Integral Densest Subgraph Search on Directed Graphs

ABSTRACT

The densest subgraph (DS) search over a directed graph focuses on finding the subgraph with the highest density among all subgraphs. This problem has raised numerous applications, such as fraud detection and community detection. The state-of-the-art DS algorithms have prohibitively high costs or poor approximation ratios, making them unsuitable for practical applications. To address these dilemmas, in this paper, we propose a novel model called integral densest subgraph (IDS). We show that IDS can serve as a near-DS model that has a tight floor relationship with the density of the DS. To compute IDS, we first propose a novel flow network named (α, β)-dense network, based on which we design an exact network-flow algorithm GetIDS with $O(p \cdot \log |V| \cdot |E|^{1.5})$ time complexity, where p is typically a small constant in real-world graphs. Additionally, we propose several non-trivial pruning techniques to further improve the efficiency. Subsequently, we propose a novel $(2+\epsilon)$-approximation algorithm MultiCore with near-linear time complexity, providing a good approximation guarantee with high efficiency. Finally, our extensive experiments on 10 real-world graphs demonstrate the effectiveness of the proposed IDS model, and the high efficiency and scalability of the proposed solutions.

1 INTRODUCTION

The directed graph is a ubiquitous data model that captures the directional nature of relationships [1, 8, 22]. For example, in a social media platform, a directed graph can model the "follow" relationships between users, where an edge from user A to user B indicates that A follows B [23]. In biology networks, directed graphs are essential for representing gene regulatory relationships, where nodes represent genes and directed edges indicate regulatory influences [17, 26, 35]. Therefore, numerous directed graph algorithms have been proposed in the literature [3, 10, 19, 24, 28, 30, 31, 36]. Among them, identifying the densest subgraph from a directed graph has gained wide attention due to its nice structure properties and solid theoretical foundation [3, 10, 24, 30, 31]. Besides, it has also enjoyed numerous applications, such as fraud detection [21], e-commerce recommendation [13], and community detection [27].

Given a directed graph $G = (V, E)$ and two node subsets $S, T \subseteq V$, the density of the subgraph induced by (S, T) is defined as $\rho(S, T) = \frac{|E(S,T)|}{\sqrt{|S|\cdot|T|}}$, where $E(S, T)$ denotes the set of directed edges from vertices in S to vertices in T. The densest subgraph (DS) problem aims to find a pair (S^*, T^*) with the maximum density [3, 10, 24, 30, 31].

To solve the DS problem, many exact [24, 30, 31] and approximation algorithms [3, 10, 24] have been introduced in the literature. The state-of-the-art (SOTA) exact algorithm is CP-Exact [30], which maps the DS problem into a set of linear programs. However, since CP-Exact has to evaluate all possible values of $|S^*|/|T^*|$, it may need to invoke $O(|V|^2)$ convex-programming algorithms (e.g., Frank-Wolfe [30]) and maximum flow computations in the worst case, resulting in prohibitively high time complexity. The authors [30] also proposed the approximation algorithm CP-Approx to accelerate CP-Exact. Unfortunately, CP-Approx remains time-consuming while offering a poor practical approximation ratio, as indicated in our experiments (Section 7). Therefore, designing efficient algorithms to compute either an exact solution or a high-quality approximation of the DS problem in directed graphs is very challenging for massive graphs with billions of edges.

To this end, in this paper, we propose a novel model called *integral densest subgraph* (IDS), which is not only computationally efficient but also achieves near-optimal density. Specifically, we first present the novel concepts of fractional (α, β)-dense subgraph $F_{\alpha,\beta}$ and integral (α, β)-dense subgraph $D_{\alpha,\beta}$. These subgraphs can achieve different densities based on the parameters α and β. Subsequently, we prove that the non-empty $F_{\alpha,\beta}$ that maximizes the product $\alpha \cdot \beta$ is exactly equal to the DS (Theorem 4). Based on these theoretical innovations, we define the IDS as the $D_{\alpha,\beta}$ that maximizes $\alpha \cdot \beta$. On top of that, let DS be F_{α^*,β^*}, we prove that its density equals $2\sqrt{\alpha^*\beta^*}$, and the density of our IDS is at least $2\sqrt{\lfloor\alpha^*\rfloor\lfloor\beta^*\rfloor}$ (Theorem 6). Thus, the IDS provides a near-optimal density guarantee. For example, on all datasets in our experiments, the density of the IDS is at least 0.9994 times that of the DS (Table 4 in Section 7). This demonstrates that IDS is a near-DS model and can achieve results comparable to DS.

To compute IDS, we propose a novel flow network called (α, β)-dense network to compute a $D_{\alpha,\beta}$ subgraph given α and β. To search for the non-empty $D_{\alpha,\beta}$ that maximizes $\alpha \cdot \beta$, we develop a parameterized network-flow algorithm GetIDS, which leverages binary search framework to compute $D_{\alpha,\beta}$ with large $\alpha \cdot \beta$ values and selects the one that maximizes $\alpha \cdot \beta$ as the IDS. Furthermore, by analyzing the properties of $D_{\alpha,\beta}$, we propose the *emptiness* theorem (Theorem 10) and the *non-emptiness* theorem (Theorem 11). Based on these theorems, we design several non-trivial pruning techniques and propose a more efficient algorithm GetIDS++ based on a carefully-designed divide-and-conquer technique, which has a time complexity of $O(p \cdot \log |V| \cdot |E|^{1.5})$, where p is typically a small constant in real-world graphs (Table 2). This complexity is significantly lower than the time complexity $O(|V|^2 t_{FW})$ of the SOTA algorithm CP-Exact for computing DS (t_{FW} is the time for convex-programming and maximum flow computations), making the computation of IDS much more efficient than that of DS.

To further improve the efficiency, we also propose approximation algorithms for solving our IDS problem. We first observe that although existing approximation algorithms for computing

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2025 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

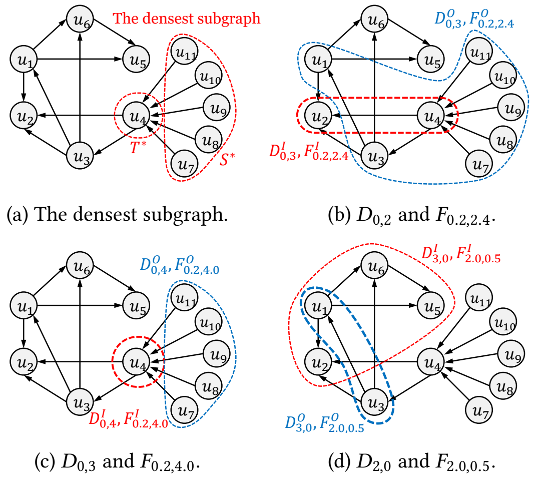
Table 1: Summary of exact and approximation algorithms. t_{FW} is the time for convex-programming and maximum flow computations; n_f is the number of flow computations and $n_f \ll p \cdot \log |V|$; p is often a small constant and typically $p \ll \sqrt{|E|}$ in real graphs; c^* is the ratio $c^* = |S^*|/|T^*|$ of the densest subgraph $G[S^*, T^*]$.

Exact algorithms

Algorithms	Time complexity
CP-Exact [30]	$O(\|V\|^2 t_{FW})$
GetIDS [ours]	$O(p \cdot \log \|V\| \cdot \|E\|^{1.5})$
GetIDS++ [ours]	$O(n_f \cdot \|E\|^{1.5})$

Approximation algorithms

Algorithms	Approximation ratio	Time complexity
SingleCore [25]	$\sqrt{c^*} + 1/\sqrt{c^*}$	$O(\|E\|)$
AllCore [31]	2	$O(\|E\|^{1.5})$
CP-Approx [30]	$1 + \epsilon$	$O(\log_{1+\epsilon} \|V\| \cdot t_{FW})$
MultiCore [ours]	$2 + \epsilon$	$O(\log_{1+\epsilon} \|V\| \cdot \|E\|)$


The densest subgraph
u_1
u_2
u_3
u_4
u_5
u_6
u_7
u_8
u_9
u_{10}
u_{11}
T^*
S^*
$D^O_{0,3}, F^O_{0.2,2.4}$
$D^I_{0,3}, F^I_{0.2,2.4}$
$D^O_{0,4}, F^O_{0.2,4.0}$
$D^I_{0,4}, F^I_{0.2,4.0}$
$D^I_{3,0}, F^I_{2.0,0.5}$
$D^O_{3,0}, F^O_{2.0,0.5}$

(a) The densest subgraph. (b) $D_{0,2}$ and $F_{0.2,2.4}$. (c) $D_{0,3}$ and $F_{0.2,4.0}$. (d) $D_{2,0}$ and $F_{2.0,0.5}$.

Figure 1: Running example.

DS [25, 30, 31] can also be used to approximate our IDS, they either provide poor approximation quality [25] or are very time-consuming [30, 31], making them unsuitable for practical applications. Inspired by the existing SingleCore algorithm [25], we design a novel approximation algorithm MultiCore with near-linear time complexity. Specifically, we first use the proposed non-emptiness theorem to derive an approximation ratio of $(\sqrt{c^*} + \frac{1}{\sqrt{c^*}})$ for the SingleCore algorithm, where $c^* = |S^*|/|T^*|$. Then, we propose a novel $(2 + \epsilon)$-approximation algorithm, MultiCore, which extends the rationale of performing a single peeling operation in SingleCore to $O(\log_{1+\epsilon} |V|)$ peeling operations, thereby improving the approximation guarantee to $(2 + \epsilon)$. MultiCore only requires $O(|E| \cdot \log_{1+\epsilon} |V|)$ time, enabling it to achieve both a good approximation guarantee and fast computation speed. The main contributions of this paper are summarized below:

Novel integral densest subgraph model. We first introduce two novel concepts of fractional (α, β)-dense subgraph $F_{\alpha,\beta}$ and integral (α, β)-dense subgraph $D_{\alpha,\beta}$. We then prove that the non-empty $F_{\alpha,\beta}$ that maximizes $\alpha \cdot \beta$ is equal to the DS. Based on these theoretical innovations, we define the novel IDS model as the non-empty $D_{\alpha,\beta}$ that maximizes $\alpha \cdot \beta$. A striking feature is that our proposed IDS provides a near-optimal density guarantee, which has a tight floor relationship with the density of the DS (details in Section 3.2).

Efficient exact algorithms. To compute the IDS, we first develop a novel network-flow algorithm GetIDS. To enhance the efficiency, we propose several non-trivial pruning techniques to eliminate unnecessary flow network computations and further design a more efficient algorithm GetIDS++. Its time complexity is $O(p \cdot \log |V| \cdot |E|^{1.5})$, where p is often a small constant in real-world graphs. This time complexity is significantly better than the $O(|V|^2 t_{FW})$ time complexity for computing the DS. Thus, our IDS model offers a highly efficient computational approach.

Accurate and efficient approximation algorithms. To further boost efficiency, we propose a novel $(2+\epsilon)$-approximation algorithm MultiCore, based on a carefully-designed peeling technique. This technique allows us to achieve a $(2 + \epsilon)$-approximation with only a small number of $O(\log_{1+\epsilon} |V|)$ peeling operations. As a result, our MultiCore has a near-linear time complexity of $O(|E| \cdot \log_{1+\epsilon} |V|)$. Table 1 summarizes the time complexities of the algorithms and the approximation factors for the approximation algorithms.

Extensive experiments. We conducted experiments on 10 real-world graphs, and the results show that: (1) the improved GetIDS++ algorithm is up to two orders of magnitude faster than the basic GetIDS algorithm and up to four orders of magnitude faster than the SOTA algorithm for computing the DS; (2) our $(2+\epsilon)$-approximation algorithm MultiCore, with $\epsilon = 0.5$, achieves an actual approximation ratio below 1.05 on all datasets, and below 1.0005 on 5 out of 10 datasets, while its runtime is significantly lower than other algorithms with comparable approximation quality; (3) our case study evaluate the performance of the IDS and the DS in practical fraudulent detection application. The results show that the IDS detects more fraudulent users while being 126 times faster than the DS, demonstrating the high effectiveness of our IDS model.

Reproducibility and full-version paper. The source code of this paper and the full-version of this paper can be found at https://anonymous.4open.science/r/maxab-C4E8/.

2 PRELIMINARIES

Let $G = (V, E)$ be a directed graph, where V is the set of vertices and E is the set of directed edges. For any vertex $x \in V$, its outdegree and indegree are denoted by $d^O_x(G)$ and $d^I_x(G)$, respectively, or d^O_x and d^I_x for brevity. Given two subsets $S, T \subseteq V$ (not necessarily disjoint), the set $E(S, T)$ denotes all directed edges starting from vertices in S and ending at vertices in T, i.e., $E(S, T) = \{(x, y) \in E | x \in S,\ y \in T\}$. The subgraph $G[S, T] = (S \cup T, E(S, T))$ is called the (S, T)-induced subgraph. Below, we give the definitions of density and the densest subgraph.

Definition 1. **(Density)** *Given a graph G and its (S, T)-induced subgraph $G[S, T]$, the density of $G[S, T]$ is defined as*

$$\rho(S, T) = \frac{|E(S, T)|}{\sqrt{|S| \cdot |T|}}. \quad (1)$$

Definition 2. **(Densest subgraph, DS)** *Given a graph G, the DS of G is the maximal subgraph $G[S^*, T^*]$ with the largest density $\rho(S^*, T^*)$, where the maximal property means there does not exist a subgraph $G[S^+, T^+] \supset G[S^*, T^*]$ with $\rho(S^+, T^+) = \rho(S^*, T^*) = \rho^*$. The density $\rho(S^*, T^*)$ is also denoted by ρ^* for simplicity.*

Unlike in undirected graphs where the DS is induced by a single set of vertices, in directed graphs the DS is induced by two sets of vertices. This is because the definition considers the directionality in directed graphs, thereby fitting practical applications better. Taking the application of fraud user detection as an example, suppose the directed graph in Figure 1a is a social network where directed edges represent the "follow" relationships between users. Then user u_4 might have potentially hired five fraudulent robot followers $u_7, \ldots, u_{11}$ to increase its follower count. The DS in this directed graph is the (S^*, T^*)-induced subgraph, indicating that fraudulent users can be detected by the DS. However, if we ignore the directionality of edges in the directed graph and use the densest subgraph model of undirected graph to find fraudulent users, the result would be $\{u_1, \ldots, u_6\}$, which is an incorrect result. This demonstrates that in directed graphs, defining the DS as an induced subgraph of two different sets of vertices is necessary and meaningful.

Next, we define a novel concept called the fractional (α, β)-dense subgraph.

Definition 3. **(Fractional (α, β)-dense subgraph $F_{\alpha,\beta}$)** *Given a graph G and two non-negative* ***real numbers*** *α and β, the (α, β)-dense subgraph $F_{\alpha,\beta}$ is defined as the subgraph $G[F^O_{\alpha,\beta}, F^I_{\alpha,\beta}]$, satisfying: (1)* ***(internally dense)*** *for any $S \subseteq F^O_{\alpha,\beta}$, $T \subseteq F^I_{\alpha,\beta}$, and $S \cup T \neq \emptyset$, $F_{\alpha,\beta}$ has $|E(F^O_{\alpha,\beta}, F^I_{\alpha,\beta})| - |E(F^O_{\alpha,\beta} \setminus S, F^I_{\alpha,\beta} \setminus T)| \geq \alpha \cdot |S| + \beta \cdot |T|$; (2)* ***(externally sparse)*** *for any $S \cap F^O_{\alpha,\beta} = \emptyset$, $T \cap F^I_{\alpha,\beta} = \emptyset$, and $S \cup T \neq \emptyset$, $F_{\alpha,\beta}$ satisfies $|E(F^O_{\alpha,\beta} \cup S, F^I_{\alpha,\beta} \cup T)| - |E(F^O_{\alpha,\beta}, F^I_{\alpha,\beta})| < \alpha \cdot |S| + \beta \cdot |T|$.*

According to the condition (1), the internal structure of the subgraph $F_{\alpha,\beta}$ is relatively dense because removing any (S, T) within it will lose at least $\alpha \cdot |S| + \beta \cdot |T|$ edges. Similarly, condition (2) suggests that the area outside the subgraph $F_{\alpha,\beta}$ is relatively sparse because adding any (S, T) outside of it into $F_{\alpha,\beta}$ can add at most $\alpha \cdot |S| + \beta \cdot |T|$ edges. Therefore, $F_{\alpha,\beta}$ represents a subgraph that is dense internally and sparse externally.

Based on Definition 3, we define the *integral (α, β)-dense subgraph* by restricting α and β to be integers. Since $D_{\alpha,\beta}$ is defined based on $F_{\alpha,\beta}$, $D_{\alpha,\beta}$ also inherits the desirable property of being dense internally and sparse externally.

Definition 4. **(Integral (α, β)-dense subgraph $D_{\alpha,\beta}$)** *Given a graph G and two non-negative* ***integers*** *α and β, the integral (α, β)-dense subgraph $D_{\alpha,\beta}$ is the corresponding fractional subgraph $F_{\alpha,\beta}$.*

By $F_{\alpha,\beta}$ (Definitions 3) and $D_{\alpha,\beta}$ (Definitions 4), we can find a subgraph with a specific density by varying α and β. The larger α and β are, the denser the subgraph tends to be. The two-dimensional parameters (α, β) can respectively control the densities of the two parts of the obtained subgraph, $(F^O_{\alpha,\beta}, F^I_{\alpha,\beta})$ or $(D^O_{\alpha,\beta}, D^I_{\alpha,\beta})$.

Remark: (α, β)-dense subgraph and (α, β)-core. In the literature, a subgraph model similar to the (α, β)-dense subgraph is the (α, β)-core (Definition 6). Both models adjust subgraph density by varying the parameters α and β. However, the (α, β)-core model is based on degree rather than density, which often leads to inaccurate reflection of the density structure of graphs. Specifically, given a graph, the maximum density ρ_C among all (α, β)-core subgraphs can only guarantee $\rho_C \geq \rho^*/2$ and cannot ensure a near-optimal density (such as the near-optimal guarantee of IDS). Therefore, our density-based dense subgraph model provides better density guarantees compared to the (α, β)-core, enabling the discovery of denser (i.e., better) subgraphs in practical applications.

Example 1. *Take the directed graph in Figure 1a as an example and assume it is a social network where directed edges represent the following relationships between users. Intuitively, when α increases, users in $F^O_{\alpha,\beta}$ and $D^O_{\alpha,\beta}$ follow more people; when β increases, users in $F^I_{\alpha,\beta}$ and $D^I_{\alpha,\beta}$ have more followers. For example, when α is set low and β is set high, comparing the subgraphs in Figure 1b and Figure 1c, $D^I_{0,3}$ and $F^I_{0.2,2.4}$ include users u_2 and u_4, while $D^I_{0,4}$ and $F^I_{0.2,4.0}$ include only u_4, which has more followers than u_2. This indicates that increasing β can filter out users with more followers. Symmetrically, in Figure 1d, when α is set high and β is set low, we can filter out users who follow many users, u_1 and u_3, in D^O and F^O.*

Below, we prove both $D_{\alpha,\beta}$ and $F_{\alpha,\beta}$ exist and are unique.

Theorem 1. *Given a graph G and non-negative real number (resp., integers) α, β, we have $F_{\alpha,\beta}$ (resp., $D_{\alpha,\beta}$) exists and is unique.*

Proof. Let $f(S, T) = |E(S, T)| - \alpha \cdot |S| - \beta \cdot |T|$, and F^O and F^I be the inputs that maximize f. Assume that (F^O, F^I) is maximal, meaning there does not exist a subgraph $G[S, T] \supset G[F^O, F^I]$ such that $f(S, T) = f(F^O, F^I)$. According to the definition of $F_{\alpha,\beta}$, if $G[F^O, F^I]$ does not satisfy the condition in its definition, we could either remove S and T (condition (1)) or add S and T (condition (2)) to obtain a subgraph with a higher f value. This contradicts the assumption that $G[F^O, F^I]$ maximizes f, and thus, $G[F^O, F^I]$ satisfies both conditions. Thus, $G[F^O, F^I]$ is $F_{\alpha,\beta}$ and is unique. Analogously, we can also prove that $D_{\alpha,\beta}$ exists and is unique. □

In Section 3, we will prove that if two real numbers α^* and β^* satisfy $F_{\alpha^*,\beta^*} \neq \emptyset$ and maximize the product $\alpha^* \cdot \beta^*$, then F_{α^*,β^*} is exactly the DS. This demonstrates that we can find the densest subgraph (DS) by maximizing $\alpha \cdot \beta$ in our fractional (α, β)-dense subgraph model. Based on this intuition, we define the IDS by restricting α and β to be integers.

Definition 5. **(Integral densest subgraph, IDS)** *Given a graph G, let two integers $\overline{\alpha}^*$ and $\overline{\beta}^*$ satisfy $D_{\overline{\alpha}^*,\overline{\beta}^*} \neq \emptyset$ and maximize the product $\overline{\alpha}^* \cdot \overline{\beta}^*$. Then, $D_{\overline{\alpha}^*,\overline{\beta}^*}$ is defined as the IDS.*

In this paper, we use the symbols α^* and β^* (resp., $\overline{\alpha}^*$ and $\overline{\beta}^*$) to denote the α and β values of the DS (resp., IDS).

According to Theorem 4 given in Section 3, DS is equivalent to F_{α^*,β^*} that maximizes $\alpha^* \cdot \beta^*$. Therefore, both DS and IDS can be viewed as results of maximizing the product of α and β. However, there is no inherent relationship between them. Specifically, the DS is not guaranteed to be contained within the IDS, because DS optimizes over real-valued solutions while IDS optimizes over integer-valued solutions. Therefore, existing solutions for computing the DS cannot be directly applied to compute the IDS, and novel approaches are required to compute the IDS.

3 RELATION WITH DENSEST SUBGRAPH

3.1 F_{α^*,β^*} is the densest subgraph

In this subsection, we will prove that F_{α^*,β^*} is the DS, which provides intuition for the IDS that we have defined. Besides, the theoretical results derived in the proof also serve as the vital theoretical foundation for our algorithm design. Specifically, let $c^* = \frac{S^*}{T^*}$. We first prove that $F_{\frac{\rho^*}{2\sqrt{c^*}},\frac{\rho^*\sqrt{c^*}}{2}}$ is the DS (Theorem 2). Then, in Lemma 1 and Theorem 3, we show that $F_{\frac{\rho^*}{2\sqrt{c^*}},\frac{\rho^*\sqrt{c^*}}{2}}$ is F_{α^*,β^*}, meaning that $F_{\frac{\rho^*}{2\sqrt{c^*}},\frac{\rho^*\sqrt{c^*}}{2}}$ maximizes $\alpha\cdot\beta$. Therefore, F_{α^*,β^*} is the DS.

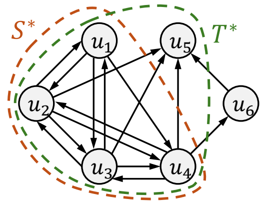
Theorem 2. *Given a graph G, DS is the $F_{\frac{\rho^*}{2\sqrt{c^*}},\frac{\rho^*\sqrt{c^*}}{2}}$, i.e., $G[S^*,T^*] = F_{\frac{\rho^*}{2\sqrt{c^*}},\frac{\rho^*\sqrt{c^*}}{2}}$.*

Proof. To prove the theorem, it is sufficient to show that when $\alpha = \frac{\rho^*}{2\sqrt{c^*}}$ and $\beta = \frac{\rho^*\sqrt{c^*}}{2}$, $G[S^*,T^*]$ satisfies the two conditions in Definition 3. For condition (1), assume for contradiction that there exist $S^- \subseteq S^*$, $T^- \subseteq T^*$, and $S^- \cup T^- \neq \emptyset$ such that $|E(S^*,T^*)| - |E(S^*\setminus S^-, T^*\setminus T^-)| < \frac{\rho^*}{2\sqrt{c^*}}|S^-| + \frac{\rho^*\sqrt{c^*}}{2}|T^-|$. Then we have $\rho(S^*\setminus S^-, T^*\setminus T^-) = \frac{|E(S^*\setminus S^-,T^*\setminus T^-)|}{\sqrt{|S^*\setminus S^-|\cdot|T^*\setminus T^-|}} > \frac{|E(S^*,T^*)| - \frac{\rho^*}{2\sqrt{c^*}}|S^-| - \frac{\rho^*\sqrt{c^*}}{2}|T^-|}{\sqrt{|S^*\setminus S^-|\cdot|T^*\setminus T^-|}} = \rho^*\frac{\frac{1}{2\sqrt{c^*}}|S^*\setminus S^-| + \frac{\sqrt{c^*}}{2}|T^*\setminus T^-|}{\sqrt{|S^*\setminus S^-|\cdot|T^*\setminus T^-|}} \geq \rho^*$, a contradiction that ρ^* is the maximum density. For condition (2), we can prove it using a method similar to that of condition (1). Therefore, $F_{\frac{\rho^*}{2\sqrt{c^*}},\frac{\rho^*\sqrt{c^*}}{2}} = G[S^*,T^*]$. □

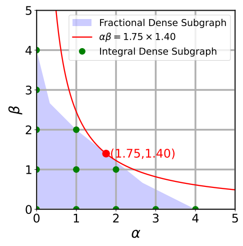
Lemma 1. *Given a non-empty $F_{\alpha,\beta}$ (resp., $D_{\alpha,\beta}$), we have $\rho(F^O_{\alpha,\beta},F^I_{\alpha,\beta}) \geq 2\sqrt{\alpha\beta}$ (resp., $\rho(D^O_{\alpha,\beta},D^I_{\alpha,\beta}) \geq 2\sqrt{\alpha\beta}$).*

Proof. By condition (1) in the definition of $F_{\alpha,\beta}$, when $S = F^O_{\alpha,\beta}$ and $T = F^I_{\alpha,\beta}$, we have $|E(F^O_{\alpha,\beta},F^I_{\alpha,\beta})| \geq \alpha|F^O_{\alpha,\beta}| + \beta|F^I_{\alpha,\beta}|$. Therefore, we have $\rho(F^O_{\alpha,\beta},F^I_{\alpha,\beta}) = \frac{|E(F^O_{\alpha,\beta},F^I_{\alpha,\beta})|}{\sqrt{|F^O_{\alpha,\beta}|\cdot|F^I_{\alpha,\beta}|}} \geq \frac{\alpha|F^O_{\alpha,\beta}|+\beta|F^I_{\alpha,\beta}|}{\sqrt{|F^O_{\alpha,\beta}|\cdot|F^I_{\alpha,\beta}|}} \geq 2\sqrt{\alpha\beta}$. Analogously, we can get $\rho(D^O_{\alpha,\beta},D^I_{\alpha,\beta}) \geq 2\sqrt{\alpha\beta}$. □

Theorem 3. *Given a graph G, we have $\alpha^* = \frac{\rho^*}{2\sqrt{c^*}}$ and $\beta^* = \frac{\rho^*\sqrt{c^*}}{2}$, thus $F_{\frac{\rho^*}{2\sqrt{c^*}},\frac{\rho^*\sqrt{c^*}}{2}}$ is F_{α^*,β^*}.*



(a) Example graph.



(b) The emptiness of $F_{\alpha,\beta}$ and $D_{\alpha,\beta}$ when varying α and β.

Figure 2: Example of maximizing $\alpha^*\cdot\beta^*$ and $\overline{\alpha}^*\cdot\overline{\beta}^*$.

Proof. For $F_{\alpha,\beta} \neq \emptyset$, it follows from Lemma 1 that $\rho^* \geq \rho(F_{\alpha,\beta}) \geq 2\sqrt{\alpha\beta}$, which implies $\alpha\beta \leq \frac{\rho^{*2}}{4}$. So, $\alpha\beta$ cannot exceed $\frac{\rho^{*2}}{4}$. Since $F_{\frac{\rho^*}{2\sqrt{c^*}},\frac{\rho^*\sqrt{c^*}}{2}}$ achieves $\alpha\beta = \frac{\rho^{*2}}{4}$, it maximizes $\alpha\beta$. □

Theorem 4. *Given a graph G, let two real numbers α^* and β^* satisfy $F_{\alpha^*,\beta^*} \neq \emptyset$ and maximize $\alpha^*\cdot\beta^*$. Then, F_{α^*,β^*} is the DS.*

Example 2. *Taking the directed graph in Figure 2a as an example, whose DS is $G[S^*,T^*]$. Figure 2b shows whether $F_{\alpha,\beta}$ and $D_{\alpha,\beta}$ are non-empty when varying α and β. When (α,β) is in the blue region, $F_{\alpha,\beta} \neq \emptyset$; when (α,β) is at the green coordinate points, $D_{\alpha,\beta} \neq \emptyset$. For F_{α^*,β^*}, we have $\alpha^* = 1.75$ and $\beta^* = 1.40$, and the red line represents all (α,β) where $\alpha\beta = 1.75 \times 1.40$. It can be seen that, within the blue region, $(1.75, 1.40)$ is the only point on the red line, i.e., maximizing $\alpha\beta$. For $D_{\overline{\alpha}^*,\overline{\beta}^*}$, both $(\overline{\alpha}^* = 2, \overline{\beta}^* = 1)$ and $(\overline{\alpha}^* = 1, \overline{\beta}^* = 2)$ maximize $\overline{\alpha}^*\cdot\overline{\beta}^*$, but both $D_{2,1}$ and $D_{1,2}$ equal the DS.*

According to Theorem 2 and Theorem 3, it follows that F_{α^*,β^*} is the DS. This indicates that, by maximizing $\alpha^*\cdot\beta^*$, the DS can be obtained. Therefore, intuitively, the IDS $D_{\overline{\alpha}^*,\overline{\beta}^*}$, obtained by maximizing $\overline{\alpha}^*\cdot\overline{\beta}^*$, should also have very high density. In the next subsection, we will prove that $D_{\overline{\alpha}^*,\overline{\beta}^*}$ indeed has a near-optimal density lower bound.

3.2 Density of the integral densest subgraph

The following theorem show a hierarchy property of $F_{\alpha,\beta}$ and $D_{\alpha,\beta}$, which is a vital property for our theoretical analysis and algorithms.

Theorem 5. *(1) Given $F_{\alpha,\beta}$ and F_{α^+,β^+}, if $\alpha^+ \geq \alpha$ and $\beta^+ \geq \beta$, then we have $F_{\alpha^+,\beta^+} \subseteq F_{\alpha,\beta}$.*

(2) Given $D_{\alpha,\beta}$ and D_{α^+,β^+}, if $\alpha^+ \geq \alpha$ and $\beta^+ \geq \beta$, then we have $D_{\alpha^+,\beta^+} \subseteq D_{\alpha,\beta}$.

(3) Given $D_{\alpha,\beta}$ and F_{α^+,β^+}, if $\alpha^+ \geq \alpha$ and $\beta^+ \geq \beta$, then we have $F_{\alpha^+,\beta^+} \subseteq D_{\alpha,\beta}$.

Proof. For (1), assume for contradiction that there exist $S = F^O_{\alpha^+,\beta^+} \setminus F^O_{\alpha,\beta}$ and $T = F^I_{\alpha^+,\beta^+} \setminus F^I_{\alpha,\beta}$, with $S \cup T \neq \emptyset$. Then we have $|E(F^O_{\alpha^+,\beta^+},F^I_{\alpha^+,\beta^+})| - |E(F^O_{\alpha^+,\beta^+}\setminus S, F^I_{\alpha^+,\beta^+}\setminus T)| \geq \alpha|S| + \beta|T|$, and $|E(F^O_{\alpha,\beta}\cup S, F^I_{\alpha,\beta}\cup T)| - |E(F^O_{\alpha,\beta},F^I_{\alpha,\beta})| < \alpha|S| + \beta|T|$. However, by analyzing the inclusion relationships between the subgraphs, we can see that $|E(F^O_{\alpha^+,\beta^+}, F^I_{\alpha^+,\beta^+})| - |E(F^O_{\alpha^+,\beta^+} \setminus S, F^I_{\alpha^+,\beta^+} \setminus T)| \leq |E(F^O_{\alpha,\beta} \cup S, F^I_{\alpha,\beta} \cup T)| - |E(F^O_{\alpha,\beta}, F^I_{\alpha,\beta})|$, leading to a contradiction. Thus, $F_{\alpha^+,\beta^+} \subseteq F_{\alpha,\beta}$. Analogously, (2) and (3) also hold. □

Then, we derive the following bound for the density of $D_{\overline{\alpha}^*,\overline{\beta}^*}$.

Theorem 6. *Given a graph G, we have $\rho^* = \rho(F^O_{\alpha^*,\beta^*}, F^I_{\alpha^*,\beta^*}) = 2\sqrt{\alpha^* \cdot \beta^*}$ and $\rho(D^O_{\overline{\alpha}^*,\overline{\beta}^*}, D^I_{\overline{\alpha}^*,\overline{\beta}^*}) \geq 2\sqrt{\lfloor\alpha^*\rfloor\lfloor\beta^*\rfloor}$.*

Proof. According to Theorem 3, we have $2\sqrt{\alpha^* \cdot \beta^*} = \rho^*$. By the conclusion (3) in Theorem 5, we know that $D_{\lfloor\alpha^*\rfloor,\lfloor\beta^*\rfloor}$ is non-empty, thus $\rho(D^O_{\overline{\alpha}^*,\overline{\beta}^*}, D^I_{\overline{\alpha}^*,\overline{\beta}^*}) \geq 2\sqrt{\overline{\alpha}^* \cdot \overline{\beta}^*} \geq 2\sqrt{\lfloor\alpha^*\rfloor \cdot \lfloor\beta^*\rfloor}$. □

According to Theorem 6, the difference in density between the DS and the IDS is only a *floor* operation. This indicates that the lower bound on the density of the IDS is quite tight. Our experiments have confirmed that the density of the IDS is at least 0.9994 times that of the DS, with a maximum difference of only 0.0271.

4 A NEW FLOW-BASED ALGORITHM

To compute $D_{\overline{\alpha}^*,\overline{\beta}^*}$, we need to vary the values of (α, β) to guess $(\overline{\alpha}^*, \overline{\beta}^*)$ and then compute $D_{\overline{\alpha}^*,\overline{\beta}^*}$. Therefore, we first present the approach to, given α and β, compute a single $D_{\alpha,\beta}$.

4.1 Compute a single $D_{\alpha,\beta}$

To obtain a single $D_{\alpha,\beta}$, we first transform the problem into an optimization problem, and then solve it using the minimum cut method to derive $D_{\alpha,\beta}$, i.e., mapping the minimum cut to $D_{\alpha,\beta}$. Below, we introduce the optimization formulation.

Theorem 7. *Given a graph G and two integers α and β, $D_{\alpha,\beta}$ is the only maximal subgraph $G[D^O, D^I]$ that minimizes $\alpha|D^O| + \beta|D^I| + \sum_{x\in V\setminus D^O} d^O_x + |E(D^O, V \setminus D^I)|$.*

Proof. Similar to the proof of Theorem 1, it can be shown that $D_{\alpha,\beta}$ is the only maximal subgraph $G[D^O, D^I]$ that minimizes $\alpha|D^O| + \beta|D^I| - |E(D^O, D^I)|$. We have $\alpha|D^O| + \beta|D^I| - |E(D^O, D^I)| = \alpha|D^O| + \beta|D^I| - \left(\sum_{x\in D^O} d^O_x - |E(D^O, V \setminus D^I)|\right) = \alpha|D^O| + \beta|D^I| + \sum_{x\in V\setminus D^O} d^O_x + |E(D^O, V \setminus D^I)| - \sum_{x\in V} d^O_x$. In the above equation, $\sum_{x\in V} d^O_x$ is a constant in a graph, thus $G[D^O, D^I]$ minimizes $\alpha|D^O| + \beta|D^I| + \sum_{x\in V\setminus D^O} d^O_x + |E(D^O, V \setminus D^I)|$. □

Next, we introduce some basic concepts of the flow network [14]. A flow network can be represented as a triplet (V_f, E_f, c), where V_f is the set of nodes, $s \in V_f$ is the source node, $t \in V_f$ is the sink node, E_f is the set of directed edges, and c is the capacity of the edges. The flow on an edge cannot exceed its capacity. The maximum flow of a flow network is equal to its minimum cut value, $\min_{S\subseteq V_f, T=V_f\setminus S} \text{cut}(S, T)$, where $s \in S$ and $t \in T$, and the cut value is defined as $\text{cut}(S, T) = \sum_{(x,y)\in E_f, x\in S, y\in T} c(x, y)$. Note that since $D_{\alpha,\beta}$ in Theorem 7 is maximal, we also define the minimum cut

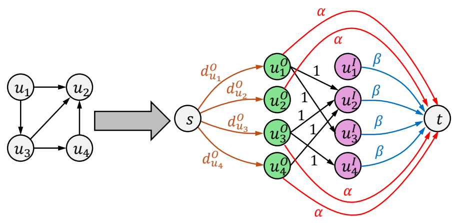


Figure 3: Example of (α, β)-dense flow network.

(S, T) as maximal here, i.e., there does not exist $S^+ \supset S, T^+ = V_f \setminus S^+$, such that $\text{cut}(S^+, T^+)$ also yields the minimum cut value.

Based on the above notions, we propose a novel flow network called (α, β)-dense flow network, whose definition is as follows: Given a directed graph $G = (V, E)$ and two non-negative integers α and β, the (α, β)-dense flow network is (V_f, E_f, c), where: (1) $V_f = \{s, t\} \cup V^O \cup V^I$, and for each node $x \in V$, add x^O to V^O, and add x^I to V^I; (2) For each node $x^O \in V^O$, add an edge (s, x^O) to E_f with capacity $c(s, x^O) = d^O_x$; (3) For each edge (x, y) in E, add an edge (x^O, y^I) to E_f with capacity 1; (4) For each node $x^O \in V^O$, add an edge (x^O, t) to E_f with capacity $c(x^O, t) = \alpha$; (5) For each node $x^I \in V^I$, add an edge (x^I, t) to E_f with capacity $c(x^I, t) = \beta$. An example is shown in Figure 3.

Based on the (α, β)-dense flow network, we propose Algorithm 1, which can compute $D_{\alpha,\beta}$ given α and β. First, the algorithm constructs the flow network (lines 1–7), then computes the minimum cut (S, T) on this flow network (line 8), and finally derives $D_{\alpha,\beta}$ based on (S, T) (lines 8–11). The correctness and complexity of Algorithm 1 are analyzed as follows.

Theorem 8. *Algorithm 1 can correctly output $D_{\alpha,\beta}$ with a time complexity of $O(|E|^{1.5})$ and a space complexity of $O(|E|)$.*

Proof. For any (S, T), where $s \in S, t \in T, S \subseteq V^f$, and $T = V^f \setminus S$, analyzing the structure of the flow network shows that $\text{cut}(S, T) = \alpha|S\setminus\{s\}| + \beta|T\setminus\{t\}| + \sum_{x^O\in V^O\setminus S} d^O_x + |E_f(S\setminus\{s\}, T\setminus\{t\})|$. According to Theorem 7, (S, T) is the minimum cut if and only if $S = D^O_{\alpha,\beta} \cup \{s\}$ and $T = D^I_{\alpha,\beta} \cup \{t\}$. Therefore, Algorithm 1 correctly outputs $D_{\alpha,\beta}$.

For the time complexity, it can be derived from [5] that the (α, β)-dense flow network is an AUC-2 network (i.e., all weights of the edges in the flow network are 1 except for the edges connected to s and t). The time and space complexity of the minimum cut computation on such an AUC-2 network is $O(|E|^{1.5})$ and $O(|E|)$, respectively [5]. Therefore, the time and space complexity of Algorithm 1 is $O(|E|^{1.5})$ and $O(|E|)$, respectively. □

Armed with the GetD algorithm, we propose a basic network-flow algorithm to compute the IDS.

4.2 A basic algorithm: GetIDS

To compute the IDS, we need to maximize $\alpha\beta$. A straightforward idea is to compute all $D_{\alpha,\beta}$ with large $\alpha\beta$, and then take the maximum value as $\overline{\alpha}^* \cdot \overline{\beta}^*$. Based on this idea, we define $\beta_{\max}(\alpha)$ (resp., $\alpha_{\max}(\beta)$) as the maximum integer such that $D_{\alpha,\beta_{\max}(\alpha)} \neq \emptyset$ (resp.,

Algorithm 1: GetD(G, α, β)

```
Input: A directed graph $G = (V, E)$, two non-negative integers $\alpha$ and $\beta$.
Output: $D_{\alpha,\beta}$.
1  $V^O \leftarrow \emptyset, V^I \leftarrow \emptyset, E_f \leftarrow \emptyset, D^O_{\alpha,\beta} \leftarrow \emptyset, D^I_{\alpha,\beta} \leftarrow \emptyset$;
2  foreach $x \in V$ do $V^O \leftarrow x^O, V^I \leftarrow x^I$;
3  foreach $(x, y) \in E$ do $E_f \leftarrow E_f \cup \{(x^O, y^I)\}, c(x^O, y^I) \leftarrow 1$;
4  foreach $x^O \in V^O$ do $E_f \leftarrow E_f \cup \{(s, x^O)\}, c(s, x^O) \leftarrow d^O_x$;
5  foreach $x^O \in V^O$ do $E_f \leftarrow E_f \cup \{(x^O, t)\}, c(x^O, t) \leftarrow \alpha$;
6  foreach $y^I \in V^I$ do $E_f \leftarrow E_f \cup \{(y^I, t)\}, c(y^I, t) \leftarrow \beta$;
7  $V_f \leftarrow V^O \cup V^I \cup \{s, t\}$;
8  Compute the minimum cut $(S, T)$ in the flow network $(V_f, E_f, c)$;
9  foreach $x^O \in S \setminus s$ do $D^O_{\alpha,\beta} \leftarrow x$;
10 foreach $y^I \in T \setminus t$ do $D^I_{\alpha,\beta} \leftarrow y$;
11 return $G[D^O_{\alpha,\beta}, D^I_{\alpha,\beta}]$;
```

$D_{\alpha_{\max}(\beta),\beta} \neq \emptyset$), given an integer α (resp., β). Thus, each $\alpha \times \beta_{\max}(\alpha)$ and $\alpha_{\max}(\beta) \times \beta$ could potentially be $\overline{\alpha}^* \cdot \overline{\beta}^*$. By selecting the maximum value among all $\alpha \times \beta_{\max}(\alpha)$ and $\alpha_{\max}(\beta) \times \beta$, we can obtain $\overline{\alpha}^* \cdot \overline{\beta}^*$.

First, we enumerate α and compute its corresponding $\beta_{\max}(\alpha)$ and the product $\alpha \times \beta_{\max}(\alpha)$. Then, we enumerate β and compute its corresponding $\alpha_{\max}(\beta)$ and the product $\alpha_{\max}(\beta) \times \beta$. The maximum value of $\alpha\beta$ obtained in this process is $\overline{\alpha}^* \cdot \overline{\beta}^*$. Based on this idea, we propose the GetIDS algorithm as shown in Algorithm 2. To determine the range for enumerating α and β, the algorithm first computes p using the procedure Getp (line 1), where p is the largest integer such that $D_{p,p} \neq \emptyset$. To compute p, Getp first uses binary lifting search to obtain an approximate range $[p_l, p_u]$ for p (line 19), and then performs a binary search within this range to find the exact value of p (lines 20–23). Then, α is enumerated from 1 to p, and for each α, we compute $\beta_{\max}(\alpha)$ and $\alpha \times \beta_{\max}(\alpha)$ (lines 3–11). Similarly, β is enumerated from 1 to p, and for each β, we compute $\alpha_{\max}(\beta)$ and $\alpha_{\max}(\beta) \times \beta$ (lines 12–15). The maximum $\alpha\beta$ obtained in this process is stored in the variable $maxab = maxa \times maxb$. Finally, the algorithm outputs $D_{maxa,maxb} = D_{\overline{\alpha}^*,\overline{\beta}^*}$ (line 16). Next, we show the correctness and complexity of GetIDS.

Theorem 9. *Algorithm 2 can correctly output $D_{\overline{\alpha}^*,\overline{\beta}^*}$ with a time complexity of $O(p \cdot \log|V| \cdot |E|^{1.5})$ and a space complexity of $O(|E|)$.*

Theoretically, $p \leq \sqrt{|E|}/2$, because $|E(D^O_{p,p}, D^I_{p,p})| \geq p|D^O_{p,p}| + p|D^I_{p,p}| \geq p\frac{|E(D^O_{p,p},D^I_{p,p})|}{|D^I_{p,p}|} + p|D^I_{p,p}| \geq 2p\sqrt{|E(D^O_{p,p}, D^I_{p,p})|}$. This implies that $p \leq \sqrt{|E(D^O_{p,p}, D^I_{p,p})|}/2 \leq \sqrt{|E|}/2$. However, p approaches this theoretical upper bound $\sqrt{|E|}/2$ only when the graph is nearly a complete graph. In real-world sparse graphs, p typically satisfies $p \ll \sqrt{|E|}/2$, as indicated in our experiments (details in Table 2). For example, on the dataset UK with $|E| = 936.4$M, $p = 454$, which is far less than $\sqrt{|E|}/2 \approx 15,300$. Therefore, the time complexity of GetIDS, $O(p \cdot \log|V| \cdot |E|^{1.5})$, is significantly lower than that of the state-of-the-art algorithm for computing DS, $O(|V|^2 t_{FW})$, where t_{FW} includes the time required for several convex programming and flow network computations.

5 A DIVIDE-AND-CONQUER ALGORITHM

In the GetIDS algorithm, for each $\alpha = 1, \ldots, p$ and $\beta = 1, \ldots, p$, it performs a binary search to compute $\beta_{\max}(\alpha)$ and $\alpha_{\max}(\beta)$. However, only the enumeration when α or β equals $\overline{\alpha}^*$ or $\overline{\beta}^*$ can actually compute $\overline{\alpha}^* \cdot \overline{\beta}^*$. In most cases, $\alpha \times \beta_{\max}(\alpha)$ and $\alpha_{\max}(\beta) \times \beta$ are relatively small, and there is no need to perform time-consuming binary searches and flow network calculations for them. Next, we develop an improved algorithm GetIDS++, which selectively decides whether to perform binary searches for the currently enumerated α and β, and in most cases, it avoids the costly binary search. The decision-making method used by GetIDS++ is based on upper and lower bounds of $\beta_{\max}(\alpha)$ and $\alpha_{\max}(\beta)$ to determine whether it is necessary to use binary search to compute their exact values. The theoretical foundation for obtaining these bounds comes from our following emptiness theorem and non-emptiness theorem.

Lemma 2. *The following equivalences hold:*

$$(1)\ F_{\alpha,\beta} \neq \emptyset \Leftrightarrow \exists (S,T) \subseteq (V,V), |E(S,T)| \geq \alpha|S| + \beta|T|\ (\alpha, \beta \in \mathbb{R});$$
$$(2)\ F_{\alpha,\beta} = \emptyset \Leftrightarrow \forall (S,T) \subseteq (V,V), |E(S,T)| < \alpha|S| + \beta|T|\ (\alpha, \beta \in \mathbb{R});$$
$$(3)\ D_{\alpha,\beta} \neq \emptyset \Leftrightarrow \exists (S,T) \subseteq (V,V), |E(S,T)| \geq \alpha|S| + \beta|T|\ (\alpha, \beta \in \mathbb{N});$$
$$(4)\ D_{\alpha,\beta} = \emptyset \Leftrightarrow \forall (S,T) \subseteq (V,V), |E(S,T)| < \alpha|S| + \beta|T|\ (\alpha, \beta \in \mathbb{N}).$$

Proof. For equivalence (1), '⇒': When $F_{\alpha,\beta} \neq \emptyset$, it is clear that there exists $(S,T) = (F^O_{\alpha,\beta}, F^I_{\alpha,\beta})$ such that $|E(S,T)| \geq \alpha|S| + \beta|T|$. '⇐': When there exists (S,T) such that $|E(S,T)| \geq \alpha|S| + \beta|T|$, $F_{\alpha,\beta}$ must be non-empty, otherwise (S,T) would violate condition (2) in the definition of $F_{\alpha,\beta}$. Using a similar proof, equivalence (3) can also be proven. Equivalence (2) and equivalence (4) are the inverse of equivalence (1) and equivalence (3), respectively, so equivalence (2) and equivalence (4) also hold. □

Theorem 10. **(Emptiness Theorem)** *(1) If $F_{\alpha_1,\beta_1} = F_{\alpha_2,\beta_2} = \emptyset$, then for any $\lambda \in [0,1]$, we have $F_{\lambda\alpha_1+(1-\lambda)\alpha_2, \lambda\beta_1+(1-\lambda)\beta_2} = \emptyset$; (2) If $D_{\alpha_1,\beta_1} = D_{\alpha_2,\beta_2} = \emptyset$, then for any $\lambda \in [0,1]$, we have $D_{\lceil\lambda\alpha_1+(1-\lambda)\alpha_2\rceil, \lceil\lambda\beta_1+(1-\lambda)\beta_2\rceil} = \emptyset$.*

Proof. According to equivalence (2) in Lemma 2, since $F_{\alpha_1,\beta_1} = F_{\alpha_2,\beta_2} = \emptyset$, for all $(S,T) \subseteq (V,V)$, we have $|E(S,T)| < \alpha_1|S| + \beta_1|T|$ and $|E(S,T)| < \alpha_2|S| + \beta_2|T|$. Multiplying these two inequalities by λ and $(1-\lambda)$ respectively, and then adding them, we get $|E(S,T)| < (\lambda\alpha_1 + (1-\lambda)\alpha_2)|S| + (\lambda\beta_1 + (1-\lambda)\beta_2)|T|$. Therefore, $F_{\lambda\alpha_1+(1-\lambda)\alpha_2, \lambda\beta_1+(1-\lambda)\beta_2} = \emptyset$. For $D_{\lceil\lambda\alpha_1+(1-\lambda)\alpha_2\rceil, \lceil\lambda\beta_1+(1-\lambda)\beta_2\rceil} = \emptyset$, a similar method can be used to prove it. □

The emptiness theorem can be used to determine the upper bound of $\beta_{\max}(\alpha_m)$ when we know $D_{\alpha_1,\beta_1} = D_{\alpha_2,\beta_2} = \emptyset$, where $\alpha_m \in [\alpha_1, \alpha_2]$. By setting $\lambda = \frac{\alpha_m - \alpha_2}{\alpha_1 - \alpha_2}$ and letting $\beta_m = \lceil\lambda\beta_1 + (1-\lambda)\beta_2\rceil = \lceil\frac{\beta_1(\alpha_m-\alpha_2)+\beta_2(\alpha_1-\alpha_m)}{\alpha_1-\alpha_2}\rceil$, we get $D_{\alpha_m,\beta_m} = \emptyset$, which implies $\beta_{\max}(\alpha_m) \leq \beta_m - 1$. Without further computation, we can conclude that $\alpha_m\beta_{\max}(\alpha_m)$ can be at most $\alpha_m(\beta_m - 1)$. If this value is small (e.g., less than $\alpha_1 \cdot \beta_1$ or $\alpha_2 \cdot \beta_2$), $\alpha_m\beta_{\max}(\alpha_m)$ must not be $\overline{\alpha}^* \cdot \overline{\beta}^*$ and there is no need to calculate the exact value of $\beta_{\max}(\alpha_m)$, thus, we can safely prune α_m.

Algorithm 2: GetIDS(G)

```
   Input: A directed graph $G = (V, E)$.
   Output: $D_{\overline{\alpha}^*,\overline{\beta}^*}$.
1  $p \leftarrow$ Getp($G$);
2  $maxab \leftarrow p \times p$, $maxa \leftarrow p$, $maxb \leftarrow p$;
3  for $\alpha = 1, \ldots, p$ do
4      $\beta_l \leftarrow p$, $\beta_u \leftarrow \max_{x \in V} d_x^I$;
5      while $\beta_l < \beta_u$ do
6          $\beta_m \leftarrow \lfloor(\beta_l + \beta_u)/2\rfloor$;
7          if GetD($G, \alpha, \beta_m$) $\neq \emptyset$ then $\beta_l \leftarrow \beta_m + 1$;
8          else $\beta_u \leftarrow \beta_m$;
9      $\beta_{\max}(\alpha) \leftarrow \beta_l$;
10     if $\alpha \times \beta_{\max}(\alpha) > maxab$ then
11         $maxab \leftarrow \alpha \times \beta_{\max}(\alpha)$, $maxa \leftarrow \alpha$, $maxb \leftarrow \beta_{\max}(\alpha)$;
12 for $\beta = 1, \ldots, p$ do
13     Use the same method as lines 4-9 to compute $\alpha_{\max}(\beta)$;
14     if $\alpha_{\max}(\beta) \times \beta > maxab$ then
15         $maxab \leftarrow \alpha_{\max}(\beta) \times \beta$, $maxa \leftarrow \alpha_{\max}(\beta)$, $maxb \leftarrow \beta$;
16 return GetD($G, maxa, maxb$);
17 Function Getp($G$)
18     $p_l \leftarrow 1$, $p_u \leftarrow 2$;
19     while GetD($G, p_u, p_u$) $\neq \emptyset$ do $p_l \leftarrow 2p_l$, $p_u \leftarrow 2p_u$;
20     while $p_l < p_u$ do
21         $p_m \leftarrow \lfloor(p_l + p_u)/2\rfloor$;
22         if GetD($G, p_m, p_m$) $\neq \emptyset$ then $p_l \leftarrow p_m + 1$;
23         else $p_u \leftarrow p_m$;
24     return $p_l$;
```

Theorem 11. **(Non-emptiness Theorem)** *(1) If $F_{\alpha,\beta} \neq \emptyset$, then for any $t \in [0,1]$, we have $F_{t(\alpha+\beta/c_F),(1-t)(\alpha \cdot c_F+\beta)} \neq \emptyset$, where $c_F = |F^O_{\alpha,\beta}|/|F^I_{\alpha,\beta}|$; (2) If $D_{\alpha,\beta} \neq \emptyset$, then for any $t \in [0,1]$, we have $D_{\lfloor t(\alpha+\beta/c_D)\rfloor,\lfloor(1-t)(\alpha \cdot c_D+\beta)\rfloor} \neq \emptyset$, where $c_D = |D^O_{\alpha,\beta}|/|D^I_{\alpha,\beta}|$.*

Proof. If $F_{\alpha,\beta} \neq \emptyset$, then there exists $(S,T) = (F^O_{\alpha,\beta}, F^I_{\alpha,\beta})$ such that $|E(S,T)| \geq \alpha|S| + \beta|T| = t(\alpha|S| + \beta|T|) + (1-t)(\alpha|S| + \beta|T|) = t(\alpha + \beta/c_F)|S| + (1-t)(\alpha \cdot c_F + \beta)|T|$. According to equivalence (3) in Lemma 2, we know that $F_{t(\alpha+\beta/c_F),(1-t)(\alpha \cdot c_F+\beta)} \neq \emptyset$. Analogously, $D_{\lfloor t(\alpha+\beta/c_D)\rfloor,\lfloor(1-t)(\alpha \cdot c_D+\beta)\rfloor} \neq \emptyset$ is true. □

Using the non-emptiness theorem, if we have already computed $D_{\alpha,\beta}$, we not only know that $\overline{\alpha}^* \cdot \overline{\beta}^* \geq \alpha\beta$ but also $\overline{\alpha}^* \cdot \overline{\beta}^* \geq \max_{t \in [0,1]} \{\lfloor t(\alpha + \beta/c_D)\rfloor \cdot \lfloor(1-t)(\alpha \cdot c_D + \beta)\rfloor\}$. Because when $t = \frac{\alpha}{\alpha+\beta/c_d}$, $\lfloor t(\alpha + \beta/c_D)\rfloor \cdot \lfloor(1-t)(\alpha \cdot c_D + \beta)\rfloor = \alpha\beta$, this maximum value is certainly not smaller than $\alpha\beta$, providing a better lower bound for $\overline{\alpha}^* \cdot \overline{\beta}^*$. A better lower bound allows us to prune more α and β values, thereby improving efficiency.

Based on the emptiness theorem and the non-emptiness theorem, we propose the algorithm GetIDS++, as shown in Algorithm 3. Similar to the GetIDS algorithm, GetIDS++ considers each $\alpha = 1, \ldots, p$ and $\beta = 1, \ldots, p$. However, unlike GetIDS, which sequentially enumerates α and β, GetIDS++ uses a divide-and-conquer approach to leverage the emptiness theorem. For α, the function Divide-a(α_l, α_u)

Algorithm 3: GetIDS++(G)

```
   Input: A directed graph $G = (V, E)$.
   Output: $D_{\overline{\alpha}^*,\overline{\beta}^*}$.
1  $p \leftarrow$ Getp($G$);
2  $maxab \leftarrow p \times p$, $maxa \leftarrow p$, $maxb \leftarrow p$;
3  $upper[0] \leftarrow \max_{x \in V} d_x^I$, $upper[p+1] \leftarrow p$;
4  Divide-a($0, p+1$);
5  $upper[0] \leftarrow \max_{x \in V} d_x^O$, $upper[p+1] \leftarrow p$;
6  Divide-b($0, p+1$);
7  return GetD($G, maxa, maxb$);
8  Function Divide-a($\alpha_l, \alpha_u$)
9      if $\alpha_l > \alpha_u - 2$ then return;
10     $\alpha_m \leftarrow \lfloor(\alpha_l + \alpha_u)/2\rfloor$;
11     $\alpha_1 \leftarrow \alpha_l$, $\alpha_2 \leftarrow \alpha_u$, $\beta_1 \leftarrow upper[\alpha_l] + 1$, $\beta_2 \leftarrow upper[\alpha_u] + 1$;
12     $upper[\alpha_m] \leftarrow \lceil \frac{\beta_1(\alpha_m-\alpha_2)+\beta_2(\alpha_1-\alpha_m)}{\alpha_1-\alpha_2} \rceil - 1$;   // Theorem 10
13     if $\alpha_m \times upper[\alpha_m] \leq maxab$ then $\alpha_m$ is pruned;
14     else if GetD($\alpha_m, \lfloor \frac{maxab}{\alpha_m} \rfloor + 1$) $= \emptyset$ then $\alpha_m$ is pruned;
15     else // Binary search
16         $\beta_l \leftarrow \lfloor \frac{maxab}{\alpha_m} \rfloor + 1$, $\beta_u \leftarrow upper[\alpha_m]$;
17         while $\beta_l < \beta_u$ do
18             $\beta_m \leftarrow \lfloor(\beta_l + \beta_u)/2\rfloor$;
19             if GetD($G, \alpha_m, \beta_m$) $\neq \emptyset$ then $\beta_l \leftarrow \beta_m + 1$;
20             else $\beta_u \leftarrow \beta_m$;
21         $\beta_{\max}(\alpha_m) \leftarrow \beta_l$, $upper[\alpha_m] \leftarrow \beta_{\max}(\alpha_m)$;
22         $D_{\alpha_m,\beta_{\max}(\alpha_m)} \leftarrow$GetD($G, \alpha_m, \beta_{\max}(\alpha_m)$);
23         $c_D \leftarrow |D^O_{\alpha_m,\beta_{\max}(\alpha_m)}|/|D^I_{\alpha_m,\beta_{\max}(\alpha_m)}|$;
24         forall $\alpha' = \lfloor t(\alpha_m + \beta_{\max}(\alpha_m)/c_D)\rfloor$, $\beta' = \lfloor(1-t)(\alpha_m \cdot c_D + \beta_{\max}(\alpha_m))\rfloor$, where $t \in [0,1]$ do
25             if $\alpha' \times \beta' > maxab$ then // Theorem 11
26                 $maxab \leftarrow \alpha' \times \beta'$, $maxa \leftarrow \alpha'$, $maxb \leftarrow \beta'$;
27     Divide-a($\alpha_l, \alpha_m$);
28     Divide-a($\alpha_m, \alpha_u$);
29 Function Divide-b($\beta_l, \beta_u$)
30     Same as lines 9-28 but interchanging $\alpha$ with $\beta$;
```

in GetIDS++ is responsible for considering all alpha values in the range $[\alpha_l + 1, \alpha_u - 1]$. First, it determines a middle α value α_m using the average of α_l and α_u (line 12), then considers α_m. Next, the algorithm recursively calls Divide-a(α_l, α_m) and Divide-a(α_m, α_u) to further consider the alpha values in $[\alpha_l + 1, \alpha_m - 1]$ and $[\alpha_m + 1, \alpha_u - 1]$, respectively (lines 27-28).

For a fixed α_m, the algorithm first uses the emptiness theorem to obtain an upper bound $upper[\alpha_m]$ for $\beta_{\max}(\alpha_m)$ (line 12). If $\alpha_m \times upper[\alpha_m]$ is still not greater than $maxab$, then $\alpha_m \times \beta_{\max}(\alpha_m)$ will also not exceed $maxab$, and α_m can be pruned without calculating the exact value of $\beta_{\max}(\alpha_m)$ (line 13). If $\alpha_m \times upper[\alpha_m]$ is greater than $maxab$, this suggests that $\alpha_m \times \beta_{\max}(\alpha_m)$ might exceed $maxab$. In this case, on line 13, the algorithm sets $\beta = \lfloor \frac{maxab}{\alpha_m} \rfloor + 1$, i.e., the smallest integer such that $\alpha_m \times \beta > maxab$, and tests whether $D_{\alpha_m,\lfloor \frac{maxab}{\alpha_m} \rfloor+1}$ is empty. If it is empty, this indicates that $\alpha_m \times \beta_{\max}(\alpha_m)$ will also not exceed $maxab$, and α_m is pruned (line 13). If it is not empty, this means that $\alpha_m \times \beta_{\max}(\alpha_m)$ is greater than $maxab$,

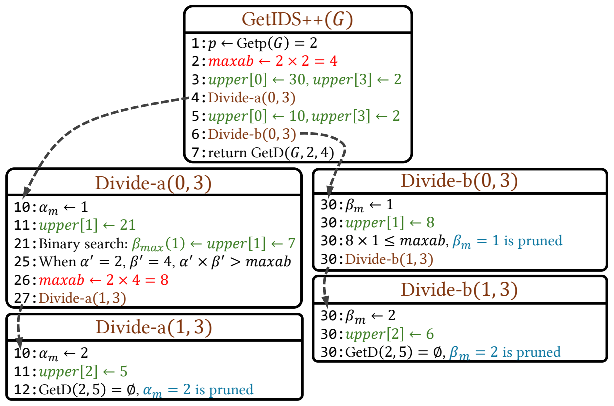

GetIDS++(G)
1: $p \leftarrow$ Getp$(G) = 2$
2: $maxab \leftarrow 2 \times 2 = 4$
3: $upper[0] \leftarrow 30, upper[3] \leftarrow 2$
4: Divide-a$(0, 3)$
5: $upper[0] \leftarrow 10, upper[3] \leftarrow 2$
6: Divide-b$(0, 3)$
7: return GetD$(G, 2, 4)$
Divide-a$(0, 3)$
10: $\alpha_m \leftarrow 1$
11: $upper[1] \leftarrow 21$
21: Binary search: $\beta_{max}(1) \leftarrow upper[1] \leftarrow 7$
25: When $\alpha' = 2, \beta' = 4, \alpha' \times \beta' > maxab$
26: $maxab \leftarrow 2 \times 4 = 8$
27: Divide-a$(1, 3)$
Divide-a$(1, 3)$
10: $\alpha_m \leftarrow 2$
11: $upper[2] \leftarrow 5$
12: GetD$(2, 5) = \emptyset$, $\alpha_m = 2$ is pruned
Divide-b$(0, 3)$
30: $\beta_m \leftarrow 1$
30: $upper[1] \leftarrow 8$
30: $8 \times 1 \leq maxab$, $\beta_m = 1$ is pruned
30: Divide-b$(1, 3)$
Divide-b$(1, 3)$
30: $\beta_m \leftarrow 2$
30: $upper[2] \leftarrow 6$
30: GetD$(2, 5) = \emptyset$, $\beta_m = 2$ is pruned

Figure 4: An illustrative example of GetIDS++.

and at this point, a binary search is performed to calculate the exact value of $\beta_{\max}(\alpha_m)$ (lines 16-21). Although the algorithm could directly use $\alpha_m \times \beta_{\max}(\alpha_m)$ to update *maxab*, a better approach is to potentially find a larger value to update *maxab* using the non-emptiness theorem. Thus, in lines 24-26, the algorithm uses the method from the non-emptiness theorem to update *maxab*. After the update terminates, the algorithm proceeds with deeper recursive calls (lines 27-28).

The GetIDS++ algorithm uses the function Divide-a to compute $D_{\overline{\alpha}^*,\overline{\beta}^*}$. First, GetIDS++ computes p (line 1). Since the function Divide-a(α_l, α_u) requires $upper[\alpha_l]$ and $upper[\alpha_u]$ in line 11, these values need to be computed before calling Divide-a(α_l, α_u). The algorithm GetIDS++ sets $upper[0]$ as $\max_{x \in V} d^I_x$, which is the upper bound for $\beta_{\max}(0)$, and sets p as $upper[p+1]$, which is the upper bound for $\beta_{\max}(p+1)$ (line 3). Then, the algorithm calls the function Divide-a$(0, p+1)$ to consider all alpha values and update *maxab* (line 4). Symmetrically, the algorithm considers all beta values (lines 5-6). After considering all alpha and beta values, *maxab* has been updated to its maximum value $\overline{\alpha}^* \cdot \overline{\beta}^*$, and the algorithm then returns $D_{maxa,maxb}$. Next, we give an running example of GetIDS++ and prove its correctness.

Example 3. *We present the execution of* GetIDS++ *on the* Actor[1] *dataset ($|V| = 32,271$, $|E| = 36,242$) in Figure 4. The figure illustrates the critical steps of the algorithm and their corresponding line numbers in the pseudocode. When invoking* GetIDS++(G)*, the algorithm first calls* Getp(G) *to compute $p = 2$. Since $D_{2,2} \neq \emptyset$, maxab can be updated as maxab = maxa$\times$maxb = 2×2. With $p = 2$, the algorithm considers $\alpha = 1, 2$ and their corresponding $\beta_{\max}(\alpha)$ values, as well as $\beta = 1, 2$ and their corresponding $\alpha_{\max}(\beta)$ values. These four values of α and β are considered in the four* Divide *functions shown in the figure, respectively.*

In Divide-a$(0, 3)$*, the algorithm considers $\alpha_m = 1$ and its $\beta_{\max}(1)$. First, it computes the upper bound for $\beta_{\max}(1)$ as $upper[1] = 21$. Since $\alpha_m \times upper[1] = 21 > maxab$ (line 13) and $D_{1,5} \neq \emptyset$ (line 14), $\alpha_m = 1$ cannot be pruned. Instead, binary search is used to precisely calculate $\beta_{\max}(\alpha_m) = 7$, which also indicates that $D_{1,7} \neq \emptyset$. Although $1\times7 > maxab = 4$, the algorithm does not update maxab to 7. Instead, it searches for a better solution in lines 24-26. When $t = 0.6$, $\alpha' = 2$, and $\beta' = 4$, we have $D_{2,4} \neq \emptyset$. Therefore, maxab is updated to a higher value of $2 \times 4 = 8$.*

The algorithm then proceeds to Divide-a$(1, 3)$ *to consider $\alpha_m = 2$. Here, if and only if $\beta_{\max}(2) \geq 5$ would result in $2 \times \beta_{\max}(2) > maxab$, and α_m can be pruned. To confirm whether $D_{2,5}$ is empty, the algorithm invokes* GetD$(2, 5)$*. Since $D_{2,5} = \emptyset$, $\alpha_m = 2$ is pruned.*

Next, the algorithm returns to GetIDS++ *and enters* Divide-b$(0, 3)$ *to consider $\beta_m = 1$. First, it calculates $\alpha_{\max}(1) \leq upper[1] = 8$. As a result, $\beta_m \times \alpha_{\max}(\beta_m) \leq 1 \times 8 \leq maxab$, and the current β_m is also pruned. Using a process similar to* Divide-a$(1, 3)$ *for $\alpha_m = 2$, it is confirmed in* Divide-b$(1, 3)$ *that $\beta_m = 2$ can also be pruned. Finally, when $\alpha = 2$ and $\beta = 4$, the product $\alpha \times \beta$ reaches its maximum value, and* GetIDS++ *returns $D_{2,4}$.*

Theorem 12. *Algorithm* GetIDS++ *can correctly output $D_{\overline{\alpha}^*,\overline{\beta}^*}$.*

Proof. We need to prove that during the execution of GetIDS++, *maxab* can be updated to $\overline{\alpha}^* \cdot \overline{\beta}^*$. First, the return condition in Divide-a(α_l, α_u) at line 9 ensures that all alpha values within the range $[\alpha_l + 1, \alpha_u - 1]$ are considered, i.e., α_m will be set to every alpha value in the range $[\alpha_l + 1, \alpha_u - 1]$. The same logic applies to Divide-b. Moreover, the initial values of α_l, α_u, β_l, and β_u in lines 4 and 6 ensure that $\overline{\alpha}^*$ and $\overline{\beta}^*$ will certainly be covered. Without loss of generality, assume that during a call to Divide-a(α_l, α_u), α_m is set to $\overline{\alpha}^*$. At this point, $maxab < \overline{\alpha}^* \cdot \overline{\beta}^*$, so in line 13, we have $\alpha_m \times upper[\alpha_m] \geq \alpha_m \times \beta_{\max}(\alpha_m) = \overline{\alpha}^* \cdot \overline{\beta}^* > maxab$, meaning that $\overline{\alpha}^*$ will not be pruned in line 13. Then, in line 14, since $\lfloor \frac{maxab}{\alpha_m} \rfloor + 1 \leq \overline{\beta}^*$, the test GetD$(\alpha_m, \lfloor \frac{maxab}{\alpha_m} \rfloor + 1)$ will be non-empty, and $\overline{\alpha}^*$ will not be pruned in line 14 either. During the binary search, the correctness of the bounds is ensured by the correctness of the emptiness theorem, so the binary search will correctly compute $\beta_{\max}(\alpha_m) = \overline{\beta}^*$. Finally, in the forall-loop at line 24, when $t = \frac{\alpha_m}{\alpha + \beta_{\max}(\alpha_m)/c_D}$, we will have $\alpha' = \overline{\alpha}^*$ and $\beta' = \overline{\beta}^*$. As a result, *maxab*, *maxa*, and *maxb* will be updated to $\overline{\alpha}^* \cdot \overline{\beta}^*$, $\overline{\alpha}^*$, and $\overline{\beta}^*$, respectively. Therefore, GetIDS++ correctly returns $D_{\overline{\alpha}^*,\overline{\beta}^*}$. □

The time complexity of GetIDS++ is $O(n_f \cdot |E|^{1.5})$, where n_f is the number of network flow computations performed in GetIDS++. For each α_m and β_m, the worst-case scenario involves using binary search and network flow, so theoretically $n_f \in O(p \cdot \log |V|)$. However, in the real-world graphs used in our experiments, $n_f \ll p \cdot \log |V|$. This is because about 99% of the α_m and β_m values were pruned in line 13 and line 14, making it unnecessary to use binary search to compute the exact values of $\beta_{\max}(\alpha_m)$ and $\alpha_{\max}(\beta_m)$.

6 APPROXIMATION IDS ALGORITHMS

This section further studies the approximation IDS algorithm. First, we introduce some existing approximation algorithms which originally designed for the traditional DS problem. Next, we propose a novel $(2 + \epsilon)$-approximation algorithm with a near-linear time complexity of $O(|E| \log_{1+\epsilon} |V|)$, which offers both strong approximation performance and high efficiency.

[1] Data source: https://docs.conscia.ai/solutions/dx-graph/tutorials/example-movie-dataset.

Algorithm 4: SingleCore(G)[25]

```
Input: A directed graph $G = (V, E)$.
Output: A $(\sqrt{c^*} + \frac{1}{\sqrt{c^*}})$-approximate IDS $\widetilde{D}$.
1 $\widetilde{\rho^*} \leftarrow \rho(V,V)$, $\widetilde{D} \leftarrow G$, $G'[S,T] = G$, $k \leftarrow 0$;
2 while $G'$ is not empty do
3     $x = \text{argmin}_{x \in S}\, d_x^O(G')$, $y = \text{argmin}_{y \in T}\, d_y^I(G')$;
4     if $\min\{d_x^O, d_y^I\} > k$ then
5         $k \leftarrow \min\{d_x^O, d_y^I\}$;  // Now, we have $G'[S,T] = C_{k,k}$
6     if $d_x^O \le d_y^I$ then $G'[S,T] \leftarrow G'[S \setminus \{x\}, T]$, $S \leftarrow S \setminus \{x\}$;
7     else $G'[S,T] \leftarrow G'[S, T \setminus \{y\}]$, $T \leftarrow T \setminus \{y\}$;
8     if $\rho(S,T) > \widetilde{\rho^*}$ then $\widetilde{\rho^*} \leftarrow \rho(S,T)$, $\widetilde{D} \leftarrow G'[S,T]$;
9 return $\widetilde{D}$;
```

It is worth mentioning that the $2+\epsilon$ approximation factor of our algorithm is based on the DS problem, where the density of the approximate subgraph $\tilde{\rho}$ satisfies the condition $\tilde{\rho} \ge \frac{1}{2+\epsilon}\rho^*$. However, since the density of IDS is always no greater than the density of DS, the density of these approximate subgraphs must also satisfy $\tilde{\rho} \ge \frac{1}{2+\epsilon}\overline{\rho^*}$, where $\overline{\rho^*}$ is the density of IDS. This indicates that our approximation algorithm can approximate DS and, consequently, can also approximate IDS.

6.1 Existing approximation algorithms

Since the density of the DS is greater than or equal to that of the IDS, existing approximation algorithms for the DS can also be used to approximate the IDS. Below, we introduce the key ideas of the state-of-the-art approximation algorithms SingleCore [25], AllCore [31], and CP-Approx [30].

Definition 6. **((α,β)-core)** *Given a graph G, the (α,β)-core of G, denoted by $C_{\alpha,\beta}$, is defined as the maximal subgraph $G[C_{\alpha,\beta}^O, C_{\alpha,\beta}^I]$, such that in $G[C_{\alpha,\beta}^O, C_{\alpha,\beta}^I]$, for all $x \in C_{\alpha,\beta}^O$, $d_x^O \ge \alpha$, and for all $y \in C_{\alpha,\beta}^I$, $d_y^I \ge \beta$.*

Theorem 13. *[31] Given a graph G, we have $\rho(C_{\alpha,\beta}) \ge \sqrt{\alpha\beta}$.*

$C_{\alpha,\beta}$ also represents a dense subgraph, but it is not a density-based model, so its density guarantee is not as strong as the $2\sqrt{\alpha\beta}$ guarantee of $F_{\alpha,\beta}$ and $D_{\alpha,\beta}$ (Lemma 1). However, since computing a core is less expensive, many core-based approximation algorithms have been developed, such as SingleCore and AllCore.

The core-based SingleCore **algorithm [25].** The SingleCore algorithm (as shown in Algorithm 4) uses a peeling technique, where in each step, the node with the smallest indegree or outdegree in the current graph is selected and removed (lines 3-7). This process is repeated until the graph becomes empty (line 2). After removing each node, the algorithm records the density of the current graph (line 8) and finally selects the graph with the highest density during the peeling process as the output (line 9). Core computation algorithms typically use a similar peeling method. During the peeling process in SingleCore, $G'[S,T]$ sequentially becomes $C_{0,0}$, $C_{1,1}$, $C_{2,2}, \ldots$, meaning that one peeling process can compute all non-empty $C_{k,k}$.

Since peeling the graph to an empty graph takes $O(|E|)$ time complexity, the overall time complexity of SingleCore is also $O(|E|)$ [25]. While SingleCore has a linear-time complexity, its approximation ratio is not very good. Previous works [25, 31] have proved that the approximation ratio of SingleCore is worse than 2. However, to the best of our knowledge, there is no work that has determined the exact approximation ratio of SingleCore. We are the first to prove that the approximation ratio of SingleCore is $(\sqrt{c^*} + \frac{1}{\sqrt{c^*}})$. Next, we present our results.

Lemma 3. *Given a graph G, if $F_{\alpha,\beta}$ or $D_{\alpha,\beta}$ is non-empty, then $C_{\lceil\alpha\rceil,\lceil\beta\rceil}$ is also non-empty.*

Proof. If $F_{\alpha,\beta} = G[F^O, F^I]$ is non-empty, then by the definition of $F_{\alpha,\beta}$, for any $x \in F^O$, we have $|E(F^O,F^I)| - |E(F^O \setminus \{x\}, F^I)| \ge \alpha$, which implies $d_x^O(F_{\alpha,\beta}) \ge \alpha \Rightarrow d_x^O(F_{\alpha,\beta}) \ge \lceil\alpha\rceil$. Similarly, for any $y \in F^I$, we have $d_y^I(F_{\alpha,\beta}) \ge \lceil\beta\rceil$. By the definition of $C_{\lceil\alpha\rceil,\lceil\beta\rceil}$, we know that $F_{\alpha,\beta} \subseteq C_{\lceil\alpha\rceil,\lceil\beta\rceil}$, so $C_{\lceil\alpha\rceil,\lceil\beta\rceil}$ is non-empty. □

Based on Lemma 3, we prove the approximation ratio of SingleCore in Theorem 14.

Theorem 14. *The output $\widetilde{D} = G[\widetilde{D}^O, \widetilde{D}^I]$ of the algorithm* SingleCore *satisfies $\rho(\widetilde{D}^O, \widetilde{D}^I) \ge \frac{1}{\sqrt{c^*}+\frac{1}{\sqrt{c^*}}}\rho^* \ge \frac{1}{\sqrt{c^*}+\frac{1}{\sqrt{c^*}}}\overline{\rho}^*$, i.e.,* SingleCore *is a $(\sqrt{c^*} + \frac{1}{\sqrt{c^*}})$-approximation algorithm.*

Proof. According to Theorem 11, since the DS $F_{\alpha^*,\beta^*} \neq \emptyset$, it follows that $F_{t(\alpha^*,\beta^*/c^*),(1-t)(\alpha^* c^* + \beta^*)} \neq \emptyset$. Substituting $\alpha^* = \frac{\rho^*}{2\sqrt{c^*}}$ and $\beta^* = \frac{\rho^*\sqrt{c^*}}{2}$ (based on Theorem 3), we get $F_{t\frac{\rho^*}{\sqrt{c^*}},(1-t)\rho^*\sqrt{c^*}} \neq \emptyset$. Setting $t = \frac{c}{1+c}$, we obtain $F_{\frac{\rho^*\sqrt{c^*}}{1+c},\frac{\rho^*\sqrt{c^*}}{1+c}} \neq \emptyset$. According to Lemma 3, we know that $C_{\lceil\frac{\rho^*\sqrt{c^*}}{1+c}\rceil,\lceil\frac{\rho^*\sqrt{c^*}}{1+c}\rceil} \neq \emptyset$. Let $G[C^O,C^I] = C_{\lceil\frac{\rho^*\sqrt{c^*}}{1+c}\rceil,\lceil\frac{\rho^*\sqrt{c^*}}{1+c}\rceil}$. Since SingleCore computes all non-empty $C_{k,k}$ for integer k, its output $\widetilde{D} = G[\widetilde{D}^O, \widetilde{D}^I]$ satisfies $\rho(\widetilde{D}^O, \widetilde{D}^I) \ge \rho(C^O,C^I) \ge \sqrt{\lceil\frac{\rho^*\sqrt{c^*}}{1+c}\rceil\lceil\frac{\rho^*\sqrt{c^*}}{1+c}\rceil} = \lceil\frac{\rho^*}{\sqrt{c^*}+\frac{1}{\sqrt{c^*}}}\rceil \ge \frac{\rho^*}{\sqrt{c^*}+\frac{1}{\sqrt{c^*}}} \ge \frac{\overline{\rho}^*}{\sqrt{c^*}+\frac{1}{\sqrt{c^*}}}$. □

According to Theorem 14, although SingleCore is highly efficient with its linear-time complexity, its approximation ratio is $(\sqrt{c^*}+\frac{1}{\sqrt{c^*}})$. When the c^* of the graph is large (e.g., 10^5), the approximation performance of SingleCore can be very bad, as confirmed by our experiments.

The core-based AllCore **algorithm [31].** Unlike SingleCore, which uses a single peeling process, AllCore [31] uses peeling for $O(\sqrt{|E|})$ times to compute all non-empty $C_{\alpha,\beta}$ and then selects the non-empty $C_{\alpha,\beta}$ that maximizes $\alpha\beta$ as the approximate subgraph. Since AllCore performs $O(\sqrt{|E|})$ peeling iterations, with each peeling taking $O(|E|)$, its time complexity is $O(|E|^{1.5})$ [31]. Although this complexity is much higher than the linear-time complexity $O(|E|)$, AllCore guarantees an approximation ratio of 2. However,

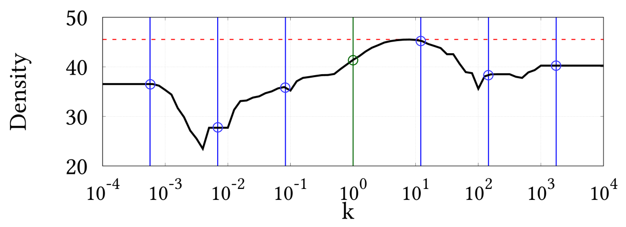


Figure 5: The density of core when varying k on dataset IM **($|V| = 896K$, $|E| = 3.8M$, $\rho^* = 45.52$).**

in our experiments, AllCore consumes too much time, and on all of the datasets, it is even slower than our exact algorithm GetIDS++.

The convex-programming-based CP-Approx **algorithm [30].** The $(1+\epsilon)$-approximation algorithm CP-Approx [30] relaxes the convex-programming-based algorithm CP-Exact presented for computing the exact DS, sacrificing density to save runtime. The time complexity of CP-Approx is $O(t_{FW} \cdot \log_{1+\epsilon}|V|)$, where t_{FW} is the time taken for several time-consuming convex programming and maximum flow computation, leading to inefficiency. As confirmed by our experiments, CP-Approx struggles to achieve both good approximation quality and low runtime simultaneously.

In summary, existing approximation algorithms either have poor practical approximation qualities (SingleCore and CP-Approx algorithms with large ϵ) or are very time-consuming (AllCore and CP-Approx algorithms with small ϵ). Therefore, it is necessary to design a novel approximation algorithm that achieves both good approximation performance and less runtime.

6.2 A novel $(2+\epsilon)$-approximation algorithm

SingleCore [25] performs a single peeling operation, which is fast but does not provide a good approximation guarantee. On the other hand, AllCore [31] performs $O(\sqrt{|E|})$ peeling operations, offering a better approximation guarantee but being slower. Our proposed algorithm, MultiCore, performs a small amount of peeling operations, achieving a balance between good approximation guarantees and fast runtime. Besides, MultiCore uses the error parameter ϵ to trade off between approximation accuracy and running time.

Intuitively, the peeling rationale behind MultiCore is inspired by the peeling approach used in SingleCore, which only finds the core subgraph where $\alpha = \beta$. However, MultiCore introduces a novel strategy by selecting multiple values of $k \in (0, +\infty)$ and searching for core subgraphs where $\alpha \approx k \cdot \beta$. Among all the core subgraphs found, it selects the one with the highest density as the approximate IDS. This non-trivial approach significantly improves the approximation ratio from $\sqrt{c^*} + \frac{1}{\sqrt{c^*}}$ in SingleCore to $2+\epsilon$. Next, we use an example to illustrate this rationale.

Example 4. MultiCore *computes all (α, β)-cores where $\alpha \approx k \cdot \beta$ in a single peeling process (the* Peeling *function in Algorithm 5). The black line in Figure 5 shows how the choice of k affects the density of the core obtained during the peeling process on the* IM *dataset ($|V| = 896K$, $|E| = 3.8M$, $\rho^* = 45.52$). (1) For* SingleCore*, it only invokes once peeling and computes the $C_{\alpha,\beta}$ where $\alpha = \beta$ (i.e., $k = 1$), so it can only consider the core on the green line. Consequently, it finds a subgraph with a density of only 41.26. (2)* AllCore *invokes the peeling function $O(\sqrt{|E|})$ times to compute all possible core subgraphs, i.e., all core subgraphs on the black line, which is very time-consuming. (3) In contrast,* MultiCore *selects a series of k values, shown by the blue and green lines in the figure. This strategy involves only a small number of peelings while still finding a core with high density. For example,* MultiCore *chooses $k = 12.05$ and obtains a core with a density of* 45.20 $\approx \rho^* = 45.52$*. Therefore,* MultiCore *gets a core subgraph with near-optimal density.*

Next, we give a theorem as the theoretical foundation of MultiCore. This theorem demonstrates that $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil}$ can serve as a $\frac{\sqrt{kc^*}}{k+c^*}$-approximate subgraph. Subsequently, we will explain how to compute $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil}$ and how to vary k to ensure obtaining a $\frac{1}{2+\epsilon}$-approximate subgraph.

Theorem 15. *Given a graph G and a number $k \in (0, +\infty)$, $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil}$ is non-empty and its density $\rho \ge \frac{\sqrt{kc^*}}{k+c^*}\rho^* \ge \frac{\sqrt{kc^*}}{k+c^*}\widetilde{\rho}^*$.*

Proof. In the proof of Theorem 14, we obtained that $F_{t\frac{\rho^*}{\sqrt{c^*}},(1-t)\rho^*\sqrt{c^*}} \neq \emptyset$. Let $t = \frac{c^*}{k+c^*}$, then we have $F_{\frac{\rho^*\sqrt{c^*}}{k+c^*}, \frac{k\rho^*\sqrt{c^*}}{k+c^*}} \neq \emptyset$. According to Lemma 3, we have $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil} \neq \emptyset$. Moreover, according to Theorem 13, we know that its density $\rho \ge \sqrt{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil} \ge \frac{\sqrt{kc^*}}{k+c^*}\rho^* \ge \frac{\sqrt{kc^*}}{k+c^*}\widetilde{\rho}^*$. □

In the MultiCore algorithm, we present a method to obtain $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil}$, which allows us to obtain a $\frac{\sqrt{kc^*}}{k+c^*}$-approximate subgraph. Let $\mathscr{E} = \epsilon^2 + 4\epsilon + 2 + \sqrt{(\epsilon^2 + 4\epsilon + 2)^2 - 4}$, and the solution to the inequality $\frac{\sqrt{kc^*}}{k+c^*} \ge \frac{1}{2+\epsilon}$ is $c^* \in [\frac{2k}{\mathscr{E}}, \frac{k\mathscr{E}}{2}]$. This means that if c^* happens to fall within the range $[\frac{2k}{\mathscr{E}}, \frac{k\mathscr{E}}{2}]$, we can obtain a $(2+\epsilon)$-approximate subgraph $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil}$. Based on the definition of c^*, we know that $c^* \in [\frac{1}{|V|}, |V|]$, so we need to select multiple values of k and obtain multiple intervals $[\frac{2k}{\mathscr{E}}, \frac{k\mathscr{E}}{2}]$ such that the union of these intervals completely covers $[\frac{1}{|V|}, |V|]$. Specifically, let N be a positive and even integer, and let $k = (\mathscr{E}/2)^i$, then by choosing $i = -N, -N+2, \ldots, -2, 0, 2, \ldots, N-2, N$, we can cover the interval $[(\frac{\mathscr{E}}{2})^{-N-1}, (\frac{\mathscr{E}}{2})^{-N+1}] \cup [(\frac{\mathscr{E}}{2})^{-N+1}, (\frac{\mathscr{E}}{2})^{-N+3}] \cup \cdots \cup [(\frac{\mathscr{E}}{2})^{N-3}, (\frac{\mathscr{E}}{2})^{N-1}] \cup [(\frac{\mathscr{E}}{2})^{N-1}, (\frac{\mathscr{E}}{2})^{N+1}] = [(\frac{\mathscr{E}}{2})^{-(N+1)}, (\frac{\mathscr{E}}{2})^{N+1}]$. Solving the inequalities $(\frac{\mathscr{E}}{2})^{-(N+1)} \le \frac{1}{|V|}$ and $(\frac{\mathscr{E}}{2})^{N+1} \ge |V|$, we get $N \ge \log_{\mathscr{E}/2}|V| - 1$, i.e., N should be set as the minimum even integer such that $N \ge \log_{\mathscr{E}/2}|V| - 1$. Therefore, we need a total of $-N, -N+2, \ldots, N-2, N$, i.e., $N+1$ values of k, which means we need to compute $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil}$ a total of $N+1 \approx \lceil \log_{\mathscr{E}/2}|V| \rceil$ times. By selecting the subgraph with the maximum density among these $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*}\rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*}\rceil}$, we can obtain a $(2+\epsilon)$-approximate subgraph.

Using the above rationale, we propose the algorithm MultiCore, as shown in Algorithm 5. First, MultiCore initializes $\widetilde{\rho^*}$ and $\widetilde{D}$, and computes the values of $\mathscr{E}$ and N (lines 1-2). Then, MultiCore sets

```
Algorithm 5: MultiCore($G, \epsilon$)
Input: A directed graph $G = (V, E)$, the error parameter $\epsilon > 0$.
Output: A $(2+\epsilon)$-approximate IDS $\widetilde{D}$.
1  $\widetilde{\rho^*} \leftarrow \rho(V, V)$, $\widetilde{D} \leftarrow G$, $\mathscr{E} \leftarrow \epsilon^2 + 4\epsilon + 2 + \sqrt{(\epsilon^2+4\epsilon+2)^2 - 4}$;
2  Let $N$ be the minimum even integer such that $N \geq \log_{\mathscr{E}/2}|V| - 1$;
3  for $i = -N, -N+2, \ldots, N-2, N$ do Peeling($(\frac{\mathscr{E}}{2})^i$) ;
4  return $\widetilde{D}$;
5  Function Peeling($k$)
6      $\alpha \leftarrow 0, \beta \leftarrow 0, G'[S,T] \leftarrow G$;
7      while $G'$ is not empty do
8          $x = \mathrm{argmin}_{x \in S}\, d_x^O(G')$, $y = \mathrm{argmin}_{y \in T}\, d_y^I(G')$;
9          if $d_x^O(G) < \alpha$ then
10             $G'[S,T] \leftarrow G'[S \setminus \{x\}, T]$, $S \leftarrow S \setminus \{x\}$;
11         else if $d_y^I(G) < \beta$ then
12             $G'[S,T] \leftarrow G'[S, T \setminus \{y\}]$, $T \leftarrow T \setminus \{y\}$;
13         else
               // Now, we have $G'[S,T] = C_{\alpha,\beta}$;
14             if $\beta > k \cdot \alpha$ then $\alpha \leftarrow \alpha + 1$;
15             else $\beta \leftarrow \beta + 1$;
16         if $\rho(S,T) > \widetilde{\rho^*}$ then $\widetilde{\rho^*} \leftarrow \rho(S,T)$, $\widetilde{D} \leftarrow G'[S,T]$;
```

$k = \left(\frac{\mathscr{E}}{2}\right)^i$ and calls the peeling function with the parameter k (line 3). The Peeling function gradually removes nodes from the original graph G until an empty graph is obtained. During the removal process, two values α and β are tracked (line 6), representing the thresholds such that all nodes in S with outdegree less than α and nodes in T with indegree less than β should be removed (lines 8-12). For each node removed, the algorithm records the current density (line 16). Specifically, when every node in S of the current graph $G'[S,T]$ has an outdegree not less than α and every node in T has an indegree not less than β, we have obtained $C_{\alpha,\beta}$ (line 13). At this point, either α or β needs to be increased to continue peeling nodes (lines 14-15). Finally, when α and β become large enough, all nodes will be peeled off, and $G'[S,T]$ will become an empty graph. For each enumerated $k = \left(\frac{\mathscr{E}}{2}\right)^i$, the peeling process is repeated. The subgraph with the highest density encountered during this process is recorded, and this subgraph is output as the approximate IDS (line 4). Next, we prove the correctness of MultiCore.

Theorem 16. *The* MultiCore *can output a subgraph $\widetilde{D}$ whose density $\rho \geq \frac{1}{2+\epsilon}\rho^* \geq \frac{1}{2+\epsilon}\overline{\rho}^*$, and the time complexity and space complexity of* MultiCore *is* $O(|E| \cdot \log_{1+\epsilon}|V|)$ *and* $O(|E|)$*, respectively.*

Proof. For correctness, we first prove that during the peeling process in Peeling(k), the way α and β increase (lines 14-15) ensures that at an iteration of the while loop, $G'[S,T]$ will equal $C_{\lceil \frac{\rho^*\sqrt{c^*}}{k+c^*} \rceil, \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*} \rceil}$. This means that after running Peeling(k), we must have $\widetilde{\rho^*} \geq \frac{\sqrt{kc^*}}{k+c^*}\rho^*$. We will only prove the case where $k \geq 1$, and the case where $k < 1$ can be proven similarly. Let $\alpha' = \lceil \frac{\rho^*\sqrt{c^*}}{k+c^*} \rceil$ and $\beta' = \lceil \frac{k\rho^*\sqrt{c^*}}{k+c^*} \rceil$. It can be directly obtained that $\beta' \in [k(\alpha'-1)+1, k\alpha']$. In the Peeling function, when α increases from $\alpha'-1$ to α', let β at this point be β_1. Based on the algorithm's condition in line 14, we can conclude that $\beta_1 \geq k\alpha'$. When α increases from $\alpha'-1$ to α', let β at this point be β_2. Since $k \geq 1$, when $\alpha = \alpha'-1$ and $\beta = \beta_1 - 1$, it is β that increases, i.e., $\beta_2 - 1 < k(\alpha'-1) \Rightarrow \beta_2 < k(\alpha-1)+1$. During this period, for $\beta = \beta_2, \ldots, \beta_1$, the algorithm obtains $C_{\alpha',\beta}$. Notice that the lower bound of β_1, $k\alpha'$, and the upper bound of β_2, $k(\alpha-1)+1$, exactly cover the upper and lower bounds of β', $[k(\alpha'-1)+1, k\alpha']$. Therefore, the algorithm must have obtained $C_{\alpha',\beta'}$ during this process, with density $\rho \geq \frac{\sqrt{kc^*}}{k+c^*}\rho^*$. Hence, after running Peeling(k), we must have $\widetilde{\rho^*} \geq \frac{\sqrt{kc^*}}{k+c^*}\rho^*$.

From the previous discussion, when Peeling($(\frac{\mathscr{E}}{2})^i$) is called and $c^* \in [(\frac{\mathscr{E}}{2})^{i-1}, (\frac{\mathscr{E}}{2})^{i+1}]$, we can ensure that $\widetilde{\rho^*} \geq \frac{1}{2+\epsilon}\rho^*$. Additionally, because the value of N we selected guarantees that $(\frac{\mathscr{E}}{2})^{i-1} \leq \frac{1}{|V|}$ and $(\frac{\mathscr{E}}{2})^{i+1} \geq |V|$, no matter what value c^* takes within $[\frac{1}{|V|}, |V|]$, we can always ensure that $\widetilde{\rho^*} \geq \frac{1}{2+\epsilon}\rho^* \geq \frac{1}{2+\epsilon}\overline{\rho}^*$.

For the time complexity, using the same implementation as SingleCore [25], one execution of the Peeling algorithm can be completed in $O(|E|)$ time. Since the algorithm calls the Peeling function $O(\log_{\mathscr{E}/2}|V|)$ times, the total time complexity is $O(|E|\log_{\mathscr{E}/2}|V|)$, where $\mathscr{E}/2 \leq 1+\epsilon$. Therefore, the time complexity is $O(|E|\log_{1+\epsilon}|V|)$. As for the space complexity, it is proportional to the graph size, which is $O(|E|)$. □

Given $\epsilon > 0$, MultiCore requires only a small number of $O(|E|)$-time Peeling function invocations, making it highly efficient. For example, in our experiments, on the large TW graph with 40.1 million nodes and 1.5 billion edges, setting $\epsilon = 0.5$ requires only 13 peeling operations to obtain the approximate subgraph. Therefore, MultiCore can efficiently finish the computation in near-linear time. In contrast, the time complexity of the CP-Approx algorithm is as high as $O(t_{FW} \cdot \log_{1+\epsilon}|V|)$, where t_{FW} denotes the time taken for several time-consuming convex programming and maximum flow computations. Although the approximation guarantee $(1+\epsilon)$ of CP-Approx is tighter than the $(2+\epsilon)$ guarantee of MultiCore, in our experiments on real-world graphs, the proposed MultiCore algorithm often produces subgraphs with higher (i.e., better) density in significantly less runtime compared to CP-Approx.

7 EXPERIMENTS

Algorithms. For exact algorithms, we implement our proposed algorithms GetIDS (Algorithm 2) and GetIDS++ (Algorithm 3). We compare our algorithms with the flow-based algorithm DC-Exact [31] and the SOTA algorithm CP-Exact [30] for computing the densest subgraph. For approximation algorithms, we evaluate the $(\sqrt{c^*} + \frac{1}{\sqrt{c^*}})$-approximation algorithm SingleCore [25] (Algorithm 4) and our new $(2+\epsilon)$-approximation algorithm MultiCore (Algorithm 5) with the SOTA 2-approximation algorithm AllCore [31] and $(1+\epsilon)$-approximation algorithm CP-Approx [30]. All algorithms are implemented in C++ with O3 optimization. Our experiments are conducted on a Linux system PC with a 2.2GHz AMD 3990X 64-Core CPU and 256GB of memory. We set the upper limit of runtime to 10^6 seconds.

Table 2: Statistics of datasets.

For the datasets HE, OR, and TW, the runtime of computing ρ^* exceeds time limit 10^6 seconds, so we use the density of the IDS to approximate ρ^*.

Dataset	Category	$\|V\|$	$\|E\|$	ρ^*	p
IM	affiliation	896.3K	3.8M	45.5170	20
HE	citation	28.1K	4.6M	≈ 776.4999	371
TR	lexical	1.2M	83.6M	2,086.3359	440
WI	hyperlink	4.8M	113.1M	2,170.7350	419
IN	hyperlink	7.4M	194.1M	6,924.0780	3,462
DE	interaction	33.8M	301.2M	2,345.1849	970
OR	affiliation	8.7M	327.0M	≈ 2079.0586	438
UK	hyperlink	39.4M	936.4M	7,642.1016	454
IT	hyperlink	41.3M	1.2B	4,473.7718	1,834
TW	social	40.1M	1.4B	≈ 4581.4683	1,427

Datasets. Following the concept of the "auxiliary bipartite graph" introduced in [30], we can equivalently transform directed graphs and bipartite graphs. This allows our solutions to be directly applied to bipartite graphs as well. Consequently, in addition to the six directed graphs HepPh (HE), WikiFr (WI), IndoChina (IN), UKAll (UK), ITAll (IT), Twitter (TW), we also select four bipartite graphs IMDB (IM), Trec (TR), Delicious (DE), Orkut (OR) to enhance the comprehensiveness of our experimental evaluations. Detailed dataset information is shown in Table 2, which can be obtained from the Network Repository [34].

7.1 Results of exact algorithms

Exp-1: Runtime of various exact algorithms. The results of this experiment are shown in Figure 6, and we have the following observations: (1) The algorithms DC-Exact and GetIDS++ both are flow-based algorithms. However, DC-Exact tackles the more challenging DS problem, which in the worst case requires up to $O(|V|^2 \log |V|)$ network flow computations [31]. In contrast, GetIDS++ addresses the simpler IDS problem, requiring only $O(p \cdot \log |V|)$ network flow computations in the worst case. Consequently, the runtime of GetIDS++ is significantly lower than that of DC-Exact, highlighting the computational efficiency of the IDS problem. (2) The runtime of the CP-Exact algorithm exhibits an unstable trend. For example, its runtime on the smaller million-edge dataset HE exceeded 10^6 seconds, whereas on the billion-edge dataset IT, it achieves a faster runtime of 43,044 seconds, indicating the instability of convex-programming-based algorithm CP-Exact for computing the densest subgraph. (3) In contrast to CP-Exact, the runtime of our GetIDS++ is more stable, with runtime on all datasets under 15,000 seconds. Moreover, GetIDS++ is significantly faster than DC-Exact (up to 3,589x), CP-Exact (up to 71,327x) and GetIDS (up to 158x).

For example, on the dataset HE, the runtimes for DC-Exact, CP-Exact, GetIDS, and GetIDS++ are 16,945 seconds, over 1,000,000 seconds, 823.45 seconds, and 14.02 seconds, respectively, indicating a speedup of 1,209x, 71,327x and 59x for GetIDS++. Similarly, on the dataset OR, the runtimes for DC-Exact, CP-Exact, GetIDS, and GetIDS++ are more than 1,000,000 seconds, more than 1,000,000 seconds, 279,438 seconds, and 1,765 seconds, respectively, indicating a speedup of 567x and 128x for GetIDS++. These results demonstrate the significant efficiency of the proposed GetIDS++ algorithm.

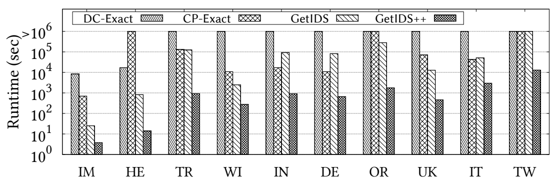
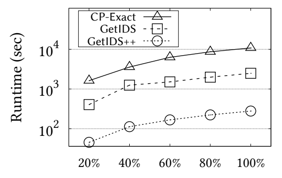
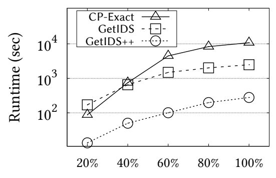


Figure 6: Runtime of various exact algorithms.



(a) Vary $|E|$.



(b) Vary $|V|$.

Figure 7: Scalability testing on dataset WI.

Table 3: Comparison between GetIDS and GetIDS++.

n_b and n_f are the number of binary search and flow computations invoked in the corresponding algorithm, respectively.

Algorithm		HE	TR	WI	OR	UK
GetIDS	n_b	743	881	839	877	909
	n_f	8,898	16,574	17,055	16,063	18,928
GetIDS++	n_b	6	7	8	5	7
	n_f	471	328	694	325	213

Exp-2: Scalability testing. To evaluate the scalability of our solutions, we first generate eight subgraphs by randomly sampling 20%, 40%, 60%, and 80% vertices or edges from the WI dataset (similar results can be obtained on other datasets). The results are shown in Figure 7. As can be seen, the runtime of our proposed algorithm GetIDS++ grows relatively slowly as the data scale increases, whereas the runtime of GetIDS and CP-Exact increases rapidly. In particular, our GetIDS++ is 1-2 orders of magnitude faster than both GetIDS and CP-Exact in all cases. These results demonstrate the high scalability of our GetIDS++ algorithm.

Exp-3: Comparison between GetIDS and GetIDS++. We select five datasets (similar results can be obtained on other datasets) and record the number of binary searches and maximum flow computations used by GetIDS and GetIDS++. The results are shown in Table 3. Since GetIDS performs binary searches for every enumerated α and β, the number of binary searches and flow computations is high. For example, on the UK dataset, the number of binary search and flow computations reaches as high as 909 and 18,928, respectively. In contrast, GetIDS++ prunes most of the enumerated α and β values, performing at most 7 binary searches and 213 flow computations. These results demonstrate the effectiveness of the proposed pruning techniques integrated into GetIDS++, making the computation of the IDS highly efficient.

Exp-4: Comparison between the DS and IDS. As shown in Table 4, the DS and IDS exhibit a high degree of similarity. For instance, in the datasets WI, IN, UK, and IT, the subgraphs obtained from both DS and IDS are identical. In the remaining datasets, the density of these subgraphs differs by at most 0.0271, and their similarity

Table 4: Comparison between the DS and IDS.

ρ^* and $\overline{\rho}^*$ denote the density of the DS and the IDS, respectively. The "similarity" of two subgraphs $G[S_1, T_1]$ and $G[S_2, T_2]$ is defined as $(|S_1 \cap S_2| + |T_1 \cap T_2|)/(|S_1 \cup S_2| + |T_1 \cup T_2|)$.

Dataset	ρ^*	$\overline{\rho}^*$	Similarity
IM	45.5170	45.4899 **(-0.0271)**	6376/6796 = 0.938
TR	2,086.3359	2,086.3119 **(-0.0240)**	327972/328769 = 0.998
WI	2,170.7350	2,170.7350 **(-0.0000)**	793311/793311 = 1
IN	6,924.0780	6,924.0780 **(-0.0000)**	13889/13889 = 1
DE	2,345.1849	2,345.1814 **(-0.0035)**	806/808 = 0.998
UK	7,642.1016	7,642.1016 **(-0.0000)**	1769782/1769782 = 1
IT	4,473.7718	4,473.7718 **(-0.0000)**	1319477/1319477 = 1

Table 5: Comparison between GetIDS++ and CS-Community.

Dataset	Algorithm	Density	Runtime (sec)
HE	GetIDS++	776.50	14.02
	CS-Community	260.16	687.36
Epinions	GetIDS++	91.92	0.73
	CS-Community	47.77	98.37

is at least 0.938. These findings highlight the strong correlation between DS and IDS, suggesting that the IDS is not only easier to compute but also produces results comparable to those of the DS.

Exp-5: Comparison with other density-based model. In this experiment, we compare the IDS model with the dense subgraph model proposed in [12], which is computed using the CS-Community algorithm. Although CS-Community is designed for undirected graphs, [12] suggests that it can also handle directed graphs by simply disregarding their directionality. The algorithm outputs multiple communities, and we select the one with the highest density for comparison. We evaluate the performance of the GetIDS++ algorithm and CS-Community on the HE and Epinions (details in our case study) datasets, as summarized in Table 5 (only two datasets are evaluated because CS-Community fails to complete computations on other datasets within 10^4 seconds). The results demonstrate that GetIDS++ identifies IDS subgraphs with significantly higher densities and much faster runtime compared to the communities found by CS-Community. This is because CS-Community handles directed graphs by naively ignoring directionality. While simple, this approach fails to capture the directional structure of directed graphs, resulting in subgraphs with lower densities. These findings highlight the limitations of applying undirected graph models to directed graphs and underscore the importance of developing dense subgraph models specifically tailored for directed graphs.

7.2 Results of approximation algorithms

Exp-6: Runtime and approximation ratio of different algorithms. For our $(2+\epsilon)$-approximation algorithm MultiCore, we set $\epsilon = 0.5$ and denoted as MultiCore-0.5. For the $(1+\epsilon)$-approximation algorithm CP-Approx, we set ϵ to 1.5, 0.5, and 0.1, denoted as CP-Approx-1.5, CP-Approx-0.5, and CP-Approx-0.1, respectively. The results are shown in Figure 8 and Figure 9.

In this experiment, we have the following observations. (1) The Single algorithm, while the fastest among all tested algorithms, exhibits a significantly poor approximation ratio. For example, it has

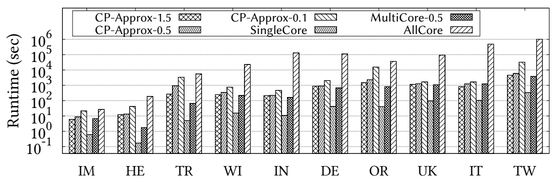


Figure 8: Runtime of approximation algorithms.

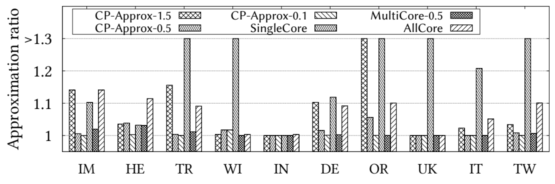
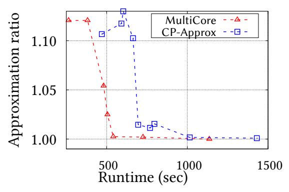


Figure 9: Approximation ratio of approximation algorithms.

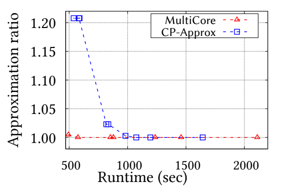
an approximation ratio of 12.9 on the UK dataset. (2) The AllCore algorithm exhibits a considerably high runtime, reaching up to 10^6 seconds on the TW dataset, and does not provide the best approximation ratio, exceeding 1.05 on seven of the ten datasets. (3) The algorithm CP-Approx-1.5 shows a significant instability in its approximation ratio, peaking at 1.48 on the OR dataset, and does not gain any runtime advantage over MultiCore-0.5. (4) Across all datasets, MultiCore-0.5 runs faster than CP-Approx-0.5, and on 8 out of 10 datasets, the approximation ratio of MultiCore-0.5 is better than or equal to that of CP-Approx-0.5. For example, on the HE dataset, MultiCore-0.5 achieves an approximation ratio of 1.031, which is better than CP-Approx-0.5's ratio of 1.038, while its runtime is only 1.74 seconds, making it 7.6x faster than CP-Approx-0.5, which takes 13.25 seconds. (5) Both CP-Approx-0.1 and MultiCore-0.5 achieve excellent approximation ratios, remaining consistently below 1.05 across all datasets. Notably, MultiCore-0.5 is considerably faster than CP-Approx-0.1 in all cases. For example, in the OR dataset, both algorithms achieve a commendable approximation ratio of 1.0005, but their runtimes differ significantly: 15,430 seconds for CP-Approx-0.1 versus just 819 seconds for MultiCore-0.5, yielding an impressive 18.8x speedup.

Overall, compared to other approximation algorithms, our proposed MultiCore algorithm strikes an exceptional balance between excellent approximation performance and rapid runtime, aligning well with our theoretical analysis in Section 6.

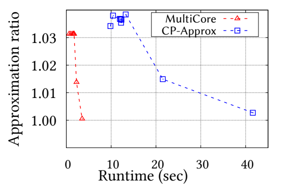
Exp-7: Comparison between MultiCore and CP-Approx with varying ϵ. We conduct experiments on the DE and IT datasets (similar results can be obtained on other datasets). For the MultiCore algorithm, we set ϵ to 4, 2, 1, 0.75, 0.5, 0.25, 0.1. For the CP-Approx algorithm, we set ϵ to 5, 3, 2, 1.5, 1, 0.75, 0.5, 0.25, 0.1. The results are shown in Figure 10. Compared to CP-Approx, MultiCore produces a subgraph with higher density in a shorter runtime on both datasets no matter how ϵ varies. For example, on the DE dataset, MultiCore with $\epsilon = 0.5$ finds a subgraph with an approximation ratio as low as 1.0025 in 541 seconds. In contrast, CP-Approx with $\epsilon = 2$ takes a longer runtime of 603 seconds and only achieves an approximation



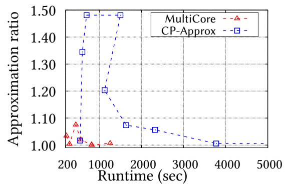
(a) Results on DE.



(b) Results on IT.



(c) Results on HE.



(d) Results on OR.

Figure 10: Comparison between MultiCore **and** CP-Approx**.**

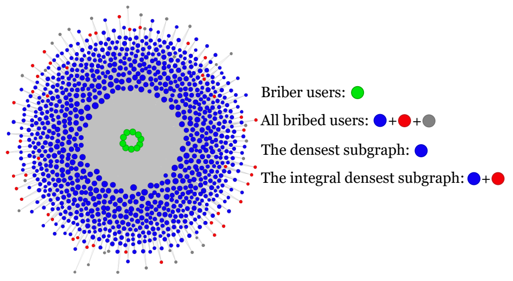


Figure 11: Case Studies on Epinions **social network.**

ratio of 1.1299. On the IT dataset, MultiCore with $\epsilon = 2$ takes just 575 seconds to find a subgraph with an approximation ratio of 1, i.e., the exact IDS. In comparison, CP-Approx with $\epsilon = 5$ takes a longer runtime of 589 seconds, while achieving only an approximation ratio of 1.2078. On the HE dataset, the runtime of the MultiCore algorithm remains under 4 seconds for any ϵ setting, requiring only 3.53 seconds to produce a near-optimal subgraph with an approximation ratio of 1.0005. In contrast, the CP-Approx algorithm takes significantly longer runtime, requiring 41.60 seconds to obtain a subgraph with a worse approximation ratio of 1.0027. On the OR dataset, the MultiCore algorithm consistently maintains both efficient runtime and stable approximation ratios, whereas the runtime of the CP-Approx algorithm grows rapidly, and its approximation ratio becomes highly unstable, reaching a poor approximation ratio of 1.4811 when $\epsilon = 2$. These results demonstrate that our proposed MultiCore algorithm is efficient and provides high-quality approximations no matter how ϵ varies.

7.3 Case Study

This experiment utilizes the Epinions social network ($|V| = 75,879$, $|E| = 508,837$)[2] to evaluate the practical application of fraudulent detection using both the densest subgraph (DS) and the integral densest subgraph (IDS) models. Epinions is a consumer review network where each node represents a user, and each edge (u_1, u_2) represents user u_1 trusting u_2. The subgraph identified by the DS has $c^* = |S^*|/|T^*| = 1261/485$. However, some fraudulent users may resort to bribing others to gain trust, creating the illusion of being trusted by many. This behavior can lead to a new DS with an even higher c^* (e.g., $c^* = 1000/10$). To simulate this scenario, we introduced 10 briber users and 1,000 bribed users into the network, where the bribed users randomly trusted the briber users a total of 10,000 times—resulting in the addition of 10,000 edges from the bribed users to the briber users. In practical scenarios, bribed users may also trust many normal users to further conceal the fact that they have been bribed [21]. Consequently, we also randomly add 10,000 edges from the bribed users to normal users in the network.

After inserting fraudulent users, including briber and bribed users, into the Epinions dataset, we compute both the DS and the IDS. The results reveal that both methods exclusively identified fraudulent users, with no normal users present, successfully detecting all 10 briber users. The DS detects 925 bribed users, while the IDS detects 970 bribed users, suggesting that IDS is better than the DS for detecting fraudulent users. Figure 11 visualizes these findings, where all nodes represent fraudulent users. The green nodes correspond to the 10 bribe users, and the blue nodes represent the 925 bribed users identified by the DS. The IDS successfully detected all 925 bribed users identified by the DS, as well as an additional 45 bribed users represented by the red nodes, bringing the total to 970 detected bribed users. The gray nodes indicate bribed users that are not detected by both the DS and the IDS. This case study demonstrates that, compared to the DS, our proposed IDS is more effective in detecting fraudulent users, establishing its superiority in the practical application of fraud detection.

8 RELATED WORK

Densest subgraph search on directed graphs. For exact algorithms, Charikar [10] first proposed an LP-based algorithm that requires solving $O(|V|^2)$ linear programs. Later, Khuller and Saha [25] introduced a flow-based algorithm, which requires solving $O(|V|^2)$ maximum flow computations. Recently, Ma et al. [31] used a divide-and-conquer technique to accelerate this flow-based approach. Subsequently, Ma et al. [30] leveraged convex programming and various optimization techniques to design the SOTA CP-Exact algorithm. To improve efficiency, several approximation algorithms have also been proposed. For example, the core-based 2-approximation algorithms with time complexities of $O(|E|^{1.5})$ [31] and $O(|V|^2|E|)$ [10], as well as a core-based $(\sqrt{c^*} + \frac{1}{\sqrt{c^*}})$-approximation algorithm [25] with $O(|E|)$ time complexity, whose approximation ratio is proved in Section 6.1. The $(1+\epsilon)$-approximation algorithm CP-Approx was proposed [30]. However, all these algorithms have prohibitively runtime or poor approximation qualities, making them unsuitable for practical applications.

Densest subgraph search on undirected graphs. The densest subgraph problem on undirected graphs is defined as finding a subgraph that maximizes the ratio of the number of edges to the number of vertices in the subgraph [3, 4, 6, 9–11, 15, 16, 18, 20, 33, 38, 40, 43, 44]. A well-known algorithm is based on a parameterized Goldberg flow network, with a time complexity of $O(|E||V|\log|V|)$ [20]. To improve efficiency, Danisch et al. accelerated the process

[2]Dataset source: https://snap.stanford.edu/data/soc-Epinions1.html.

using convex-programming techniques [15]. Besides, several approximation algorithms have also been proposed, such as the 2-approximation algorithm [10] with a linear $O(|E|)$ time complexity, the $(1+\epsilon)$-approximation iterative algorithm [6, 11], the 2-approximation algorithm [16], and the $2(1+\epsilon)$-approximation algorithm [3]. Many variants of the densest subgraph problem have also been studied, such as the locally top-k densest subgraph [29, 33], anchored densest subgraph [42], densest k-subgraph [2, 7], and clique-density-based densest subgraph [16, 32, 37, 39, 41]. However, the densest subgraph problem on undirected graphs is fundamentally different from that on directed graphs, and the models and algorithms cannot be directly applied to directed graphs.

9 CONCLUSION

In this paper, we propose a novel IDS model that is simpler to compute than the DS while providing a near-optimal density guarantee. First, we define the fractional (α, β)-dense subgraph $F_{\alpha,\beta}$ and the integral (α, β)-dense subgraph $D_{\alpha,\beta}$. We prove that the non-empty $F_{\alpha,\beta}$ that maximizes $\alpha \cdot \beta$ is exactly the DS, and we define the non-empty $D_{\alpha,\beta}$ that maximizes $\alpha \cdot \beta$ as our IDS. Then, we propose a carefully-designed (α, β)-dense flow network, based on which we develop a flow-based algorithm, GetIDS, to search for the maximum value of $\alpha\beta$ with time complexity $O(p \cdot \log |V| \cdot |E|^{1.5})$. Building on our proposed emptiness and non-emptiness theorems, we develop several powerful pruning techniques and further propose an advanced GetIDS++ algorithm with the time complexity of $O(n_f \cdot |E|^{1.5})$, where n_f is the number of maximum flow computations and is often a small number in practice. To further improve efficiency, we propose a novel core-based $(2+\epsilon)$-approximation algorithm with a near-linear time complexity of $O(|E| \log_{1+\epsilon} |V|)$, which can extract near-densest subgraphs while maintaining short runtime practically. Finally, we conduct extensive experiments on 10 real-world graphs, with the results validating the effectiveness of our IDS model, and the high efficiency and scalability of our proposed algorithms.

REFERENCES

[1] Réka Albert, Hawoong Jeong, and Albert-László Barabási. 1999. Diameter of the world-wide web. *nature* 401, 6749 (1999), 130–131.

[2] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. 2000. Greedily Finding a Dense Subgraph. *J. Algorithms* 34, 2 (2000), 203–221. https://doi.org/10.1006/JAGM.1999.1062

[3] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest Subgraph in Streaming and MapReduce. *Proc. VLDB Endow.* 5, 5 (2012), 454–465.

[4] Oana Denisa Balalau, Francesco Bonchi, T.-H. Hubert Chan, Francesco Gullo, and Mauro Sozio. 2015. Finding Subgraphs with Maximum Total Density and Limited Overlap. In *WSDM*. 379–388.

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

[6] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos E. Tsourakakis, Di Wang, and Junxing Wang. 2020. Flowless: Extracting Densest Subgraphs Without Flow Computations. In *WWW*. 573–583.

[7] Nicolas Bourgeois, Aristotelis Giannakos, Giorgio Lucarelli, Ioannis Milis, and Vangelis Th Paschos. 2013. Exact and approximation algorithms for densest k-subgraph. In *WALCOM: Algorithms and Computation: 7th International Workshop, WALCOM 2013, Kharagpur, India, February 14-16, 2013. Proceedings 7*. Springer, 114–125.

[8] Andrea Capocci, Vito DP Servedio, Francesca Colaiori, Luciana S Buriol, Debora Donato, Stefano Leonardi, and Guido Caldarelli. 2006. Preferential attachment in the growth of social networks: The internet encyclopedia Wikipedia. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 74, 3 (2006), 036116.

[9] Lijun Chang and Miao Qiao. 2020. Deconstruct Densest Subgraphs. In *WWW*. 2747–2753.

[10] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Saarbrücken, Germany, September 5-8, 2000, Proceedings (Lecture Notes in Computer Science, Vol. 1913)*. Springer, 84–95.

[11] Chandra Chekuri, Kent Quanrud, and Manuel R. Torres. 2022. Densest Subgraph: Supermodularity, Iterative Peeling, and Flow. In *SODA*. SIAM, 1531–1555.

[12] Jie Chen and Yousef Saad. 2012. Dense Subgraph Extraction with Application to Community Detection. *IEEE Trans. Knowl. Data Eng.* 24, 7 (2012), 1216–1230. https://doi.org/10.1109/TKDE.2010.271

[13] 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.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition*. MIT Press.

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

[16] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V. S. Lakshmanan, and Xuemin Lin. 2019. Efficient Algorithms for Densest Subgraph Discovery. *Proc. VLDB Endow.* 12, 11 (2019), 1719–1732.

[17] Eugene Fratkin, Brian T. Naughton, Douglas L. Brutlag, and Serafim Batzoglou. 2006. MotifCut: regulatory motifs finding with maximum density subgraphs. In *ISMB*. 156–157.

[18] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. 2016. Top-k overlapping densest subgraphs. *Data Min. Knowl. Discov.* 30, 5 (2016), 1134–1165.

[19] Christos Giatsidis, Dimitrios M. Thilikos, and Michalis Vazirgiannis. 2013. D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowl. Inf. Syst.* 35, 2 (2013), 311–343.

[20] Andrew V Goldberg. 1984. *Finding a maximum density subgraph*. Technical Report. University of California Berkeley, Berkeley, CA, USA.

[21] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. ACM, 895–904.

[22] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. 2007. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*. 56–65.

[23] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. 2007. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis* (San Jose, California) *(WebKDD/SNA-KDD '07)*. Association for Computing Machinery, New York, NY, USA, 56–65.

[24] Samir Khuller and Barna Saha. 2009. On Finding Dense Subgraphs. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 5555)*. Springer, 597–608.

[25] Samir Khuller and Barna Saha. 2009. On Finding Dense Subgraphs. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 5555)*. Springer, 597–608.

[26] Haseong Kim. 2014. Modelling and analysis of gene regulatory networks based on the G-network. *Int. J. Adv. Intell. Paradigms* 6, 1 (2014), 28–51.

[27] Jon M. Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (1999), 604–632.

[28] Xuankun Liao, Qing Liu, Jiaxin Jiang, Xin Huang, Jianliang Xu, and Byron Choi. 2022. Distributed D-core Decomposition over Large Directed Graphs. *Proc. VLDB Endow.* 15, 8 (2022), 1546–1558.

[29] Chenhao Ma, Reynold Cheng, Laks V. S. Lakshmanan, and Xiaolin Han. 2022. Finding Locally Densest Subgraphs: A Convex Programming Approach. *Proc. VLDB Endow.* 15, 11 (2022), 2719–2732. https://doi.org/10.14778/3551793.3551826

[30] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, and Xiaolin Han. 2022. A Convex-Programming Approach for Efficient Directed Densest Subgraph Discovery. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 845–859.

[31] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2020. Efficient Algorithms for Densest Subgraph Discovery on Large Directed Graphs. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1051–1066.

[32] 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 *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams (Eds.). ACM,

815–824. https://doi.org/10.1145/2783258.2783385

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

[34] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. AAAI Press, 4292–4293.

[35] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. 2010. Dense Subgraphs with Restrictions and Applications to Gene Annotation Graphs. In *RECOMB (Lecture Notes in Computer Science, Vol. 6044)*. 456–472.

[36] Saurabh Sawlani and Junxing Wang. 2020. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*. ACM, 181–193.

[37] Bintao Sun, Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. 2020. KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs. *Proc. VLDB Endow.* 13, 10 (2020), 1628–1640. https://doi.org/10.14778/3401960.3401962

[38] Nikolaj Tatti and Aristides Gionis. 2013. Discovering Nested Communities. In *ECML PKDD (Lecture Notes in Computer Science, Vol. 8189)*. 32–47.

[39] Charalampos E. Tsourakakis. 2015. The K-clique Densest Subgraph Problem. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi (Eds.). ACM, 1122–1132. https://doi.org/10.1145/2736277.2741098

[40] Elena Valari, Maria Kontaki, and Apostolos N. Papadopoulos. 2012. Discovery of Top-k Dense Subgraphs in Dynamic Graph Collections. In *SSDBM (Lecture Notes in Computer Science, Vol. 7338)*. 213–230.

[41] Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongzhi Chen, and Guoren Wang. 2024. Efficient k-Clique Counting on Large Graphs: The Power of Color-Based Sampling Approaches. *IEEE Trans. Knowl. Data Eng.* 36, 4 (2024), 1518–1536. https://doi.org/10.1109/TKDE.2023.3314643

[42] Xiaowei Ye, Rong-Hua Li, Lei Liang, Zhizhen Liu, Longlong Lin, and Guoren Wang. 2024. Efficient and Effective Anchored Densest Subgraph Search: A Convex-programming based Approach. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, Ricardo Baeza-Yates and Francesco Bonchi (Eds.). ACM, 3907–3918. https://doi.org/10.1145/3637528.3671727

[43] Yalong Zhang, Ronghua Li, Qi Zhang, Hongchao Qin, Lu Qin, and Guoren Wang. 2024. Efficient Algorithms for Pseudoarboricity Computation in Large Static and Dynamic Graphs. *Proc. VLDB Endow.* 17, 11 (2024), 2722–2734.

[44] Yalong Zhang, Ronghua Li, Qi Zhang, Hongchao Qin, and Guoren Wang. 2024. Efficient Algorithms for Density Decomposition on Large Static and Dynamic Graphs. *Proc. VLDB Endow.* 17, 11 (2024), 2933–2945.