS++ (sec)
'C' represents the time consumption of core reduction, 'D' represents the network flow time after reduction, 'Sum' represents the total time consumption, and we set $\alpha = \beta = \lfloor 0.5 \cdot p \rfloor$.

Dataset	$\lvert E \rvert$	$\lvert E(R) \rvert$	ratio	C	D	Sum
IMDB	3.8M	714.3K	18.8%	1.8	0.2	2.0
Livejournal	112.3M	10.9M	9.7%	21.7	2.9	24.6
Yahoo	256.8M	17.1M	6.7%	15.4	3.7	19.1
Orkut	327.0M	27.3M	8.3%	66.3	8.3	74.6
Twitter	1.5B	42.0M	2.8%	110.7	69.6	180.3

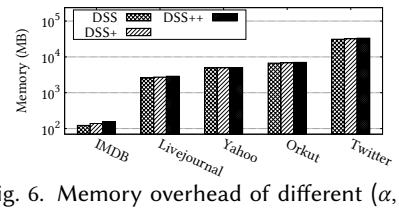
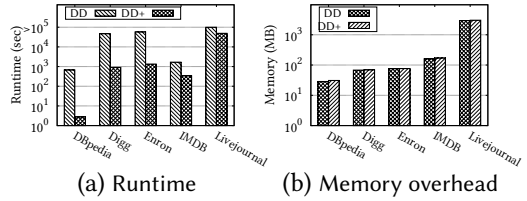


Fig. 6. Memory overhead of different (α, β)-dense subgraph search algorithms.

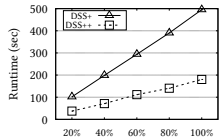


(a) Runtime (b) Memory overhead

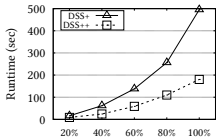
Fig. 7. Runtime and memory overheads of density decomposition algorithms.

DSS, DSS+, and DSS++ consume 31.2 GB, 32.2 GB, and 33.6 GB respectively, while the original bipartite graph takes 28.9 GB. These results demonstrate the memory efficiency of our proposed algorithms.

Exp-3: Evaluation of various denstiy decomposition algorithms. In this experiment, we compare the runtime and memory overhead of our proposed density decomposition algorithms, with the results depicted in Figure 7. As seen in Figure 6a, DD takes over 10^4 seconds on the Digg, Enron, and Livejournal datasets to compute the density decomposition, while DD+ consumes up to 244.5x less time than DD. For example, on the DBpedia dataset, DD takes 672.5 seconds to complete

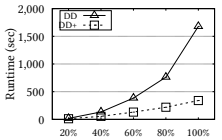


(a) Vary $|E|$

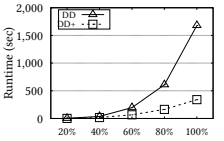


(b) Vary $|U \cup V|$

Fig. 8. Scalability test of the dense subgraph search algorithms on Twitter ($\alpha = \beta = \lfloor 0.5 \cdot p \rfloor$)



(a) Vary $|E|$



(b) Vary $|U \cup V|$

Fig. 9. Scalability test of the density decomposition algorithms on IMDB

the decomposition, while DD+ consumes 2.75 seconds, achieving a speedup of more than two orders of magnitude. These results demonstrate the high efficiency of DD+. For memory overheads, as shown in Figure 6b, DD and DD+ consume around the same amount of memory, indicating that our proposed algorithms are highly space efficient.

Exp-4: Scalability tests. We generate 8 subgraphs for each dataset by randomly sampling the node set $U \cup V$ and edge set E, and then perform our algorithms on these subgraphs. The running times of the dense subgraph search algorithms on Twitter and density decomposition algorithms on IMDB are shown in Figure 8 and Figure 9, respectively. The outcomes on the other datasets are consistent. For the dense subgraph search algorithms DSS+ and DSS++, DSS++ is consistently faster than DSS+ across all data scales, as expected. The runtime of DSS+ increases significantly with the data scale, while the runtime of DSS++ increases more smoothly. For the density decomposition algorithms, the runtime of DD+ increases more gently as the data size increases and is consistently lower than the runtime of DD. These results demonstrate the high scalability of the proposed DSS++ and DD+ algorithms.

Exp-5: Comparison of different models. In this experiment, we compare the runtime, memory consumption, density, and conductance of different models. Density (Definition 3) measures the internal structure tightness of the subgraph, and the conductance of a subgraph X, defined as $\frac{|E_{\times}(X,(U\cup V)\setminus X)|}{\sum_{x\in X} d_x}$, indicates the closeness of the subgraph to the vertices outside the subgraph. Intuitively, a cohesive subgraph should exhibit high density and low conductance. The models compared include our proposed dense subgraph (Definition 1), core subgraph [22], bitruss [35], biplex [15], and biclique [13]. For our dense subgraph, we use the proposed DSS++ algorithm to compute $D_{p,p}$, where p is the largest integer such that $D_{p,p} \neq \emptyset$. For the core subgraph, we compute $C_{\delta,\delta}$ using the state-of-the-art algorithm proposed in [17], where δ is the largest integer such that $C_{\delta,\delta} \neq \emptyset$. For bitruss, we implemented the state-of-the-art algorithm proposed in [35] to compute the k^*-bitruss, where k^* is the largest integer such that the k^*-bitruss is non-empty. For biplex and biclique, we use the codes from [15] and [13] to compute the maximum 1-biplex and biclique in the graph, respectively.

The results on the IMDB dataset are shown in Table 3 (similar results can be obtained on other datasets). For runtime, our dense subgraph and core subgraph have the shortest runtime, being 1-2 orders of magnitude faster than bitruss, biplex, and biclique. For memory usage, the algorithms for our proposed dense subgraph, core subgraph, and bitruss all have space complexity of the graph size $O(|E|)$, thus they consume less memory than biplex and biclique. For density and conductance, our proposed dense subgraph significantly outperforms the other models, demonstrating high effectiveness of our proposed model.

Specifically, the runtime of the core subgraph is only 0.05 seconds faster than that of the proposed dense subgraph, but its density and conductance are significantly worse than those of our proposed dense subgraph. Moreover, even if we do not use the maximum integer δ that makes $C_{\delta,\delta} \neq \emptyset$, and

Table 3. Comparison of different models on dataset IMDB.

Models	Runtime	Memory	Density	Conductance
dense subgraph [ours]	**0.20**	**155.7MB**	**41.487**	**0.493**
core subgraph [22]	**0.15**	**152.3MB**	33.955	0.552
bitruss [35]	31.59	**153.1MB**	34.346	0.516
biplex [15]	8.08	377.8MB	18.632	0.809
biclique [13]	21.2	801.1MB	16.492	0.838

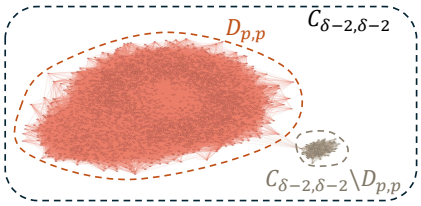


Fig. 10. $D_{p,p}$ and $C_{\delta-2,\delta-2}$ on dataset IMDB.

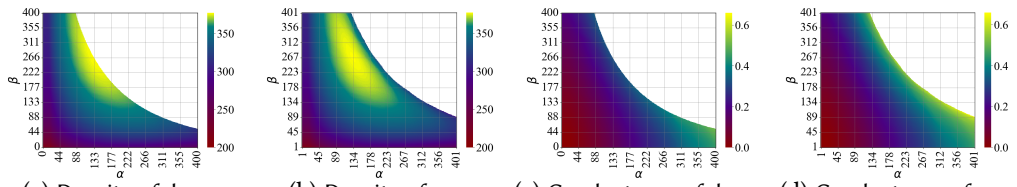
instead use $C_{\delta-1,\delta-1}$, $C_{\delta-2,\delta-2}$, or other core subgraph parameters, the density still cannot reach that of $D_{p,p}$. For example, as shown in Figure 10, $C_{\delta-2,\delta-2}$ consists of two parts: one is the denser $D_{p,p}$ with a density of 41.487, and the other is the sparser $C_{\delta-2,\delta-2} \setminus D_{p,p}$ with a density of 36.754. Besides, the subgraphs induced by $D_{p,p}$ and $C_{\delta-2,\delta-2} \setminus D_{p,p}$ contain 106,223 and 2,801 edges, respectively, but there are only $|E_{\times}(D_{p,p}, C_{\delta-2,\delta-2} \setminus D_{p,p})| = 15$ edges connecting these two parts. This suggests that $D_{p,p}$ and $C_{\delta-2,\delta-2} \setminus D_{p,p}$ belong to different communities, yet they are grouped together by $C_{\delta-2,\delta-2}$. A more reasonable partitioning method would be to discard the sparser $C_{\delta-2,\delta-2} \setminus D_{p,p}$ and retain only the denser $D_{p,p}$, as $D_{p,p}$ does. This demonstrates that the degree-based core subgraph model cannot partition subgraphs as accurately as our proposed density-based dense subgraph model.

In summary, our proposed dense subgraph consumes little runtime and memory while producing a high-density subgraph, indicating that our dense subgraph model is more well-suited for detecting densely-connected subgraphs compared with other models.

6.2 Case Studies

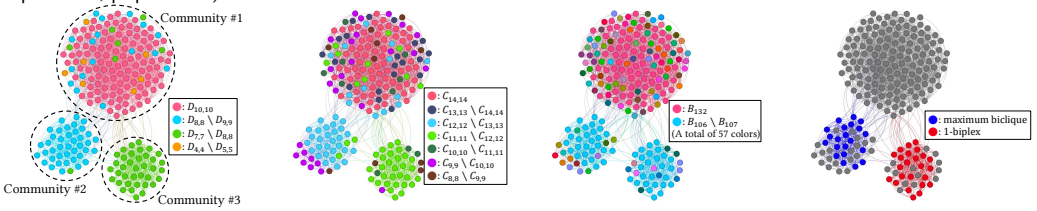
Comparison of (α, β)-dense subgraph and (α, β)-core models. We conduct a case study on the movie ratings dataset, Movielens, downloaded from https://grouplens.org/datasets/movielens/. This dataset contains $|U|$ = 6.0K users, $|V|$ = 4.0K movies, and $|E|$ = 1.0M edges representing user ratings of movies. We perform the density decomposition and core decomposition algorithms on Movielens and use two metrics: density and conductance to evaluate decomposed subgraphs.

The density (Definition 3) and conductance (definition in Exp-5) of (α, β)-dense subgraphs and (α, β)-cores are shown in Figure 11. As seen in Figure 11a and Figure 11c, with the increase of α and β, the density of $D_{\alpha,\beta}$ shows a monotonically increasing trend, which means that $D_{\alpha,\beta}$ as a density-based subgraph can well reflect the density structure of the graph. Additionally, the conductance of $D_{\alpha,\beta}$ remains low, suggesting that the connection between $D_{\alpha,\beta}$ and $(U \cup V) \setminus D_{\alpha,\beta}$ is not tight, highlighting that $D_{\alpha,\beta}$ is a relatively independent community. In contrast, as shown in Figure 11b and Figure 11d, when α and β are large, the density of $C_{\alpha,\beta}$ decreases drastically, resulting in $C_{\alpha,\beta}$ no longer being a densely-connected community. Moreover, its conductance increases dramatically, suggesting that its connection with $(U \cup V) \setminus C_{\alpha,\beta}$ is very tight and thus $C_{\alpha,\beta}$ might be a subgraph forcibly separated from a tightly-connected community. For instance, let p and δ be the largest integers such that $D_{p,p} \neq \emptyset$ and $C_{\delta,\delta} \neq \emptyset$, respectively. The densities of $D_{p,p}$ and $C_{\delta,\delta}$ are 371.8 and 341.1, and the conductances are 0.340 and 0.534 respectively. These results indicate that our (α, β)-dense subgraph model is denser and more *independent* compared



(a) Density of dense subgraphs (b) Density of core subgraphs (c) Conductance of dense subgraphs (d) Conductance of core subgraphs

Fig. 11. Density and conductance of dense subgraphs and core subgraphs on dataset Movielens ($|E| = 1.0M$, $|U| = 6.0K$, $|V| = 4.0K$).



(a) Our dense subgraph model. (b) The core subgraph model. (c) The k-bitruss subgraph model. (d) The biclique and the k-biplex model.

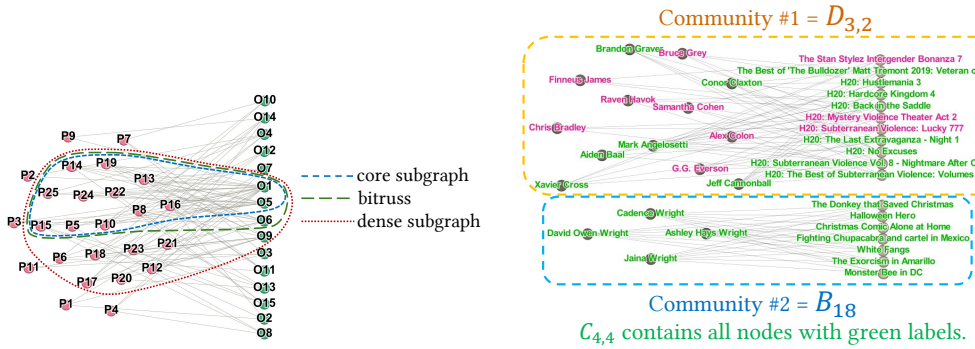
Fig. 12. Visualization on dataset Email.

to the (α, β)-core. The density decomposition is density-based and does not force the division of tightly-connected communities, whereas core decomposition is likely to do so. This result suggests that our density decomposition is superior to core decomposition for characterizing hierarchical dense subgraphs in bipartite graphs.

The hierarchy of different models on dataset Email**.** In this case study, we visualize the hierarchy of different models, i.e., we show how the proposed dense subgraph $D_{\alpha,\beta}$, core subgraph $C_{\alpha,\beta}$ [22], and k-bitruss B_k [35] change with their parameters α, β, and k, and we compare them with the maximum biclique [13] and k-biplex [15] model. We use the dataset Email[1] ($|U \cup V| = 220, |E| = 2094$), which represents email communication between users. This dataset contains three ground-truth communities, as shown in Figure 12a. The results of different models on this dataset are shown in Figure 12. Firstly, from Figure 12a, we can see that $D_{10,10}$, $D_{8,8} \setminus D_{9,9}$, and $D_{7,7} \setminus D_{8,8}$ approximately capture communities #1, #2, and #3, respectively. Specifically, $D_{10,10}$ contains $112/142 = 78.9\%$ of the nodes in community #1; $D_{8,8} \setminus D_{9,9}$ contains $40/41 = 97.6\%$ of the nodes in community #2; and $D_{7,7} \setminus D_{8,8}$ contains all the nodes of community #3. This indicates that the proposed dense subgraph can accurately partition the communities.

In contrast, from Figure 12b, we observe that the communities partitioned by the core subgraph are inaccurate. No matter how the parameters of the core subgraph are chosen (e.g., whether $C_{14,14}$, $C_{13,13}$, or any other $C_{\alpha,\beta}$), they cannot detect community #1 as accurately as the dense subgraph $D_{10,10}$. For example, $C_{14,14}$ contains only $69/142 = 48.6\%$ of the nodes in community #1, fewer than the $112/142 = 78.9\%$ nodes contained in $D_{10,10}$. Moreover, community #2 is split into two communities by $C_{12,12} \setminus C_{13,13}$ and $C_{9,9} \setminus C_{10,10}$, which is unreasonable. Therefore, the core subgraph fails to accurately partition the Email graph into three communities. For the k-bitruss, it divides the Email graph into 57 different bitruss subgraphs, which is too many and unnecessary. Additionally, B_{132} contains only $66/142 = 46.5\%$ of the nodes in community #1, also fewer than the $112/142 = 78.9\%$ nodes contained in $D_{10,10}$. This shows that bitruss cannot accurately partition the communities either. Regarding the maximum biclique and k-biplex, Figure 12d displays the results

[1]Data source: https://snap.stanford.edu/data/email-Eu-core.html, where we extract three ground-truth communities within this dataset.



(a) Results on the Club dataset. (b) Results on the Actor dataset.

Fig. 13. Visualization on dataset Club and Actor.

of a maximum biclique and 1-biplex in the Email dataset. It can be seen that both the maximum biclique and the 1-biplex are small and cannot detect large communities like community #1. This is because, although communities are relatively dense, they cannot satisfy the strict complete-graph-based definitions of biclique and 1-biplex. Therefore, the maximum biclique and k-biplex cannot accurately partition the communities of Email. In summary, the proposed dense subgraph provides the most accurate partitioning of the communities.

Visualization on dataset Club**.** We conduct a case study on the Club dataset (downloaded from http://www.konect.cc/networks/brunson_club-membership/), where the vertices in U represent persons, the vertices in V represent organizations, and an edge represents that a person belongs to an organization. We compute different dense subgraphs in Club including (α, β)-core subgraph $C_{\alpha,\beta}$, k-bitruss B_k and (α, β)-dense subgraph $D_{\alpha,\beta}$, and the results are shown in Figure 13a, where the core subgraph is $C_{2,11}$, the bitruss is the 10-bitruss (denoted by B_{10}), and the dense subgraph is $D_{1,6}$. These parameters are chosen to be the maximum values, meaning that there are no larger α,β,k making $C_{\alpha,\beta}$, $D_{\alpha,\beta}$, and B_k non-empty. As shown, $D_{1,6}$ contains a densely-connected community, while $C_{2,11}$ and B_{10} are subgraphs forcibly separated from this community. This is also evidenced by their density and conductance. The densities of $D_{1,6}$, $C_{2,11}$, and B_{10} are 5.3, 4.7, and 4.9 respectively, and their conductance are 0.297, 0.443, and 0.364 respectively. $D_{1,6}$ has the highest density and the lowest conductance, indicating that it is a densely-connected community and is loosely connected to the outside. This result further confirms the superior effectiveness of our model in representing communities in bipartite graphs, compared to core-based or bitruss-based models.

Visualization on dataset Actor**.** We also conduct a case study on the Actor dataset (downloaded from https://docs.conscia.ai/tutorials/example-movie-dataset), where the vertices in U represent actors, the vertices in V represent movies, and an edge represents that an actor participated in a movie. The results are shown in Figure 13b. As can be seen, the dense subgraph $D_{3,2}$ is a densely-connected community #1, containing multiple movies from the 'H20' series and their leading actors. Bitruss B_{18} is a smaller community #2. Comparatively, the densities of community #1 and community #2 are 5.352 and 5.292, respectively, so community #1 in $D_{3,2}$ is denser and larger than community #2 in B_{18}. For the core subgraph, its $C_{4,4}$ forcibly splits community #1 and merges it with community #2, which clearly is an unreasonable community division. Once again, this result further demonstrates that the (α, β)-dense subgraph is density-based with the desirable properties, i.e., tightly connected internally and sparsely connected to the outside.

7 Related Works

Cohesive subgraph model in bipartite graphs. In the literature, there are various types of cohesive models for bipartite graphs. A fundamental model is the biclique [1, 20, 23, 26, 27, 36, 41], which is a complete bipartite subgraph but is often too strict for real-world networks. Relaxed models like the k-biplex [15, 37, 38], and quasi-biclique [21, 24, 31] allow for controlled incompleteness, making them more practical. However, computing bicliques or these relaxed bicliques is NP-hard, implying that no polynomial-time algorithm exists unless P=NP. Besides clique-based models, there are also butterfly-based models like the k-bitruss [29, 30, 35, 42]. This model decomposes graphs using butterfly structures ($(2,2)$-complete graph) to measure local density. The degree-based (α,β)-core [12, 17, 19, 22, 25], despite being computationally efficient, often fails to reflect the density structure as it relies on degree constraints. This paper introduces the (α,β)-dense subgraph, a novel density-based model that can accurately capture density structure. It is computationally efficient and suitable for large bipartite graphs.

Dense subgraph model in unipartite graphs. In unipartite graphs, Tatti [32] proposed the locally-dense decomposition based on the densest subgraph, which can identify all locally-dense subgraphs in the graph and designed an algorithm with a time complexity of $O(|V|^2|E|)$. Later, Danisch et al. [16] designed a convex optimization algorithm to accelerate the computation of locally-dense decomposition. Besides locally-dense decomposition, Qin et al. [28] introduced the concept of locally-densest subgraph, which can identify the densest subgraph in different regions of the graph. Subsequently, Trung et al. [33] proposed a verification-free method to accelerate the computation of locally-densest subgraphs. Density decomposition, which we base on in this paper, was initially proposed by Borradaile et al. [10], who presented an algorithm with a time complexity of $O(|E|^2)$ to compute the density decomposition. All the above decomposition models are for unipartite graphs. To the best of our knowledge, there is no density-based decomposition for bipartite graphs. Our work is the first to propose density decomposition for bipartite graphs.

8 Conclusion

In this paper, we propose a novel dense subgraph model on bipartite graphs called (α,β)-dense subgraph. We show that all the (α,β)-dense subgraphs form a hierarchical decomposition of a bipartite graph. We theoretically explore the relationship between (α,β)-dense subgraphs and core subgraphs, proposing the Sandwich Theorem and demonstrating the theoretical advantages of our dense subgraphs over the traditional core subgraphs. Then, we develop three new algorithms to compute the (α,β)-dense subgraph. Among them, the fastest DSS++ algorithm utilizes a nontrivial network flow method and a carefully-designed core pruning technique, reducing the time complexity to $O(|E|+|E(R)|^{1.5})$. Subsequently, we propose two algorithms for computing density decomposition. The DD+ algorithm employs a novel divide-and-conquer strategy to iteratively reduce the graph size, bringing the time complexity down to $O(p \cdot \log d_{\max} \cdot |E|^{1.5})$. Extensive experiments and case studies demonstrate the effectiveness of the (α,β)-dense subgraph model and the high efficiency and scalability of the proposed algorithms. There are several future directions deserving further investigation. First, in the case of dynamic bipartite graphs, an interesting problem would be to maintain the density decomposition in dynamic graphs. Second, it would be valuable to explore the parallel or distributed algorithms for computing the density decomposition of bipartite graphs.

Acknowledgments

This work was supported by NSFC Grants U2241211. Rong-Hua Li is the corresponding author of this paper.

References

[1] Gabriela Alexe, Sorin Alexe, Yves Crama, Stephan Foldes, Peter L. Hammer, and Bruno Simeone. 2004. Consensus algorithms for the generation of all maximal bicliques. *Discret. Appl. Math.* 145, 1 (2004), 11–21.

[2] Mohammad Allahbakhsh, Aleksandar Ignjatovic, Boualem Benatallah, Amin Beheshti, Elisa Bertino, and Norman Foo. 2013. Collusion Detection in Online Rating Systems. In *APWeb (Lecture Notes in Computer Science, Vol. 7808)*. 196–207.

[3] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawla, Gautam Das, and Cong Yu. 2009. Group Recommendation: Semantics and Efficiency. *Proc. VLDB Endow.* 2, 1 (2009), 754–765.

[4] Reid Andersen. 2010. A local algorithm for finding dense subgraphs. *ACM Trans. Algorithms* 6, 4 (2010), 60:1–60:12.

[5] Jon Louis Bentley, Dorothea Haken, and James B. Saxe. 1980. A general method for solving divide-and-conquer recurrences. *SIGACT News* 12, 3 (1980), 36–44.

[6] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. CopyCatch: stopping group attacks by spotting lockstep behavior in social networks. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*. International World Wide Web Conferences Steering Committee / ACM, 119–130.

[7] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. CopyCatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*. 119–130.

[8] Ivona Bezáková. 2000. Compact representations of graphs and adjacency testing. (2000).

[9] Markus Blumenstock. 2016. Fast Algorithms for Pseudoarboricity. In *ALENEX*. 113–126.

[10] Glencora Borradaile, Theresa Migler, and Gordon T. Wilfong. 2019. Density decompositions of networks. *J. Graph Algorithms Appl.* 23, 4 (2019), 625–651.

[11] Lucas Augusto Montalvão Costa Carvalho and Hendrik Teixeira Macedo. 2013. Users' satisfaction in recommendation systems for groups: an approach based on noncooperative games. In *WWW*. 951–958.

[12] Monika Cerinsek and Vladimir Batagelj. 2015. Generalized two-mode cores. *Soc. Networks* 42 (2015), 80–87.

[13] Jiujian Chen, Kai Wang, Ronghua Li, Hongchao Qin, Xuemin Lin, and Guoren Wang. 2024. Maximal Biclique Enumeration: A Prefix Tree Based Approach. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 2544–2556.

[14] Francesco Colace, Massimo De Santo, Luca Greco, Vincenzo Moscato, and Antonio Picariello. 2015. A collaborative user-centered framework for recommending items in Online Social Networks. *Comput. Hum. Behav.* 51 (2015), 694–704.

[15] Qiangqiang Dai, Rong-Hua Li, Donghang Cui, Meihao Liao, Yu-Xuan Qiu, and Guoren Wang. 2024. Efficient Maximal Biplex Enumerations with Improved Worst-Case Time Guarantee. *Proc. ACM Manag. Data* 2, 3 (2024), 135.

[16] Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. 2017. Large Scale Density-friendly Graph Decomposition via Convex Programming. In *WWW*. 233–242.

[17] Danhao Ding, Hui Li, Zhipeng Huang, and Nikos Mamoulis. 2017. Efficient Fault-Tolerant Group Recommendation Using alpha-beta-core. In *CIKM*. 2047–2050.

[18] Jagadeesh Gorla, Neal Lathia, Stephen Robertson, and Jun Wang. 2013. Probabilistic group recommendation via information matching. In *WWW*. 495–504.

[19] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Exploring cohesive subgraphs with vertex engagement and tie strength in bipartite graphs. *Inf. Sci.* 572 (2021), 277–296.

[20] Dorit S. Hochbaum. 1998. Approximating Clique and Biclique Problems. *J. Algorithms* 29, 1 (1998), 174–200.

[21] Dmitry I. Ignatov. 2019. Preliminary Results on Mixed Integer Programming for Searching Maximum Quasi-Bicliques and Large Dense Biclusters. In *ICFCA (CEUR Workshop Proceedings, Vol. 2378)*. 28–32.

[22] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (a,β)-core Computation: an Index-based Approach. In *WWW*. 1130–1141.

[23] Guimei Liu, Kelvin Sim, and Jinyan Li. 2006. Efficient Mining of Large Maximal Bicliques. In *DaWaK (Lecture Notes in Computer Science, Vol. 4081)*. 437–448.

[24] Xiaowen Liu, Jinyan Li, and Lusheng Wang. 2008. Quasi-bicliques: Complexity and Binding Pairs. In *COCOON (Lecture Notes in Computer Science, Vol. 5092)*. 255–264.

[25] Wensheng Luo, Qiaoyuan Yang, Yixiang Fang, and Xu Zhou. 2023. Efficient Core Maintenance in Large Bipartite Graphs. *Proc. ACM Manag. Data* 1, 3 (2023), 208:1–208:26.

[26] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum Biclique Search at Billion Scale. *Proc. VLDB Endow.* 13, 9 (2020), 1359–1372.

[27] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos E. Tsourakakis, and Shen Chen Xu. 2015. Scalable Large Near-Clique Detection in Large-Scale Networks via Sampling. In *KDD*. 815–824.

[28] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally Densest Subgraph Discovery. In *KDD*. 965–974.

[29] Ahmet Erdem Sariyüce and Ali Pinar. 2018. Peeling Bipartite Networks for Dense Subgraph Discovery. In *WSDM*, Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek (Eds.). 504–512.

[30] Jessica Shi and Julian Shun. 2022. Parallel Algorithms for Butterfly Computations. In *Massive Graph Analytics*, David A. Bader (Ed.). Chapman and Hall/CRC, 287–330.
[31] Kelvin Sim, Jinyan Li, Vivekanand Gopalkrishnan, and Guimei Liu. 2009. Mining maximal quasi-bicliques: Novel algorithm and applications in the stock market and protein networks. *Stat. Anal. Data Min.* 2, 4 (2009), 255–273.
[32] Nikolaj Tatti. 2019. Density-Friendly Graph Decomposition. *ACM Trans. Knowl. Discov. Data* 13, 5 (2019), 54:1–54:29.
[33] Tran Ba Trung, Lijun Chang, Nguyen Tien Long, Kai Yao, and Huynh Thi Thanh Binh. 2023. Verification-Free Approaches to Efficient Locally Densest Subgraph Discovery. In *ICDE*. 1–13.
[34] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR*. 501–508.
[35] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2022. Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs. *VLDB J.* 31, 2 (2022), 203–226.
[36] Jianye Yang, Yun Peng, Dian Ouyang, Wenjie Zhang, Xuemin Lin, and Xiang Zhao. 2023. (p,q)-biclique counting and enumeration for large sparse bipartite graphs. *VLDB J.* 32, 5 (2023), 1137–1161.
[37] Kaiqiang Yu, Cheng Long, Shengxin Liu, and Da Yan. 2022. Efficient Algorithms for Maximal k-Biplex Enumeration. In *SIGMOD*. 860–873.
[38] Kaiqiang Yu, Cheng Long, Deepak P, and Tanmoy Chakraborty. 2023. On Efficient Large Maximal Biplex Discovery. *IEEE Trans. Knowl. Data Eng.* 35, 1 (2023), 824–829.
[39] Quan Yuan, Gao Cong, and Chin-Yew Lin. 2014. COM: a generative model for group recommendation. In *KDD*. 163–172.
[40] Yun Zhang, Charles A. Phillips, Gary L. Rogers, Erich J. Baker, Elissa J. Chesler, and Michael A. Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinform.* 15 (2014), 110.
[41] Yun Zhang, Charles A. Phillips, Gary L. Rogers, Erich J. Baker, Elissa J. Chesler, and Michael A. Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinform.* 15 (2014), 110.
[42] Zhaonian Zou. 2016. Bitruss Decomposition of Bipartite Graphs. In *DASFAA (Lecture Notes in Computer Science, Vol. 9643)*. 218–233.

Received July 2024; revised September 2024; accepted November 2024