Efficient and Effective Algorithms for Generalized Densest Subgraph Discovery

YICHEN XU, Chinese University of Hong Kong, Shenzhen, China
CHENHAO MA*, Chinese University of Hong Kong, Shenzhen, China
YIXIANG FANG, Chinese University of Hong Kong, Shenzhen, China
ZHIFENG BAO, RMIT University, Australia

The densest subgraph problem (DSP) is of great significance due to its wide applications in different domains. Meanwhile, diverse requirements in various applications lead to different density variants for DSP. Unfortunately, existing DSP algorithms cannot be easily extended to handle those variants efficiently and accurately. To fill this gap, we first unify different density metrics into a generalized density definition. We further propose a new model, c-core, to locate the general densest subgraph and show its advantage in accelerating the searching process. Extensive experiments show that our c-core-based optimization can provide up to three orders of magnitude speedup over baselines. Moreover, we study an important variant of DSP under a size constraint, namely the densest-at-least-k-subgraph (Dal$k$S) problem. We propose an algorithm based on graph decomposition, and it is likely to give a solution that is at least 0.8 of the optimal density in our experiments, while the state-of-the-art method can only ensure a solution with density at least 0.5 of the optimal density. Our experiments show that our Dal$k$S algorithm can achieve at least 0.99 of the optimal density for over one-third of all possible size constraints.

CCS Concepts: • **Theory of computation** → **Graph algorithms analysis**; • **Information systems** → *Data management systems.*

Additional Key Words and Phrases: densest subgraphs, acceleration, core, approximation

ACM Reference Format:
Yichen Xu, Chenhao Ma, Yixiang Fang, and Zhifeng Bao. 2023. Efficient and Effective Algorithms for Generalized Densest Subgraph Discovery. *Proc. ACM Manag. Data* 1, 2, Article 169 (June 2023), 27 pages. https://doi.org/10.1145/3589314

1 INTRODUCTION

Graph data plays essential roles in modeling relationships among objects of interest in various domains, such as social networks, electrical circuits, transportation, and biology. To name a few, the Facebook community has been studied using a graph model with a mapping between users and vertices [64]. The pages and hyperlinks in the World Wide Web can be viewed as vertices and edges in a directed graph [38]. In a graph representing proteins and their interactions, chemical molecules and covalent bonds are mapped to vertices and edges, respectively [65]. To study the

*Chenhao Ma is the corresponding author.

Authors' addresses: Yichen Xu, Chinese University of Hong Kong, Shenzhen, Shenzhen, China, yichenxu@link.cuhk.edu.cn; Chenhao Ma, Chinese University of Hong Kong, Shenzhen, Shenzhen, China, machenhao@cuhk.edu.cn; Yixiang Fang, Chinese University of Hong Kong, Shenzhen, Shenzhen, China, fangyixiang@cuhk.edu.cn; Zhifeng Bao, RMIT University, Melbourne, Australia, zhifeng.bao@rmit.edu.au.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
2836-6573/2023/6-ART169 $15.00
https://doi.org/10.1145/3589314

alternation of patterns and functional connectivity in brains, neuroscientists examine weighted 3-D graphs transformed from brain images [3].

The densest subgraph problem (DSP) has received much attention and lies in the heart of graph mining [9] since it has applications in many fields such as anomaly detection [2, 17], bioinformatics [16, 28, 53], community detection [15], and financial markets [21]. The original density definition of a graph is given by the number of edges over the number of vertices, i.e., $\frac{m}{n}$, where m and n denote the edge and vertex number, respectively.

Generalized density. However, there are many scenarios that the original density cannot cover. The relationship of different densities is depicted in Figure 2 and will be illustrated soon. First, the edges of graphs in real applications often carry weights. For example, in the flight network [18] where airports are denoted as vertices, flights are denoted as edges, and the weight of an edge represents the flight frequency between two airports. Second, Goldberg [31] proposed the denominator weighted density where the weights of vertices are on denominators, and such a weighted density is also adopted by several follow-up studies such as [14, 56]. For instance, Sawlani and Wang [56] developed a method to solve DSP on directed graphs by transforming the directed graph into a set of vertex-weighted graphs and solving the DSP upon the weighted density, where vertex weights are on denominators. There are ample applications for the above density metrics. Taking the weighted density where all weights are on the numerator as an example, a method to detect fraudsters in camouflage adopts the weighted density on weighted graphs and solves the corresponding DSP [34]. Goldberg's max-flow-based algorithm [31], and Chandra's flow-based near-optimal algorithm [14] can be extended to handle the weighted density. Charikar's peeling algorithm [13] and `Greedy++` [11], which repeats the peeling algorithm several times, can also be extended to handle some of the above density metrics.

However, the limitation of these aforementioned methods under new density metrics is that they are either not scalable to large-scale graphs or not capable of yielding a dense graph with a near-optimal density guarantee. Meanwhile, previous work barely targets for building a general framework to boost DSP algorithms over a diverse range of density metrics. To fill this gap, we propose to use a generalized supermodular density to unify different density metrics and develop a framework to speed up the generalized densest subgraph problem (GDS).

Dal$k$S. In some circumstances, users demand for finding large dense graphs. For example, an activity organizer may want to have at least k participants who are familiar with each other. Given such a kind of demand, the densest at-least-k-subgraph problem (Dal$k$S) [4], which is an important yet well-studied variant of DSP, is proposed to ensure that the dense graph has at least k vertices. Large dense subgraphs are useful in many domains such as distributed system [55], spam detection [29] and social networks [47]. Finding the exact solution for Dal$k$S has been proven to be NP-hard [6–8, 26, 46]. Therefore, some algorithms [4, 9, 14, 37, 55] are developed to approximate the exact Dal$k$S. However, no existing work has devised a method to generate a solution with guarantees better than $0.5 \cdot OPT$, i.e. half of the optimal density [1]. In this paper, we will propose a new algorithm based on graph decomposition, which can obtain a solution better than the $0.5 \cdot OPT$ solution with a very high likelihood.

Contributions. In this paper, one of our main goals is to devise a method to accelerate the densest subgraph discovery w.r.t. the generalized density, particularly on large graphs, so that the near-optimal densest subgraphs can be found within a short time. Another breakthrough we make for Dal$k$S, an important variant of DSP, is proposing a new algorithm that is likely to obtain a

[1] A solution with at least $f \cdot OPT$ density ($0 < f \leq 1$) means its density is at least f of the optimal density. For simplicity, we call this solution an $f \cdot OPT$ solution.

solution better than the $0.5 \cdot OPT$ solution achieved by the state-of-the-art. We briefly summarize our contributions below.

- We introduce a new dense subgraph model, c-core, which is general to cover many density metrics for discovering GDS. We propose a computational framework to accelerate algorithms for GDS based on c-core.
- Based on c-cores, we propose an exact algorithm and an approximation algorithm with advanced pruning techniques for flow-based computations and a new strategy to search for the optimal density.
- We successfully derive the upper bound for the size of the exact solution for Dal$k$S and devise a new approximation to Dal$k$S based on our density-friendly decomposition.
- We conduct experiments on 12 real-world weighted and unweighted graphs with up to 1.8 billion edges. Our proposed algorithm `cCoreExact` for GDS is up to three orders of magnitude faster than the original `FlowExact` [31].
  In addition, we empirically show that our proposed approximation Dal$k$S algorithm can output a solution very close to the optimal in most scenarios.

Outline. The organization of the paper is as follows. In Section 2, we review the related work. In Section 3, we unify different density metrics into the generalized supermodular density and define the GDS problem. Section 4 introduces the new c-core model and builds its connection with the GDS problem. Section 5 follows with GDS algorithms based on c-core. The approximation algorithm to Dal$k$S will be presented in Section 6. Experimental results are shown in Section 8, and Section 9 concludes our work.

2 RELATED WORK

Finding dense subgraphs from graphs has been extensively studied [54, 61]. Among different types of dense subgraphs, the Densest Subgraph Problem (DSP) [25] lies at the core of large-scale data mining [9]. Other related topics include k-core [57], k-truss [36], clique and quasi-clique [12], which is described in more detail in [24]. We focus on the densest subgraph problem and its variants in the following.

Densest subgraph problem (DSP). A fundamental focus which lies in the heart of graph mining is to find dense subgraphs [30, 39]. The commonly used edge-density of an undirected unweighted graph $G(V, E)$ is $\frac{m}{n}$ with $n = |V|$ and $m = |E|$ [31]. Works on weighted graphs mainly use two density metrics; one places all the weights on the numerator [31, 34], and the other places the weights of vertices on the denominator [31, 56]. We will give a generalized density definition to cover both cases and more variants.

To solve the densest subgraph problem (DSP), Goldberg [31] devised a max-flow-based algorithm to obtain exact solutions in unweighted and weighted graphs. Despite the high accuracy, the flow-based approach fails to be scaled to very large graphs with tens of millions of edges. Later, researchers proposed the new concept of clique-density and developed efficient exact algorithms for finding the corresponding DS [45, 62]. Generally, the exact DSP algorithms [31, 45, 62] work well on graphs of small or moderate size, but suffer from large graphs.

To further boost efficiency, several approximation algorithms have been developed. Charikar [13] proposed a $\frac{1}{2}$-approximation method [2] for unweighted graphs by repeatedly peeling the vertex with the smallest degree. Bahmani et al. [9] introduced a new algorithm over streaming models running in $O(m \cdot \frac{\log n}{\epsilon})$ to guarantee a $\frac{1}{2+2\epsilon}$-approximation. Feng et al. [27] used spectral theory to develop an algorithm faster than Charikar's peeling to yield a solution with comparable accuracy. In order

[2] f-approximation method/algorithm means that for every input, the algorithm can guarantee a $f \cdot OPT$ solution, $0 < f \leq 1$. In general, all solutions output by the f-approximation algorithm are called f-approximation.

to avoid calling maximum flow, Boob et al. [11] designed an empirically efficient method called `Greedy++` by repeating the peeling process multiple times. Chandra et al. [14] gave a flow-based $(1-\epsilon)$-approximation algorithm by performing a limited number of blocking flows on the flow network.

Despite the focus on DSP, very little can find an approximation close to the exact solution while maintaining high efficiency, especially for the generalized density definition. This bottleneck becomes even trickier when large-scale graphs of up to billions of edges are considered. Therefore, some applications involving DSP on large graphs only utilize naive peeling to make the approximation. For instance, Hooi et al. [34] used a $\frac{1}{2}$-approximation DSP algorithm on weighted graphs to find the fraudsters as the alternative to the exact solution.

Variants of DSP. DSP has also been studied on other graphs, e.g., directed graphs [13, 37, 42–44], dynamic graphs [23, 36], and hypergraphs [10, 35]. Tatti et al. [60] and Danisch et al. [19] studied the density-friendly decomposition problem to decompose the graph into a chain of subgraphs, where each inner subgraph is denser than the outer ones. Qin et al. [51] and Ma et al. [41] studied the locally densest subgraphs problem to find multiple locally dense regions from the graph.

When size-bound restrictions are imposed, the densest subgraph problem becomes NP-hard [6–8, 26, 46]. Specifically, Andersen and Chellapilla [4] utilized Charikar's peeling algorithm to always yield a $\frac{1}{3} \cdot OPT$ solution to the densest at-least-k-subgraph problem (Dal$k$S), where an at-least-k-subgraph means a subgraph with at least k vertices. Chekuri et al. [14] then extended the $\frac{1}{3}$-approximation method to the densest at-least-k supermodular subset problem. To achieve better solutions, Khuller and Saha [37] provided a combinatorial algorithm and a linear-programming-based algorithm to output a $\frac{1}{2} \cdot OPT$ solution result. However, the existing Dal$k$S solution cannot obtain a better guarantee than a solution with density of at least 0.5 of the optimal density.

Comparison. Since some parts of our work are based on [14], we identify our related contributions compared to [14]: First, Chekuri et al. [14] devised a novel flow network and a brief idea for finding DSP with this flow network. Based on the flow network, we complete the implementation detail of how to search for subgraphs with new guessed densities and developed the algorithm `FlowApp`. We also observe that `FlowApp` is not efficient enough and propose a faster algorithm `FlowApp*` to achieve acceleration (Section 5). Second, Chekuri et al. [14] directly investigated the generalized supermodular density, while we manage to show that multiple density metrics are special cases of the generalized supermodular density (Section 3). To the best of our knowledge, we propose to unify weighted density (Definition 3.5), denominator weighted density (Definition 3.7) and h-clique density (Definition 3.9) by the generalized supermodular density (Definition 3.4) for the first time.

3 PROBLEM DEFINITION

In this section, we first review the concept of generalized supermodular density and the generalized densest subgraph based on this density. We then show that several existing DSP variants can be viewed as special cases of the generalized densest subgraph.

Definition 3.1 (Doubly Weighted Graph [66]). A doubly weighted graph is a 4-tuple $G(V, E, W_V, W_E)$, where V and E denote the sets of vertices and edges, respectively, $W_V = \{w_v | v \in V\}$ contains vertex weights, and $W_E = \{w_e | e \in E\}$ contains edge weights.

We denote the subgraph induced by $S \subseteq V$ as $G[S]$, and the edge set in $G[S]$ as $E(S)$.

Example 3.2. Figure 1 presents an instance of doubly weighted graph. The numbers on vertices and edges are their weights, respectively. For instance, node c has weight 4 and edge (c, e) has weight 3.

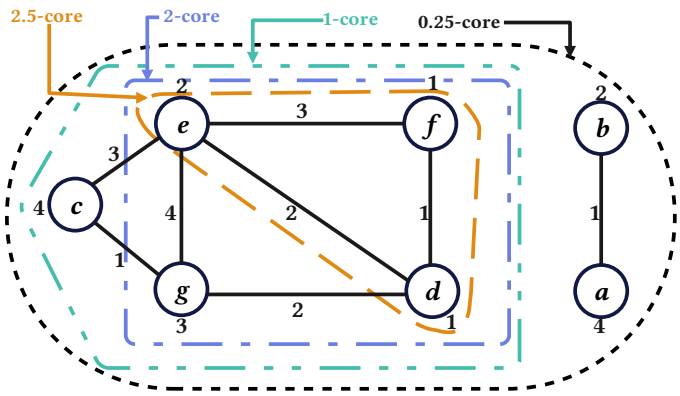

Fig. 1. Doubly weighted graph and c-core.

The doubly weighted graph is general and it covers the concept of the weighted graph. Many density metrics can be defined on the doubly weighted graph such as weighted density and clique density. We propose to unify these density metrics by a generalized supermodular density. Before introducing the generalized supermodular density definition, we first review the concepts of supermodular and submodular as its foundation.

Definition 3.3 (Supermodular & Submodular [14]). Given a space 2^V, a real-valued set function $f : 2^V \to \mathbb{R}$ is supermodular if and only if $f(W) + f(U) \leq f(W \cup U) + f(W \cap U)$, where W and U are any two subsets of V. A set function $g : 2^V \to \mathbb{R}$ is submodular if and only if $-g$ is supermodular.

Definition 3.4 (Generalized Supermodular Density [14]). Given a doubly weighted graph $G(V, E, W_V, W_E)$ and $S \subseteq V$, the generalized supermodular density of S can be described as

$$\rho(S) = \frac{f(S)}{g(S)}, \tag{1}$$

where $f : 2^V \to \mathbb{R}^+$ is a nonnegative supermodular function and $g : 2^V \to \mathbb{R}^+$ is a nonnegative submodular function.

Next, we show that several well-known density variants can be regarded as special cases of the generalized supermodular density.

Definition 3.5 (Weighted Density [31, 34]). Given a weighted graph $G(V, E, W_V, W_E)$ and $S \subseteq V$, the weighted density of S is given by

$$\rho_W(S) = \frac{\sum_{e \in E(S)} w_e + \sum_{v \in S} w_v}{|S|} \tag{2}$$

Proposition 3.6. *The weighted density (Definition 3.5) is a special case of the generalized supermodular density (Definition 3.4) with $g(S) = |S|$ and $f(S) = \sum_{e \in E(S)} w_e + \sum_{v \in S} w_v$.*

Proof. $g(S)$, the denominator, is both supermodular and submodular. For $f(S)$, given any two subsets $U, W \subseteq V$, we have $f(W \cup U) + f(W \cap U) \geq f(W) + f(U)$, as the left hand side contains extra edge weights for all $e = (u, v)$ where $u \in W \setminus U$ and $v \in U \setminus W$. □

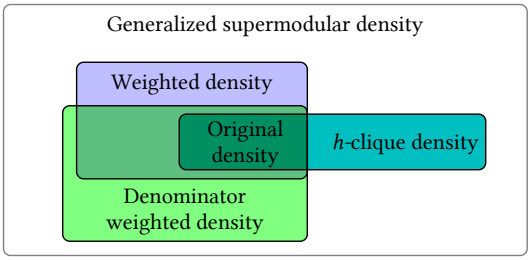

Fig. 2. Relationship of different densities.

Definition 3.7 (Denominator weighted density [31]). Given a weighted graph $G(V, E, W_V, W_E)$ and $S \subseteq V$, the denominator weighted density of S is given by

$$\rho_{DW}(S) = \frac{\sum_{e \in E(S)} w_e}{\sum_{v \in S} w_v} \tag{3}$$

Proposition 3.8. *The denominator weighted density is a special case of the generalized supermodular density (Definition 3.4).*

Definition 3.9 (h-clique density [45, 63]). Given a graph G, for any $S \subseteq V$ its h-clique density can be defined as

$$\rho_h(S) = \frac{c_h(S)}{|S|},$$

where $c_h(S)$ is the number of h-cliques induced by S.

Proposition 3.10. *The h-clique density is a special case of the generalized supermodular density (Definition 3.4).*

Proposition 3.8 and Proposition 3.10 can be proved similarly as Proposition 3.6. The relationship of different density definitions is also illustrated in Figure 2.

Figure 2 depicts the relationships among different density metrics. As can be seen, the generalized supermodular density covers the original density, the weighted density, the denominator density and the h-clique density. The original density is a special case of all of the other density metrics.

Based on the generalized supermodular density definition, we can define the generalized densest subgraph problem.

***Problem* 1** (Generalized Densest Subgraph (GDS) Problem [14])**:** Given a doubly weighted graph G and a generalized supermodular density metric $\rho(S) = \frac{f(S)}{g(S)}$, the GDS problem aims to find the generalized densest subgraph, i.e., $G[S^*]$ where $S^* = \arg\max_{S \subseteq V} \rho(S)$.

Example 3.11. Taking the graph on Figure 1 as an example, if the generalized supermodular density $\rho(S) = \rho_W(S)$, i.e., the weighted density (Definition 3.5), then the GDS will be a subgraph induced by $\{c, g, e\}$ with the density of $\frac{17}{3}$. Similarly, if $\rho(S) = \rho_{DW}(S)$, then the GDS will be $G[\{e, f, d\}]$ with a density of 1.5; if $\rho(S) = \rho_h(S)$ with $h = 3$, then the GDS will be $G[\{c, d, e, f, g\}]$ with a density of $\frac{3}{5}$.

In some applications, the densest subgraph with a size constraint is desired. For example, when organizing conferences, the organizer may want to have at least k participants. Hence, the densest at least k subgraph problem (Dal$k$S) is one kind of DSP with size constraint.

***Problem* 2** (Densest at-least-k-subgraph (Dal$k$S) problem [4])**:** Given a doubly weighted graph G, a corresponding density metric $\rho(S)$ and a size lower bound k, Dal$k$S aims to find the densest at-least-k-subgraph $G[K^*]$, where $K^* = \arg\max_{K \subseteq V} \rho(K), \forall |K| \geq k$.

In this paper, we mainly consider the weighted density (Definition 3.5) for the Dal$k$S problem. Example 3.12 can show what are the exact solutions for Dal$k$S for different size constraints k.

Example 3.12. If $\rho(S) = \rho_W(S)$ is adopted for Dal$k$S on the graph shown in Figure 3, the Dal$k$S is just the GDS when $k \leq 3$. When $k = 4$, the Dal$k$S is induced by $\{c, d, e, g\}$ with a density $\frac{11}{2}$; when $k = 5$, the Dal$k$S is induced by $\{c, d, e, f, g\}$ with a density $\frac{27}{5}$; when $k = 6$, the Dal$k$S is induced by $\{a, c, d, e, f, g\}$ with a density $\frac{31}{6}$; when $k = 7$, the whole graph serves as the Dal$k$S.

4 c-CORE AND GDS

In this section, we introduce a new core model inspired by k-core [57] on unweighted graphs. Next, we present an algorithmic framework that leverages the connection between the cores and the GDS to speed up the GDS searching process.

4.1 Contribution and c-core

The new core model is based on a novel concept, namely *contribution*.

Definition 4.1 (Contribution). Given a doubly graph $G(V, E, W_V, W_E)$, a generalized supermodular density $\rho(S) = \frac{f(S)}{g(S)}$, and a subset $S \subseteq V$, where f and g are defined on space 2^V. The contribution of a vertex $v \in S$ is

$$c_S(v) = \frac{f(S) - f(S \backslash v)}{g(S) - g(S \backslash v)} \tag{4}$$

The subscript of contribution notation means the contribution of the node is calculated with respect to a specific subset $S \subseteq V$.

Definition 4.2 (c-core). Given a weighted graph $G(V, E, W_V, W_E)$, a positive real value c, and a generalized supermodular density $\rho(S) = \frac{f(S)}{g(S)}$, where $S \in V$, a subgraph $G[S]$ is a c-core w.r.t G if it satisfies

(1) $\forall v \in S, c_S(v) \geq c$;
(2) $\nexists S' \subseteq V$, s.t. $S \subset S'$ and S' satisfies (1).

Next, we use an example to illustrate c-cores on a weighted graph when the generalized supermodular density $\rho(S) = \rho_{DW}(S)$ (Definition 3.7). Specifically, we have $\rho(S) = \frac{f(S)}{g(S)} = \frac{\sum_{e \in E(S)} w_e}{\sum_{v \in S} w_v}$.

Example 4.3. Reconsider the graph in Figure 1. According to Definition 4.1, the contribution of a vertex u w.r.t. a subset $S \subseteq V$ is $c_S(u) = \frac{\sum_{e: v \in e \wedge e \in E(S)} w_e}{w_u}$. Based on the contribution formula, the whole graph $G[V]$ is a 0.25-core, as $c_V(a) = 0.25$ is the smallest contribution value among all vertices. If we remove the vertices whose contribution values are not larger than 0.25, we will obtain a subset $S_1 = \{c, d, e, f, g\}$, i.e., a and b are removed. $G[S_1]$ is a 1-core, as $c_{S_1}(c) = 1$ is the smallest among S_1. Peeling vertex with a contribution not larger than one will give us a new subset $S_2 = \{d, e, f, g\}$, where $G[S_2]$ is a 2-core. Similarly, we can also obtain the 2.5-core, $G[S_3]$, where $S_3 = \{e, f, d\}$ by peeling vertices with a contribution not larger than two.

The above example shows that a series of c-cores with increasing coreness of a graph can be obtained by keeping peeling vertices. Similar to the generalized supermodular density covering several density variants, the c-core model can also cover several well-known core models. s-core [22], related to the weighted density (Definition 3.5), is one of such core models.

Definition 4.4 (Strength [22]). Given a doubly weighted graph $G(V, E, W_V, W_E)$ and a vertex $v \in V$. The strength of the node w.r.t. a subset S is defined as

$$s_S(v) = w_v + \sum_{e: v \in e \wedge e \in E(S)} w_e \tag{5}$$

Definition 4.5 (s-core [22]). Given a doubly weighted graph $G(V, E, W_V, W_E)$ and a vertex set $S \in V$. A subgraph $G[S]$ is a s-core w.r.t G if it satisfies

(1) $\forall v \in S, s_S(v) \geq s$;
(2) $\nexists S' \subseteq V$, s.t. $S \subset S'$ and S' satisfies (1).

Proposition 4.6. *Strength (Definition 4.4) is a special case of contribution (Definition 4.1) and thus s-core is a special case of c-core.*

Proof. Let $g(S) = |S|$ and $f(S) = \sum_{v \in S} w_v + \sum_{e \in E(S)} w_e$ in the generalized supermodular density. Observe that g is submodular and f is supermodular. We specialize contribution to strength by definition, i.e. $c_S(v) = s_S(v)$. □

Definition 4.7 (h-clique degree [45]). Given a graph $G(V, E)$ and a vertex $v \in V$. The h-clique degree of the node v w.r.t. a subset S is defined as

$$deg_S(v, h) = |\{\psi | \psi \in G[S], v \in \psi\}|, \tag{6}$$

where ψ is an instance of h-clique.

Definition 4.8 (h-clique-core [45]). Given a graph $G(V, E)$ and a vertex set $S \in V$, a subgraph $G[S]$ is a h-core w.r.t. G if it satisfies

(1) $\forall v \in S, deg_S(v, h) \geq h$;
(2) $\nexists S' \subseteq V$, s.t. $S \subset S'$ and S' satisfies (1).

Proposition 4.9. *h-clique degree (Definition 4.7) is a special case of contribution (Definition 4.1) and thus h-clique-core is a special case of c-core.*

Proof. Let $g(S) = |S|$ and $f(S) = |\{\psi | \psi \in G[S]\}|$ in the generalized supermodular density. Observe that g is submodular and f is supermodular. We specialize contribution to h-clique-degree by definition, i.e. $c_S(v) = deg_S(v, h)$. □

Based on the above discussions, we can find that c-core is efficient to compute via peeling and general to cover different core models. Next, we will show that c-core can also be used to locate the GDS in a small subgraph to speed up the GDS searching.

4.2 Locating GDS in c-cores

We derive some useful properties of c-core and show that these properties are powerful to locate the GDS in some cores. Lemma 4.10 reveals that the contribution (Definition 4.1) of any vertex in the GDS is at least the density of the GDS.

Lemma 4.10. *Given a doubly weighted graph G and a generalized supermodular density $\rho(S) = \frac{f(S)}{g(S)}$, suppose $G[S^*]$ is the GDS w.r.t. ρ. For any $U \subseteq S^*$, we have $\frac{f(S^*) - f(S^* \backslash U)}{g(S^*) - g(S^* \backslash U)} \geq \rho(S^*)$.*

Proof. We prove the lemma by contradiction. Suppose we have $\frac{f(S^*) - f(S^* \backslash U)}{g(S^*) - g(S^* \backslash U)} < \rho(S^*)$.

$$\begin{aligned} &\rho(S^*) \cdot (g(S^*) - g(S^* \backslash U)) > f(S^*) - f(S^* \backslash U) \\ &\Longrightarrow f(S^* \backslash U) > \rho(S^*) \cdot g(S^* \backslash U) \\ &\Longrightarrow \rho(S^* \backslash U) = \frac{f(S^* \backslash U)}{g(S^* \backslash U)} > \rho(S^*) \end{aligned} \tag{7}$$

□

To locate the GDS in c-cores, we first introduce an important property of vertex contribution.

Lemma 4.11. *Suppose there are two vertex subsets S_1 and S_2 satisfying $S_1 \subseteq S_2 \subseteq V$. We have $\forall v \in S_1, c_{S_1}(v) \leq c_{S_2}(v)$.*

Proof. $c_{S_2}(v) = \frac{f(S_2)-f(S_2\setminus v)}{g(S_2)-g(S_2\setminus v)} \geq \frac{f(S_1)-f(S_1\setminus v)}{g(S_1)-g(S_1\setminus v)} = c_{S_1}(v)$. The inequality holds because $f(S)$ is supermodular and $g(S)$ is submodular. □

Based on Lemmas 4.10 and 4.11, we can derive the theorem to locate the GDS in some c-cores. Let the GDS be $G[S^*]$ and its density be $\rho(S^*)$ which is the optimal density. Theorem 4.12 indicates that the GDS $G[S^*]$ is a subgraph of the c-core with c equal to the optimal density.

Theorem 4.12. *Given a doubly weighted graph $G(V, E, W_V, W_E)$, suppose $G[S^*]$ is the GDS. Denote the $\rho(S^*)$-core as $G[C]$. Then, $S^* \subseteq C$.*

Proof. We prove the theorem by contradiction. Suppose $U = S^* \setminus C \neq \emptyset$. By Lemma 4.10, for any $u \in U \subseteq S^*$, we have $c_{S^*}(u) \geq \rho(S^*)$. By Lemma 4.11, $\forall u \in U$, $\rho(S^*) \leq c_{S^*}(u) \leq c_{S^*\cup C}(u)$. Hence, $G[S^* \cup C]$ is a larger $\rho(S^*)$-core than $G[C]$, which contradicts the definition of c-core. □

4.3 c-core-based algorithmic framework

Based on Theorem 4.12, we know that the GDS can be located in the $\rho(S^*)$-core. However, we do not know the exact value $\rho(S^*)$ as a priori before the GDS is found. Fortunately, the density of the densest c-core via peeling can serve as a lower bound of $\rho(S^*)$. In practice, utilizing the density of the densest c-core can help reduce the graph size.

We present an algorithmic framework to accelerate the GDS searching in Algorithm 1. Let $G[\tilde{S}]$ be the densest c-core obtained by peeling on G. In the framework for acceleration, we first find the $G[\tilde{S}]$ via peeling (line 1), use the density of the $G[\tilde{S}]$ as the lower bound $\hat{\rho}$ of $\rho(S^*)$ (line 2) and find the $\hat{\rho}$-core, G' (line 3). Note that $G[S^*] \subseteq \rho(S^*)$-core $\subseteq \hat{\rho}$-core $\subseteq G^{'}$. Next, we can run any GDS algorithm `GDSalg` on G' to find the (approximate) GDS (line 4). We can observe that this framework can locate the GDS in a small subgraph. Hence, the invoked GDS algorithm will be boosted as it only needs to process a small subgraph.

Algorithm 1: `cCoreGDS`

Input: $G(V, E, W_V, W_E)$, density metric $\rho(\cdot)$
Output: The GDS $G[S^*]$ or its approximation

1 $G[\tilde{S}] \leftarrow$ densest c-core in G via peeling;
2 $\hat{\rho} \leftarrow \rho(\tilde{S})$;
3 $G' \leftarrow \hat{\rho}$-core in G via peeling ;
4 $S^* \leftarrow$ `GDSalg`(G');
5 Return $G[S^*]$;

Example 4.13. This example shows the process of Algorithm 1 `cCoreGDS` on the graph in Figure 1 with denominator weighted density (Definition 3.7). Following Example 4.3, we obtain a series of c-cores, $S_1 = \{c, d, e, f, g\}$, $S_2 = \{d, e, f, g\}$ and $S_3 = \{d, e, f\}$ with density 1, $\frac{16}{11}$, $\frac{12}{7}$ and $\frac{3}{2}$ respectively. Observe that in this case, the densest c-core is the subgraph induced by S_2, which is not the c-core with the largest coreness. Then we let $\tilde{S}$ in Algorithm 1 be S_2 and $\hat{\rho} = \rho(S_2) = \frac{12}{7}$. Starting from the whole graph, we peel all vertices with their contribution less than the coreness $\hat{\rho}$. Vertices a, b, c are peeled sequentially and the remaining vertices all have at least $\hat{\rho}$ contributions. The subgraph induced by $\{d, e, f, g\}$ is a $\hat{\rho}$-core by definition and it is the $G^{'}$ in Algorithm 1. Finally, we run the GDS algorithm on the graph $G^{'}$.

5 GDS ALGORITHMS

In this section, we first review existing DSP algorithms on unweighted graphs and discuss how they can be adapted to the GDS problem and fitted into our algorithmic framework. Next, we propose new acceleration techniques for flow-based algorithms to improve their efficiency.

5.1 Existing algorithms

The flow-based exact algorithm [31]. The main idea of Goldberg's flow-based approach [31] is to compare the density of the densest subgraph with a guess value g via max-flow computation and do the binary search to narrow the guess range. Although it can provide accurate results, the max-flow computation is very time costly, especially on large-scale graphs.

Algorithm 2 gives the pseudo-code of Goldberg's `FlowExact` [31]. First, the guess range of the density is initialized as $l = 0$ and $r = \max_{v \in V} c_V(v)$, the maximum contribution (Definition 4.1) among all vertices (line 1). Next, the while loop repeats the binary search to shrink the guess range until the range is smaller than a given coreness (lines 2–8). For each guessed g, the algorithm constructs a flow network (line 4), computes the minimum st-cut (line 5), and updates the range as well as S^* based on st-cut (lines 6–7). `FlowExact` can be extended to handle the weighted density (Definition 3.5). When the weights on edges and vertices are integers, we can guarantee an exact solution by requiring $\delta < \frac{1}{|V| \cdot (|V|-1)}$ [31].

Algorithm 2: `FlowExact` [31]

Input: $G(V, E, W_V, W_E)$, $\delta \in \mathbb{R}^+$
Output: The densest subgraph $G[S^*]$

1 Initialize $l \leftarrow 0$, $r \leftarrow \max_{v \in V} c_V(v)$, $S^* \leftarrow \emptyset$;
2 **while** $r - l > \delta$ **do**
3 $g \leftarrow \frac{r+l}{2}$;
4 Construct flow network F based on G and g;
5 $\langle \mathcal{S}, \mathcal{T} \rangle \leftarrow$ the min st-cut on F;
6 **if** $\mathcal{S} = \{s\}$ **then** $r \leftarrow g$;
7 **else** $l \leftarrow g, S^* \leftarrow \mathcal{S} \setminus \{s\}$;
8 Return $G[S^*]$

The flow-based approximation algorithm [14]. The flow-based approximation algorithm `FlowApp` is proposed by Chekuri et al. [14] to solve DSP. Compared to Goldberg's `FlowExact`, `FlowApp` does not need to run the full maximum flow algorithm. In other words, it can terminate in advance for a given error tolerance ϵ. But it also suffers from the huge cost of performing flow computations on large-scale graphs.

Both `FlowExact` and `FlowApp` can be applied to doubly weighted graphs. For example, if the weighted density (Definition 3.5) is adopted as the generalized supermodular density, the flow network can be constructed as shown in Figure 3 for the 2.5-core in Figure 1. Besides, [31] and [14] provide the flow network for GDS on denominator weighted density (Definition 3.7).

Example 5.1. Figure 3 shows the flow network built upon the 2.5-core in Figure 1 to solve GDS based on Definition 3.5. Each element in the vertex set $\{e, f, d\}$ and the edge set $\{(e, f), (f, d), (e, d)\}$ is treated as a node in the constructed flow network. A source node s is linked to vertex nodes and edge nodes by arcs. The capacity of arcs from s to vertex nodes are weights on the vertices, while the capacity of arcs from s to edge nodes are weights on the edges. Each vertex node is connected

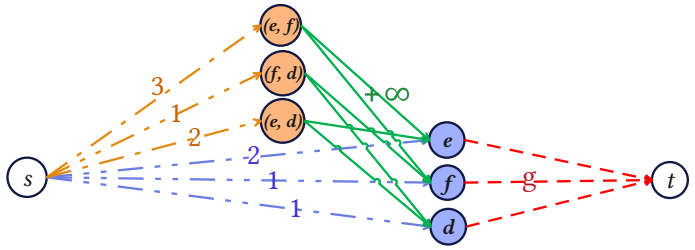

Fig. 3. Flow network constructed from the 2.5-core.

to its incident edge nodes by arcs having infinite capacity. Finally, arcs with the capacity of guessed value g will be built between each edge node and the sink node t.

Apart from the above two flow-based algorithms, `Greedy++` [11] and the Frank-Wolfe-based algorithm [19] can also be adapted to doubly weighted graphs, and fitted into the `cCoreGDS` framework by replacing `GDSalg` in line 4 of Algorithm 1 with the corresponding algorithm. By wrapping the algorithms into the `cCoreGDS` framework, we can perform the GDS searching on smaller subgraphs instead of the whole large graph.

5.2 Boosting flow-based algorithms via cores

Taking a closer look at the flow-based algorithms, we can find that the searching range of the optimal density is shrinking along with the binary search. Hence, the lower bound of the density is monotonically increasing during the binary search. In this case, when the lower bound l in Algorithm 2 increases, we can locate the GDS in a c-core with a higher coreness and smaller size. Replacing the if statement (lines 6–7) in Algorithm 2 with the following lines (Algorithm 3), `FlowExact` will be further boosted by c-cores with smaller sizes during the binary search. Similar code with minor changes can also be added to `FlowApp` to accelerate the GDS searching.

Algorithm 3: c-core-based pruning in flow-based algos

```
1 if S = {s} then r ← g ;
2 else
3     l ← g, S* ← S \ {s};
4     G ← the l-core in G via peeling;
```

5.3 New density search strategy for `FlowApp`

The flow-based approximation algorithm `FlowApp` [14] needs a strategy to search for the optimal density value like the binary search in `FlowExact`. However, Chekuri et al. only gave a brief idea about the strategy (Corollary 2.1 in [14]). That is one can initialize the error tolerance as $\tilde{\epsilon} = 0.5$ and then decrease it by half once the (1-$\tilde{\epsilon}$)-approximation of the subgraph with the new guessed density is found. However, they did not elaborate on how to find the (1-$\tilde{\epsilon}$)-approximation. We give the details and present the strategy in Algorithm 4. Next, to further reduce the searching cost, we develop an advanced searching strategy, which will be given in Algorithm 5.

Similar to the binary search in `FlowExact`, the searching strategy in `FlowApp` [14] also needs to guess the density g within a range (l, r) with some error tolerance ϵ_0. For the guessed g, `FlowApp` will perform a fixed number of blocking flows [1, 5, 32, 58, 59] on the constructed flow network

such as the one in Figure 3. On the residual network after blocking flows, either there exists an easy-to-get subgraph with a density of at least $(1-\tilde{\epsilon})\cdot g$, or there exists no subgraph of density larger than g. The searching range (l, r) will be shrunk accordingly based on one of the two possible outcomes until the error tolerance given by the user is fulfilled.

Algorithm 4: `FlowApp` [14]

Input: $G(V, E, W_V, W_E)$, $\epsilon \in (0, 1)$
Output: The $(1-\epsilon)$-approximation GDS

1 Initialize $\tilde{\epsilon} \leftarrow \frac{1}{2}$, $l \leftarrow 0$, $r \leftarrow \max_{v\in V} c_V(v)$;
2 **while** $\tilde{\epsilon} > \frac{\epsilon}{2}$ **do**
3 $g \leftarrow \frac{r+l}{2}$;
4 Construct flow network F based on G and g;
5 $h \leftarrow$ the number of blocking flows needed;
6 **for** $i = 1 \rightarrow h$ **do** perform blocking flow on F ;
7 **if** *there exists an augmenting path in* F **then**
8 **if** $(1-\tilde{\epsilon})\cdot g \le l$ **then** $\tilde{\epsilon} \leftarrow \frac{\tilde{\epsilon}}{2}$;
9 **else** $l \leftarrow (1-\tilde{\epsilon})\cdot g$, $R_l \leftarrow$ the residual graphs of F ;
10 **else**
11 $r \leftarrow g$;
12 **if** $1 - \frac{l}{r} < \tilde{\epsilon}$ **then** $\tilde{\epsilon} \leftarrow \frac{\tilde{\epsilon}}{2}$;
13 Extract the approximate GDS $G[\tilde{S}^*]$ from R_l;
14 Return $G[\tilde{S}^*]$;

Algorithm 4 gives the pseudo-code of `FlowApp`. `FlowApp` first initializes the error bound $\tilde{\epsilon}$ to $\frac{1}{2}$, and the density range (l, r) to $(0, \max_{v\in V} c_V(v))$ (line 1). Then, we have a while loop to keep guessing the density g and shrink the density range (l, r) based on the result of blocking flows (lines 2–12). In each iteration, the algorithm guesses g, constructs the flow network F, and performs a fixed number of blocking flows (lines 3–6). If there exists an augmenting path in F after blocking flows, this means that there exists a subgraph with the density of at least $(1-\tilde{\epsilon})\cdot g$ (lines 7–9). If $(1-\tilde{\epsilon})\cdot g \le l$, `FlowApp` reduces the error guarantee $\tilde{\epsilon}$ by half, as shown in Case 2 in Figure 4 (a); otherwise l will be updated to $(1-\tilde{\epsilon})\cdot g$, as shown in Case 1 in Figure 4 (a) and `FlowApp` saves the residual graph of F to R_l (lines 8–9). If no augmenting path exists, `FlowApp` updates r to g (line 11), and halves the error bound $\tilde{\epsilon}$. `FlowApp` terminates the loop until the error bound $\tilde{\epsilon}$ satisfies the requirement ϵ (line 2). Finally, it extracts the $(1-\epsilon)$-approximation GDS $G[\tilde{S}^*]$ from the residual graph R_l and returns it as the output (lines 13–14).

Reviewing the above process, we can find that when the while loop is terminated, we have the possible density range (l, r) satisfying $\frac{l}{r} > (1-\epsilon)$. Hence, we can extract a $(1-\epsilon)$-approximate GDS from the residual graph R_l by Theorem 2.1 in [14].

Observations. In practice, we find the strategy to update $\tilde{\epsilon}$ in `FlowApp` [14], which initializes $\tilde{\epsilon} = \frac{1}{2}$ and decreases it by half when appropriate, is sometimes not efficient. The reason lies in the case where there exists a subgraph with density at least $(1-\tilde{\epsilon})\cdot g$, as shown in Figure 4 (a). The narrowing of the density range is slow when $(1-\tilde{\epsilon})\cdot g$ is only slightly greater than l in Case 1 or even unchanged in Case 2. Meanwhile, the error bound $\tilde{\epsilon}$ is halved only in Case 2 and can stay the same for several iterations. Hence, the error bound $\tilde{\epsilon}$ cannot fall below $\frac{\epsilon}{2}$ quickly to fulfill the requirement.

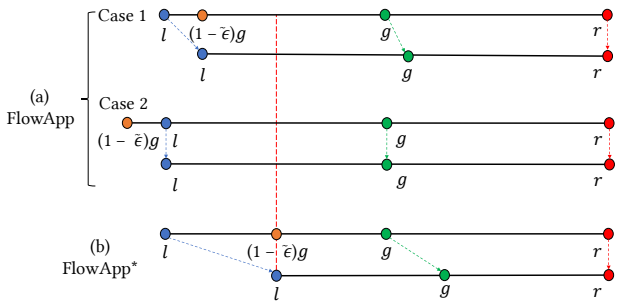

Fig. 4. Illustration of density searching strategies.

To overcome the inefficiency caused by the above intricate strategy, we propose a novel and simple strategy, where the error bound $\tilde{\epsilon}$ is decided adaptively based on the density range (l, r). The advantage of our strategy is that the density range

(1) reduces by $\frac{1}{4}$ steadily, when there exists a subgraph with density at least $(1-\tilde{\epsilon}) \cdot g$, as shown in Figure 4 (b);
(2) reduces by half, when there exists no such subgraph.

Based on this novel strategy, we design a new $(1-\epsilon)$-approximation algorithm, `FlowApp*`, in Algorithm 5. The steps of `FlowApp*` are similar to `FlowApp`. The differences are mainly related to the density searching strategy, as listed below:

(1) the error bound $\tilde{\epsilon}$ is given by $\frac{g-l}{2g}$, where $g = \frac{r+l}{2}$ is the guessed density, and does not follow a fixed decreasing strategy like that in `FlowApp` (line 3);
(2) if there exists a augmenting path in F, l can be safely updated to $(1-\tilde{\epsilon}) \cdot g = \frac{g+l}{2}$ (lines 7–8);
(3) the while loop will be terminated when $\tilde{\epsilon} < \frac{\epsilon}{3-2\epsilon}$ (line 2).

Algorithm 5: `FlowApp*`

Input: $G(V, E, W_V, W_E)$, $\epsilon \in (0, 1)$
Output: The $(1-\epsilon)$-approximation GDS

1 Initialize $\tilde{\epsilon} \leftarrow \frac{1}{2}$, $l \leftarrow 0$, $r \leftarrow \max_{v \in V} c_V(v)$;
2 **while** $\tilde{\epsilon} \geq \frac{\epsilon}{3-2\epsilon}$ **do**
3 $\quad g \leftarrow \frac{r+l}{2}$, $\tilde{\epsilon} \leftarrow \frac{g-l}{2g}$;
4 $\quad$ Construct flow network F based on G and g;
5 $\quad h \leftarrow$ the number of blocking flows needed;
6 $\quad$ **for** $i = 1 \rightarrow h$ **do** perform blocking flow on F ;
7 $\quad$ **if** *there exists an augmenting path in* F **then**
8 $\quad\quad l \leftarrow \frac{g+l}{2}$, $R_l \leftarrow$ the residual graphs of F;
9 $\quad$ **else**
10 $\quad\quad r \leftarrow g$
11 Extract the approximate GDS $G[\tilde{S}^*]$ from R_l;
12 Return $G[\tilde{S}^*]$;

With the new density searching strategy, our FlowApp* can still output a $(1-\epsilon)$-approximation result.

Proposition 5.2. *Algorithm 5 can output a $(1-\epsilon)$-approximation.*

Proof. Consider the last iteration of the while loop. If there exists a subgraph with a density of at least $(1-\tilde{\epsilon})g$, we have $(1-\tilde{\epsilon})g < \rho(S^*) \leq (1+2\tilde{\epsilon})g$. The condition $max_{\rho(S^*)}(1-\frac{l}{\rho(S^*)}) < \epsilon$ can guarantee that we have a $(1-\epsilon)$-approximation. Then we get $1-\frac{(1-\tilde{\epsilon})g}{(1+2\tilde{\epsilon})g} < \epsilon$ which is equivalent to $\tilde{\epsilon} < \frac{\epsilon}{3-2\epsilon}$. Otherwise, we do not have a subgraph with density larger than g, $l < \rho(S^*) \leq g$ and $max_{\rho(S^*)}(1-\frac{l}{\rho(S^*)}) < \epsilon$ can imply $\tilde{\epsilon} < \frac{\epsilon}{2}$. But this condition is satisfied automatically when we require $\tilde{\epsilon} < \frac{\epsilon}{3-2\epsilon}$. □

We further analyze why FlowApp* (Algorithm 5) is faster than FlowApp (Algorithm 4). Comparing (a) and (b) in Figure 4, we observe that FlowApp cannot guarantee how much the searching range is decreased, while FlowApp* ensures that it can reduce the searching range by $\frac{1}{4}$. From the perspective of the termination condition, the faster the decrease of $\tilde{\epsilon}$, the faster the speed of the whole algorithm. In FlowApp, $\tilde{\epsilon}$ cannot decrease (Case 1 in Figure 4 (a)). In FlowApp*, we notice that $\tilde{\epsilon}$ always decreases during the while loop, shown in the following proposition.

Proposition 5.3. *In Algorithm 5, $\tilde{\epsilon}$ strictly decreases. $\tilde{\epsilon}$ in the $(i+1)$-iteration is smaller than the value in the i-th iteration, i.e., $\tilde{\epsilon}_{i+1} < \tilde{\epsilon}_i$.*

Proof. Suppose in the i-th loop, there is an augmenting path in F. Then $\frac{\tilde{\epsilon}_{i+1}}{\tilde{\epsilon}_i} = \frac{\frac{g_{i+1}-l_{i+1}}{2g_{i+1}}}{\frac{g_i-l_i}{2g_i}} = \frac{g_i}{g_{i+1}} \cdot \frac{g_{i+1}-l_{i+1}}{g_i-l_i}$, where the subscripts ($i$ or $i+1$) denote the values in the corresponding iteration of the while loop. Observe that $\frac{g_i}{g_{i+1}} < 1$ and $\frac{g_{i+1}-l_{i+1}}{g_i-l_i} = \frac{3}{4}$. Consequently $\frac{\tilde{\epsilon}_{i+1}}{\tilde{\epsilon}_i} < \frac{3}{4}$. On the other hand, if there is no augmenting path, we have $\frac{\tilde{\epsilon}_{i+1}}{\tilde{\epsilon}_i} < 1$ because $\tilde{\epsilon}_{i+1} = \frac{\frac{g_i+l_i}{2}-l_i}{2\cdot\frac{g_i+l_i}{2}} = \frac{g_i-l_i}{2g_i+2l_i} < \frac{g_i-l_i}{2g_i} = \tilde{\epsilon}_i$. □

6 OUR DALkS APPROXIMATION ALGORITHM

The densest at-least-k-subgraph (Dal$k$S) problem is one kind of DSP with a size constraint, which has been proven to be NP-hard [6–8, 26, 46]. Although the peeling-based Dal$k$S algorithm [4] is fast, it can only output a $\frac{1}{3} \cdot OPT$ solution result, which is far from optimal. To the best of our knowledge, the state-of-the-art approach proposed by Khuller and Saha [37] can output a $0.5 \cdot OPT$ solution, which is still not satisfactory. This section proposes a new algorithm to extract subgraphs close to the optimal solution of Dal$k$S from the density-friendly graph decomposition [60], inspired by [37]. In Section 8, we empirically verify that our solution is usually better than a $0.5 \cdot OPT$ solution.

A key finding inspires our Dal$k$S algorithm DecomDalkS that the GDS $G[S^*]$ must be contained in the Dal$k$S $G[K^*]$ if $|S^*| \leq k$, as shown in Theorem 6.1. In this paper, we focus on the weighted density (Definition 3.5) for Dal$k$S. The reason for choosing the weighted density is that it is more general than the original density. Existing works on Dal$k$S only consider the original density. Thus, our algorithm is more general than existing ones in the literature.

Theorem 6.1. *Given a doubly weighted graph G and size constraint k, let $G[S^*]$ denote the GDS and $G[K^*]$ denote the DalkS. If $k \geq |S^*|$, we have $S^* \subseteq K^*$.*

Proof. Suppose for contradiction, $|S^* \setminus K^*| \neq \emptyset$. Adding $S^* \setminus K^*$ to K will result in a subgraph denser than $G[K^*]$ by Lemma 4.10. □

Motivated by Theorem 6.1 that the GDS is contained in Dal$k$S, can we adopt the following strategy to obtain the near-optimal Dal$k$S?

(1) Find the GDS from doubly weighted graph G;
(2) Remove the GDS from G and redistribute some weights;
(3) Repeat the above process until the size of the union of all GDS's is larger than k, and use the union as a result.

The above strategy can give us a high-quality result, which will be proven later. Meanwhile, Tatti [60] used the above process to perform the density-friendly graph decomposition on unweighted graphs. By deriving properties of density-friendly graph decomposition, which are not shown in [60] and other decomposition work [19], we successfully extract the solution close to the exact Dal$k$S from the decomposition for the first time.

Algorithm 6: `DecomDalkS`

Input: $G(V, E, W_V, W_E)$, size lower bound k
Output: The $\frac{k}{|\tilde{K}^*|}$-approximation Dal$k$S $G[\tilde{K}^*]$

1 $\tilde{K}^* \leftarrow \emptyset$;
2 **while** $|\tilde{K}^*| < k$ **do**
3 $\quad G[S^*] \leftarrow$ the GDS in G via `cCoreGDS` (Algorithm 1);
4 $\quad$ **foreach** $e = (u, v) \in E \cap (S^* \times (V \setminus S^*))$ **do**
5 $\quad\quad w_v \leftarrow w_v + w_e$
6 $\quad$ Remove S^* and its adjacent edges from G;
7 $\quad \tilde{K}^* \leftarrow \tilde{K}^* \cup S^*$;
8 Return the subgraph induced by $\tilde{K}^*$;

We present our Dal$k$S algorithm `DecomDalkS` for doubly weighted graphs in Algorithm 6. `DecomDalkS` first initializes the approximate Dal$k$S as an empty set (line 1). Next, we repeat extractions of the GDS $G[S^*]$ from G (line 3), redistribute weights of edges between vertices inside and outside S^* to corresponding vertices outside S^* (lines 4–5), remove S^* and its adjacent edges from G (line 6), and merge S^* to $\tilde{K}^*$ (line 7), until $\tilde{K}^*$ contains at least k vertices (line 2). We return $G[\tilde{K}^*]$ as the approximate Dal$k$S (line 8).

As `DecomDalkS` keeps updating G at each iteration, we use Table 1 to denote the related variables in i-th iteration of the while loop to facilitate the explanation of the procedure and relevant derivation.

Table 1. Notations in the while loop of `DecomDalkS`.

Notations	Meaning
G_i	updated G at the start of i-th iteration
$G_i[S_i^*]$	the GDS in G_i
$G_i[H_i]$	the Dal$k$S in G_i with at least $(k - \lvert\bigcup_{j=1}^{i-1} S_j^*\rvert)$ vertices

By the following theorem, our algorithm `DecomDalkS` is likely to give a solution with density larger than $0.5 \cdot OPT$, which is the density of the solution given by the state-of-the-art approximation.

Theorem 6.2. *$G[\tilde{K}^*]$ output by* `DecomDalkS` *(Algorithm 6) is a $\frac{k}{|\tilde{K}^*|}$-approximation to the DalkS, $G[K^*]$, with size lower bound k.*

Example 6.3. Take the graph in Figure 1 as an example to demonstrate steps in DecomDalkS with different required k. For clarity, we first list the result of decomposition beforehand. It is easy to obtain $S_1^* = \{c, g, e\}$, $S_2^* = \{f, d\}$, $S_3^* = \{a\}$ and $S_4^* = \{b\}$. If $k \leq 3$, the output is exactly the GDS induced by S_1^*; if $k = 4$, the output is the subgraph induced by $S_1^* \cup S_2^*$, which is a $0.8 \cdot OPT$ solution; if $k = 5$, the output is the same with the case when $k = 4$, but this time it is an exact solution; similarly when $k = 6$ or $k = 7$, DecomDalkS is able to return an exact solution.

According to our experimental results, the approximation ratio given by DecomDalkS, $\frac{k}{|\tilde{K}^*|}$ is at least 0.8 in most cases.

To prepare for the proof of Theorem 6.2, we define the so-called marginal weights as the extension of marginal edge number in [60].

Definition 6.4 (Marginal weight). Suppose we have two disjoint vertex subsets $X \subseteq V$ and $Y \subseteq V$. Denote the edge set to connect X and Y as $E(X, Y) = \{e = (u, v) \in E | u \in X, v \in Y\}$. The marginal weight of X w.r.t Y is $W_\Delta(X, Y) := \sum_{e \in E(X)} w_e + \sum_{v \in X} w_v + \sum_{e \in E(X,Y)} w_e$.

We denote the weight of X as $W(X) = \sum_{e \in E(X)} w_e + \sum_{v \in X} w_v$. Hence, the marginal weight of X w.r.t. Y contains more weights of edges connecting X and Y compared to $W(X)$.

Next, we introduce some useful lemmas related to marginal weights.

Lemma 6.5. *Let $G[S^*]$ be the GDS in G. Then we have $\forall X \subseteq V \setminus S^*$, $\frac{W(S^*)}{|S^*|} > \frac{W_\Delta(X, S^*)}{|X|}$.*

Proof. Because $G[S^*]$ is the GDS of G, the following inequality holds

$$\rho(S^* \cup X) = \frac{W(S^*) + W_\Delta(X, S^*)}{|S^*| + |X|} < \frac{W(S^*)}{|S^*|} = \rho(S^*).$$

Then the result will be straightforward to see. □

Lemma 6.6. *Suppose we have vertex subsets D, A, and B, where D is disjoint from both A and B and nonempty. If $0 < |A| < |B|$ and $\frac{W(D)}{|D|} > \frac{W_\Delta(A,D)}{|A|} > \frac{W_\Delta(B,D)}{|B|}$, we have $\rho(A \cup D) = \frac{W(D)+W_\Delta(A,D)}{|D|+|A|} > \frac{W(D)+W_\Delta(B,D)}{|D|+|B|} = \rho(B \cup D)$.*

Proof. Firstly we let

$$\begin{aligned} W_1 &= \left(\frac{W(D)}{|D|} - \frac{W_\Delta(A, D)}{|A|}\right) \cdot (|B| - |A|) \cdot |D| \\ W_2 &= \left(\frac{W_\Delta(A, D)}{|A|} - \frac{W_\Delta(B, D)}{|B|}\right) \cdot |B| \cdot (|D| + |A|) \end{aligned} \tag{8}$$

Taking the difference between $\frac{W(D)+W_\Delta(A,D)}{|D|+|A|}$ and $\frac{W(D)+W_\Delta(B,D)}{|D|+|B|}$ yields

$$\begin{aligned} &\frac{W(D) + W_\Delta(A, D)}{|D| + |A|} - \frac{W(D) + W_\Delta(B, D)}{|D| + |B|} \\ &= \frac{W_1 + W_2}{(|D| + |B|) \cdot (|D| + |A|)} > 0 \end{aligned} \tag{9}$$

□

In the following, we refer readers to Figure 5 (a) for visualizing Lemma 6.7 and Theorem 6.8; and Figure 5 (b) for visualizing Theorem 6.9.

Lemma 6.7. *For any iteration in the while loop, we have $\forall i$, $|H_i| \leq |S_i^*| + |H_{i+1}|$.*

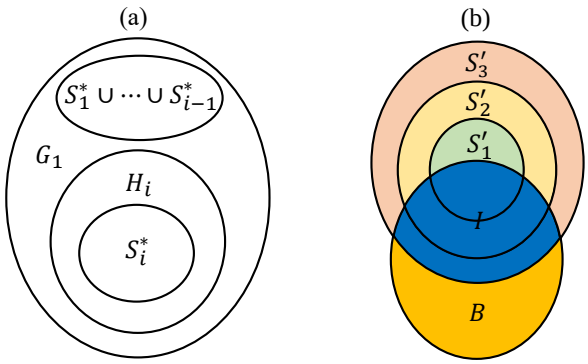

Fig. 5. Relationship among subgraphs.

Proof. Firstly, $S_i^* \subseteq H_i$ by Theorem 6.1. Let $H_i = B_i \cup S_i^*$, where B_i is disjoint from S_i^*. We claim that $|B_i| \leq |H_{i+1}|$. Otherwise suppose that $|B_i| > |H_{i+1}|$. By the definition of S_i^* and Lemma 6.5, we have $\frac{W(S_i^*)}{|S_i^*|} > \frac{W_\Delta(H_{i+1},S_i^*)}{|H_{i+1}|} > \frac{W_\Delta(B_i,S_i^*)}{|B_i|}$ w.r.t. G_i. We use Lemma 6.6 and conclude that

$$\begin{aligned} \frac{W(H_i)}{|H_i|} &= \frac{W(S_i^*) + W_\Delta(B_i, S_i^*)}{|S_i^*| + |B_i|} \\ &\leq \frac{W(S_i^*) + W_\Delta(H_{i+1}, S_i^*)}{|H_{i+1}| + |B_i|} = \frac{W(S_i^* \cup H_{i+1})}{|S_i^*| + |H_{i+1}|} \end{aligned} \tag{10}$$

Observe that the graph induced by $S_i^* \cup H_{i+1}$ is now a denser subgraph than H_i with at least $k - |\bigcup_{j=1}^{i-1} S_j^*|$ vertices on G_i. It contradicts with the fact that H_i is the densest subgraph with at least $k - |\bigcup_{j=1}^{i-1} S_j^*|$ vertices. Because $|B_i| \leq |H_{i+1}|$, we have $|H_i| = |S_i^*| + |B_i| \leq |S_i^*| + |H_{i+1}|$. □

Theorem 6.8. *Suppose the exact solution for DalkS is $G[K^*]$. Then we have $|K^*| \leq |\tilde{K}^*|$, where $\tilde{K}^*$ is the vertex set of the final output in Algorithm 6.*

Proof. Because S_i^*'s are disjoint, we have

$$|\bigcup_{j=1}^{i} S_j^* \cup H_{i+1}| = \sum_{j=1}^{i} |S_j^*| + |H_{i+1}| \tag{11}$$

Suppose the while loop is executed for p iterations. Based on Lemma 6.7, the following inequality holds $\forall 0 \leq i \leq p-2$

$$\sum_{j=1}^{i} |S_j^*| + |H_{i+1}| \leq \sum_{j=1}^{i+1} |S_j^*| + |H_{i+2}| \tag{12}$$

Then, combining the above two, we have the sequence of inequalities, where let $S_{[1,p]}^* = \bigcup_{j=1}^{p} S_j^*$

$$\begin{aligned} |K^*| = |H_1| &\leq |S_1^* \cup H_2| \leq |S_1^* \cup S_2^* \cup H_3| \leq \cdots \\ &\leq |S_{[1,p-1]}^* \cup H_p| = |S_{[1,p]}^*| = |\tilde{K}^*| \end{aligned} \tag{13}$$

We can see that $H_p = S_p^*$, so $|H_p| = |S_p^*|$. □

Theorem 6.9. *$G[\tilde{K}^*]$ is the DalkS with at least $|\tilde{K}^*|$ vertices.*

Proof. Suppose $G[J]$ is any subgraph of the original whole graph $G[V]$, where $|J| = |\tilde{K}^*|$. Let $I = J \cap \tilde{K}^*$, $B = J \setminus I$ and $S_i^{'} = S_i^* \cap (V \setminus I), \forall 1 \leq i \leq p$, where p is the number of iterations executed in the while loop. By Lemma 6.5, we have a sequence of inequalities

$$\frac{W_\Delta(B,I)}{|B|} < \frac{W_\Delta(S_p^*, S_{[1,p-1]}^*)}{|S_p^*|} < \cdots < \frac{W_\Delta(S_2^*, S_{[1,1]}^*)}{|S_2^*|}) < \frac{W(S_1^*)}{|S_1^*|} \tag{14}$$

Taking a closer look at the weights that $\tilde{K}^* \setminus I$ and B bring to I, one can verify the following results with the aid of Lemma 4.10. Note that in the i-th iteration, we transform the problem of density-friendly decomposition to solving GDS on the doubly weighted graph G_i.

$$\begin{aligned} W_\Delta(\tilde{K}^* \setminus I, I) &= \sum_{i=1}^{p} W_\Delta(S_i^{'}, S_{[1,i-1]}^*) \geq \sum_{i=1}^{p} |S_i^{'}| \cdot \frac{W_\Delta(S_i, S_{[1,i-1]}^*)}{|S_i|} \\ &> \frac{W_\Delta(B,I)}{|B|} \cdot \sum_{i=1}^{p} |S_i^{'}| = \frac{W_\Delta(B,I)}{|B|} \cdot |B| = W_\Delta(B,I) \end{aligned} \tag{15}$$

Adding both sizes by $W(I)$ and dividing by $|\tilde{K}^*|$ yields

$$\frac{W(\tilde{K}^*)}{|\tilde{K}^*|} = \frac{W(I) + W_\Delta(\tilde{K}^* \setminus I, I)}{|\tilde{K}^*|} > \frac{W(I) + W_\Delta(B,I)}{|\tilde{K}^*|} = \frac{W(J)}{|J|} \tag{16}$$

□

Based on the above theorems and lemma, we can prove Theorem 6.2 now.

Proof of Theorem 6.2. From Theorem 6.8, we know that $|K^*| \leq |\tilde{K}^*|$. Therefore, $W(K^*) \leq W(\tilde{K}^*)$ because $G(\tilde{K}^*)$ is the densest subgraph with $|\tilde{K}^*|$ vertices. Then the result follows naturally

$$\frac{W(\tilde{K}^*)}{|\tilde{K}^*|} \Big/ \frac{W(K^*)}{|K^*|} = \frac{|K^*|}{|\tilde{K}^*|} \cdot \frac{W(\tilde{K}^*)}{W(K^*)} \geq \frac{k}{|\tilde{K}^*|} \tag{17}$$

□

When $|\tilde{K}^*|$ is close to k, our approximation will be a good solution. In particular, we have the exact solution if $|\tilde{K}^*| = k$. In Section 8.4, we will empirically show that our approximate Dal$k$S's are near-optimal in most cases.

We remark that when $|\tilde{K}^*| > 2k$, one can use the combinatorial algorithm [14] (`Combinatorial-DalkSS`) to generate a $\frac{1}{2}$-approximation naturally. In other words, if $|\tilde{K}^*| = |S_{[1,p]}^*| > 2k$, one can extract $S_{[1,i]}^*, \forall 1 \leq i \leq p$ and randomly add $\max(k - |S_{[1,i]}^*|, 0)$ vertices to each $S_{[1,i]}^*$, and choose the densest induced subgraph among them, which will yield a $0.5 \cdot OPT$ solution.

7 COMPLEXITY ANALYSIS

In this section, we analyze both the time and the space complexity for our proposed algorithms `cCoreExact`, `cCoreApp*` and `DecomDalkS` with the original density metric. The complexity with other density metrics are similar. First, let the input graph be $G(V,E)$ and the c-core to locate GDS be $G^{'}(V^{'},E^{'})$.

The space complexity for all three algorithms is $O(|V|+|E|)$ since storing information for vertices and edges dominates the complexity. We provide the time complexity in the following, along with sketches of the proof.

Proposition 7.1. *The time complexity of* `cCoreExact` *is* $O(|E|+|V|\cdot\log(|V|)+|V^{'}|^2\cdot|E^{'}|\cdot\log(|V^{'}|))$.

Proof. Otaining $G^{'}$ takes $O(|E| + |V| \cdot \log(|V|))$ time. The Dinic's algorithm [20] is to run $\log(|V^{'}|)$ times blocking flows on the c-core. Every time it costs $O(|V^{'}|^2 \cdot |E^{'}|)$ time to compute the blocking flow, so the total time cost is $O(|E| + |V| \cdot \log(|V|)) + O(|V^{'}|^2 \cdot |E^{'}| \cdot log(|V^{'}|))$ and the proposition is proved. □

Proposition 7.2. *The time complexity of* `cCoreApp*` *is* $O(|E| + |V| \cdot \log(|V|) + |E^{'}| \cdot \log(|V^{'}|) \cdot log(|E^{'}|) \cdot \log(\frac{(|V^{'}|+|E^{'}|)^2}{|E^{'}|})/\epsilon)$.

Proof. In `FlowApp*` (Algorithm 5), the number of blocking flow is set to be $h = 2\lceil \log(2|E|) \rceil + 2$ [14]. When the GDS is located in $G^{'}$, we have h blocking flows for every search for new densities and each of them takes $O(2|E^{'}| \cdot \log(|E^{'}|) \cdot \log(\frac{(|V^{'}|+|E^{'}|)^2}{|E^{'}|})/\epsilon)$ [33]. Using the strategy we propose to search for new densities, we perform the search for $\log_{\frac{4}{3}}(|V^{'}|)$ times. Therefore, the total time cost for running blocking flow is $O(|E^{'}| \cdot \log(|V^{'}|) \cdot \log(|E^{'}|) \cdot \log(\frac{(|V^{'}|+|E^{'}|)^2}{|E^{'}|}))/\epsilon)$. By adding the complexity of obtaining $G^{'}$, the result follows. □

Proposition 7.3. *In the worst case, the time complexity of the algorithm* `DecomDalkS` *is* $O(k \cdot |V|^2 \cdot |E| \cdot \log(|V|))$.

Proof. In the worst case, the time cost of `cCoreExact` becomes $O(|V|^2 \cdot |E| \cdot \log(|V|))$. At most k times of decomposition is needed. □

8 EXPERIMENT

8.1 Setup

Datasets. We have used twelve real-world graphs to perform our experiments. Half of them are unweighted graphs shown in Table 2, while the other half are weighted graphs shown in Table 3. The second column on both tables gives short names for the datasets. The edge number varies from around thirty thousand up to two billion.

Table 2. Unweighted graphs.

Dataset	short	# vertices	# edges
Friendster [40]	FT	65,608,366	1,806,067,135
Orkut [40]	OK	30,724,41	117,185,083
LiveJournal [40]	LJ	3,997,962	34,681,189
YouTube [40]	YT	1,134,890	2,987,624
DBLP [40]	DP	317,080	1,049,866
Amazon [40]	AZ	334,863	925,872

We briefly introduce our weighted graphs in Table 3. Libimseti [52] is a weighted graph where vertices represent users, and the weights on edges are ratings given by a user to another one. FacebookForum [49] is a social network where vertices are users, and the weight on each edge is the number of messages. Newman [48] is a scientific collaboration network where a vertex represents an author, and the weight on the edge means the number of joint papers between two authors. OpenFlights [50] contains airports as vertices, and the weight refers to the number of routes between two airports.

An unweighted graph can be viewed as a weighted graph where each edge has a weight value of one. Depending on the application, e.g., fraud detection [34], some methods for weighing unweighted graphs have also been invented, and we use the method proposed by Hooi et al.

Table 3. Weighted graphs.

Dataset	short	# vertices	# edges	weight range
LiveJournal(w) [40]	LW	3,997,962	34,681,189	[2, 11]
Libimseti [52]	LB	220,970	17,359,346	[1, 10]
YouTube(w) [40]	YW	1,134,890	2,987,624	[2, 11]
FacebookForum [49]	FF	899	142,760	[1, 1049]
Newman [48]	NM	16,726	95,188	[1, 37]
OpenFlights [50]	OF	7,976	30,501	[1, 11]

[34]. Suppose we have vertices u and v in G with an edge e to connect them. We assign weight $w_e = [\log(\frac{10}{deg_G(u)+5})] + [\log(\frac{10}{deg_G(v)+5})]$ to the edge, where $deg_G(u)$ denotes the degree of u in G. The weighing method is applied to unweighted graphs, LiveJournal [40] and YouTube [40].

Algorithm. In our experiments, several algorithms are involved, and their performance provides evidence for our theoretical results. We list them and do a short review.

- `FlowExact` [31] is the exact GDS algorithm based on the flow network. Its details can be found in [31].
- `cCoreExact` is our exact GDS algorithm which is based on flow network [31] and c-core acceleration (Sections 4.3 and 5.2) on `FlowExact`.
- `FlowApp` [14] is the $(1-\epsilon)$-approximation algorithm based on max-flow computation. It differs from `FlowExact`, as it does not require finding the exact maximum flow.
- `FlowApp*` is our $(1-\epsilon)$-approximation algorithm with better density searching strategy. (Section 5.3)
- `cCoreApp*` is our $(1-\epsilon)$-approximation algorithm `FlowApp*` with c-core-based acceleration. (Sections 4.3 and 5.2)
- `Greedy++` is an approximate algorithm to find GDS (especially for Definition 3.5). Each time, it will use the information obtained in previous times. The detail of it can be found in [11].
- `cCoreG++` is our accelerated `Greedy++` based on c-core.
- `DecomDalkS` is our decomposition-based near-optimal Dal$k$S algorithm. (Section 6)

All algorithms are implemented in C++[3]. We perform experiments on a Linux machine equipped with two Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz processors with 10 cores and O2 optimization. In our experiments, Dinic's algorithm [20] is used to find blocking flows or attain maximum flow for all flow-based methods. For other alternative blocking flow algorithms including parallelizable ones, we refer readers to [1, 5, 32, 58, 59].

8.2 Evaluation of c-core-based acceleration

Running time. To evaluate our c-core-based acceleration techniques, we compare the running time of two core-based algorithms, `cCoreExact` and `cCoreApp*`, with their corresponding baseline methods, `FlowExact` and `FlowApp*`, respectively. To show how powerful the acceleration based on c-core is, we further perform a comparison between the c-core-based approaches and `Greedy++` [11]. For `FlowExact` and `cCoreExact`, we can obtain the exact GDS. For `FlowApp` and `FlowApp*`, we require them to give $0.999 \cdot OPT$ solution results. For `Greedy++` and `cCoreG++`, we run 100 iterations to obtain a $0.909 \cdot OPT$ solution for `Greedy++` because better solutions, e.g., $0.99 \cdot OPT$ solution, cost too much time, based on the conjecture that `Greedy++` can obtain a $(1+\frac{1}{\sqrt{T}})$ factor

[3]Our code is available at https://github.com/Xyc-arch/Efficient-and-Effecive-algorithms-for-generalized-densest-subgraph-discovery

Table 4. Running time of different GDS algorithms.

Dataset	cCoreExact	FlowExact	cCoreApp*	FlowApp	FlowApp*	Greedy++	cCoreG++	$\frac{\text{FlowExact}}{\text{cCoreExact}}$	$\frac{\text{FlowApp*}}{\text{cCoreApp*}}$	$\frac{\text{Greedy++}}{\text{cCoreExact}}$
LJ	38.17 s	> 72 h	57.55 s	> 72 h	> 72 h	18 m 38 s	29.66 s	> 6790.68	> 4503.91	29.30
FT	42 m 49 s	> 72 h	48 m 25 s	> 72 h	> 72 h	18 h 36 m	34 m 17 s	> 100.92	> 89.23	26.07
OK	11 m 21 s	> 72 h	12 m 15 s	> 72 h	> 72 h	1 h 13 m	56.92 s	> 380.72	> 352.65	6.42
YT	9.63 s	20 h 55 m	12.65 s	28 h 26 m	24 h 31 m	3 m 12 s	8.78 s	7819.68	6977.08	19.99
DP	1.74 s	1 h 35 m	1.49 s	1 h 13 m	53 m 9 s	41.42 s	3.90 s	3275.86	2140.27	23.80
AZ	1 m 42 s	1 h 16 m	1 m 53 s	1 h 6 m	48 m 6 s	10 m 31 s	28.33 s	44.80	25.54	1.08
LB	1 m 13 s	> 72 h	1 m 9 s	> 72 h	> 72 h	2 m 29 s	48.36 s	> 3550.68	> 3756.52	2.05
NM	0.14 s	5.63 s	0.16 s	1 m 9 s	9.99 s	7.10 s	3.12 s	40.21	62.44	50.71
FF	0.27 s	7.80 s	0.33 s	2 m 15 s	17.16 s	4.16 s	3.16 s	28.89	52.00	15.41
OF	0.20 s	1.25 s	0.46 s	20.27 s	3.27 s	3.65 s	3.10 s	6.25	7.11	18.25
LW	42.95 s	> 72 h	58.34 s	> 72 h	> 72 h	20 m 57 s	29.66 s	> 6034.92	> 4442.92	29.27
YW	17.87 s	22 h 17 m	21.92 s	17 h 42 m	15 h 32 m	3 m 25 s	8.78 s	4489.59	2551.09	11.49

approximation after T iterations. We present the running time of the seven algorithms in Table 4. The second to eighth columns represent the time cost of the corresponding algorithm. The last three columns show the corresponding time-cost ratios.

From Table 4, we make the following observations:

- cCoreExact is up to three orders of magnitude faster than FlowExact, especially on large scale-graphs. For example, FlowExact can provide more than 6000 times speedup on LiveJournal and YouTube. The speedup of cCoreApp* over FlowApp* is similar. The c-core-based acceleration is also effective in Greedy++. For example, cCoreG++ has 137.85, 32.55 and 76.95 speedup over Greedy++ on YouTube, Friendster and Orkut, respectively.
- Acceleration using c-core makes the flow-based approaches for GDS searching scalable to large graphs. On large graphs such as Friendster, Orkut, LiveJournal, and MovieLens, FlowExact and FlowApp* cannot give a satisfactory answer within a reasonable running time. In contrast, cCoreExact and cCoreApp* make it possible to find the exact or near-optimal solution within 50 minutes for all graphs.
- Compared with Greedy++, cCoreExact can find a better GDS with less time cost. All ratios in the last column of Table 4 are greater than one. We observe that on nine out of twelve datasets, cCoreExact is over ten times faster than Greedy++. The densities of the subgraphs found by cCoreExact and Greedy++ are shown in Table 6. On four datasets, i.e., Friendster, LiveJournal, MovieLens, and OpenFlights, Greedy++ cannot attain the optimal density achieved by cCoreExact. This result is consistent with the iteration number chosen as $T = 100$ for Greedy++. If a 0.999-approximation is required for Greedy++, the iteration should be set as $T = 1,000,000$ according to the convergence conjecture provided by [11]. However, the time cost of Greedy++ with $T = 1,000,000$ is much larger than that of cCoreExact (one can multiply the ratio of the last column by 10,000 to estimate).

Memory usage. We evaluate the memory usage of cCoreExact and FlowExact over seven datasets. For other datasets, FlowExact cannot finish reasonably within 72 hours. The memory evaluation results are reported in Figure 6. We can find that the memory cost of cCoreExact is less than FlowExact on all seven datasets. Besides, the memory cost of cCoreExact is smaller than FlowApp and FlowApp*, while the cost of the latter two is comparable.

Core size. To explain the improvement of c-core-based acceleration over running time and memory usage, we examine the sizes of $\hat{\rho}$-core in cCoreGDS (Algorithm 1) and the whole graph for different datasets. Given that the $\hat{\rho}$-core is a much smaller subgraph by several orders of magnitude, which is shown in Figure 7, the faster running time and the less memory usage are not surprising.

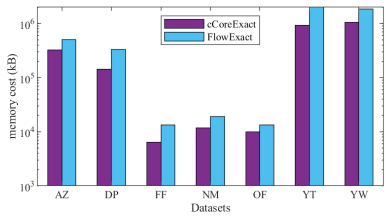

Fig. 6. Memory cost of cCoreExact and FlowExact.

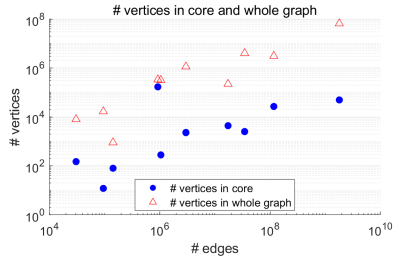

Fig. 7. Number of vertices in the whole graph and $\hat{\rho}$-core.

Table 5. Performance of GDS algorithms with Definition 3.7.

Dataset	# vertex	# edges	$\rho(S^*)$	FlowExact	cCoreExact
WikiVote [40]	7,115	103,689	71.68	7.18 s	3.38 s
Standford [40]	281,903	2,312,497	75.95	5 h 26 m	12 m 34 s
NotreDame [40]	325,729	1,497,134	123.73	2 h 38 m	1 m 11 s

Table 6. Best density by cCoreExact and Greedy++.

Dataset	$\rho(S^*)$ by cCoreExact	$\hat{\rho}(S^*)$ by Greedy++
Friendster	273.52	273.51
Orkut	227.87	227.87
LiveJournal	193.51	193.20
YouTube	45.60	45.60
DBLP	56.57	56.57
Amazon	4.80	4.80
Libimseti	1068.41	1068.24
FacebookForum	1632.10	1632.10
Newman	47.75	47.75
OpenFlights	39.85	39.78
LiveJournal(w)	774.05	774.05
YouTube(w)	168.05	168.05

Generality. We conduct additional experiments on other density metrics to empirically show the generality of c-core acceleration. We choose three directed graphs with different sizes and transform them into bipartite vertex and edge-weighted graphs as described in [56] to facilitate the directed densest subgraph finding. The GDS is found based on the denominator weighted density metric (Definition 3.7), where the vertex weight is placed on the denominator. The statistics of the three directed graphs, as well as the time cost of FlowExact and cCoreExact over those graphs, are presented in Table 5. We can find that the c-core-based acceleration can also provide up to 100 times speedup over the baseline method with Definition 3.7.

8.3 Evaluation of approximation algorithms

Here, we further evaluate the approximation algorithms.

Density searching strategies in flow-based approximation algorithms. In Section 5.3, we design a new strategy to search the optimal density for the flow-based approximation algorithm

and propose FlowApp* based on this new strategy. Here, we perform an ablation study over the strategy to evaluate the speedup provided by FlowApp* over FlowApp. Figure 8 shows the ratio of time cost by FlowApp over that by FlowApp*, i.e. $\frac{time(FlowApp)}{time(FlowApp*)}$. It is easy to see that FlowApp* is faster than FlowApp on eleven out of twelve datasets. On Orkut, the ratio is 0.98, just slightly less than 1. The average speedup for the other eleven datasets is 3.07, and the greatest speedup is 7.88.

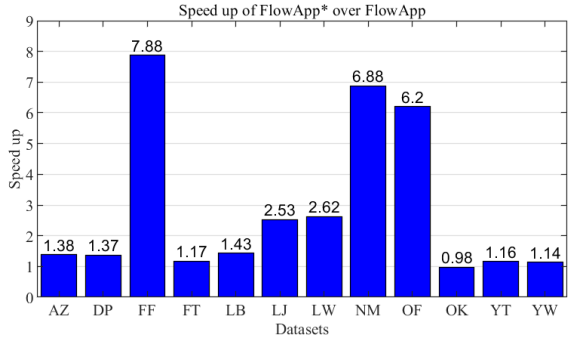

Fig. 8. Speedup of FlowApp* over FlowApp.

Time cost v.s. accuracy. We further test the tradeoff between efficiency and accuracy for the two $(1-\epsilon)$-approximation algorithms, cCoreApp* and Greedy++. We display the trend of time cost w.r.t accuracy in Figure 9. From the result, we can find that cCoreApp* can achieve high accuracy in a much shorter running time compared to Greedy++.

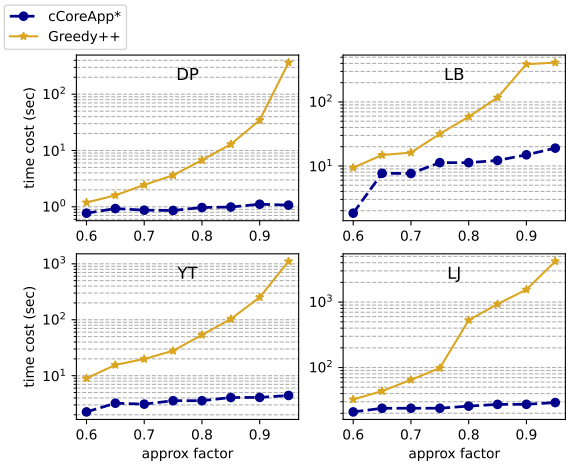

Fig. 9. Time cost v.s. accuracy of approximation algos.

8.4 Evaluation of the Dal$k$S algorithm

In Section 6, we have shown that DecomDalkS can output a $\frac{k}{|\tilde{K}^*|} \cdot OPT$ solution, but have not yet shown the optimality of the algorithm since $|\tilde{K}^*|$ is unknown until we obtain the result $\tilde{K}^*$. We execute DecomDalkS on four graphs and calculate the factor $\frac{k}{|\tilde{K}^*|}$ for any positive integer parameter

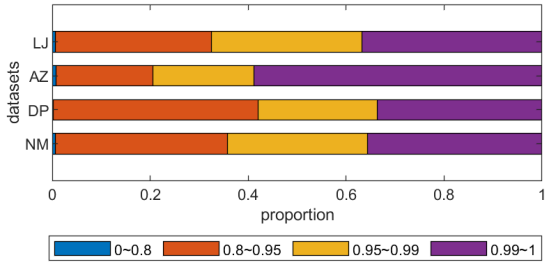

Fig. 10. Approximation ratios provided by DecomDalkS.

k, which is no larger than the total number of vertices in the whole graph. Figure 10 reports the proportion of the factor range for $\frac{k}{|\tilde{K}^*|}$, i.e., $0 \sim 0.8$, $0.8 \sim 0.95$, $0.95 \sim 0.99$ and $0.99 \sim 1$, over four datasets [4] We observe that on all four datasets, the fraction of k values where DecomDalkS cannot guarantee a solution with density at least 0.8 of the optimum is less than 1%. We also note that the factor is larger than 0.5 for any possible k on LiveJournal, Amazon, and DBLP. Interestingly, it is found that on all four graphs, our algorithm can output a subgraph better than $0.99 \cdot OPT$ solution for over one-third of possible k values. Therefore, our algorithm can usually return a solution close to the exact Dal$k$S, while the state-of-the-art offers $0.5 \cdot OPT$ solution guarantees. The time cost of DecomDalkS on LiveJournal, Amazon, DBLP, and Newman is 22 m 16 s, 1 m 53 s, 54 s, and 2 s, respectively.

9 CONCLUSION

This paper investigates the densest subgraph discovery problem with generalized supermodular density and size constraints. We first review and discuss the limitations of existing methods. Next, we show the generalized supermodular density can cover several well-known density variants and devise general acceleration strategies and efficient algorithms to find GDS. In detail, we propose a new concept called c-core and show its applications to find the densest subgraph with generalized supermodular density. Based on c-cores, we devise efficient algorithms cCoreExact and cCoreApp* to find the GDS. We propose an approximation algorithm DecomDalkS based on graph decomposition to find the Dal$k$S. We perform extensive experiments for proposed algorithms on twelve real-world graphs and show that they are efficient (by running up to three orders of magnitude faster) and accurate (by providing exact or near-optimal solutions).

For future work, we will generalize c-core to apply c-core-based acceleration to more graph variants, such as directed graphs, hypergraphs, streaming graphs, etc. For Dal$k$S, it would be interesting to investigate whether efficient exact algorithms based on graph decomposition can be developed.

ACKNOWLEDGMENTS

This work was supported in part by NSFC under Grant 62102341, Basic and Applied Basic Research Fund in Guangdong Province under Grants 2022A1515010166 and 2023A1515011280, Guangdong Talent Program under Grant 2021QN02X826, and Shenzhen Science and Technology Program JCYJ20220530143602006 and ZDSYS20211021111415025.

[4]The reason we choose these datasets is that their sizes vary from small to large, and thus are representative.

REFERENCES

[1] Ravindra Ahuja, James Orlin, Clifford Stein, and Robert Tarjan. 1994. Improved Algorithms for Bipartite Network Flow. *SIAM J. Comput.* 23 (10 1994), 906–933.

[2] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2014. Graph-based Anomaly Detection and Description: A Survey. *CoRR* abs/1404.4679 (2014).

[3] Basak Alper, Benjamin Bach, Nathalie Henry Riche, Tobias Isenberg, and Jean-Daniel Fekete. 2013. Weighted Graph Comparison Techniques for Brain Connectivity Analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, 483–492.

[4] Reid Andersen and Kumar Chellapilla. 2009. Finding Dense Subgraphs with Size Bounds. In *Algorithms and Models for the Web-Graph*. Springer Berlin Heidelberg, 25–37.

[5] Richard J. Anderson and João C. Setubal. 1992. On the Parallel Implementation of Goldberg's Maximum Flow Algorithm. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '92)*. ACM, 168–177.

[6] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. 2002. Complexity of Finding Dense Subgraphs. *Discrete Appl. Math.* 121, 1–3 (sep 2002), 15–26.

[7] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. 2002. Complexity of finding dense subgraphs. *Discrete Applied Mathematics* 121, 1 (2002), 15–26.

[8] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. 2000. Greedily Finding a Dense Subgraph. *Journal of Algorithms* 34 (2000), 203–221.

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

[10] Suman Bera, Sayan Bhattacharya, Jayesh Choudhari, and Prantar Ghosh. 2022. A New Dynamic Algorithm for Densest Subhypergraphs.

[11] 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 '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 573–583.

[12] Mauro Brunato, Holger H. Hoos, and Roberto Battiti. 2008. On Effectively Finding Maximal Quasi-cliques in Graphs. In *Learning and Intelligent Optimization*. Springer Berlin Heidelberg, 41–55.

[13] Moses Charikar. 2000. Greedy Approximation Algorithms for Finding Dense Components in a Graph. In *Approximation Algorithms for Combinatorial Optimization*. Springer Berlin Heidelberg, 84–95.

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

[15] Jie Chen and Yousef Saad. 2012. Dense Subgraph Extraction with Application to Community Detection. *TKDE* 24 (07 2012), 1216–1230.

[16] Tianyi Chen, Francesco Bonchi, David Garcia-Soriano, Atsushi Miyauchi, and Charalampos E Tsourakakis. 2022. Dense and well-connected subgraph detection in dual networks. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*. SIAM, 361–369.

[17] Tianyi Chen and Charalampos Tsourakakis. 2022. Antibenford subgraphs: Unsupervised anomaly detection in financial networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2762–2770.

[18] Edoardo Conti, Steve Cao, and AJ Thomas. 2013. Disruptions in the US airport network. *arXiv* (2013).

[19] Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. 2017. Large Scale Density-friendly Graph Decomposition via Convex Programming. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich (Eds.). ACM, 233–242.

[20] Yefim Dinitz. 2006. Dinitz' Algorithm: The Original Version and Even's Version. 218–240.

[21] Xiaoxi Du, Ruoming Jin, Liang Ding, Victor E. Lee, and John H. Thornton Jr. 2009. Migration motif: a spatial - temporal pattern mining approach for financial markets. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki (Eds.). ACM, 1135–1144.

[22] Marius Eidsaa and Eivind Almaas. 2013. s-core network decomposition: A generalization of k-core analysis to weighted networks. *Phys. Rev. E* 88 (Dec 2013), 062819. Issue 6.

[23] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015. Efficient Densest Subgraph Computation in Evolving Graphs. 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, 300–310.

[24] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *VLDB J.* 29, 1 (2020), 353–392.

[25] Yixiang Fang, Wensheng Luo, and Chenhao Ma. 2022. Densest Subgraph Discovery on Large Graphs: Applications, Challenges, and Techniques. *Proc. VLDB Endow.* 15, 12 (2022), 3766–3769.

[26] Uriel Feige, G Kortsarz, and David Peleg. 2001. The dense k-subgraph problem. *Algorithmica* 29, 3 (2001), 12.

[27] Wenjie Feng, Shenghua Liu, Danai Koutra, Huawei Shen, and Xueqi Cheng. 2020. SpecGreedy: Unified Dense Subgraph Detection. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part I*. Springer-Verlag, 181–197.

[28] Eugene Fratkin, Brian T. Naughton, Douglas L. Brutlag, and Serafim Batzoglou. 2006. MotifCut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics* 22 14 (2006), e150–7.

[29] David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering Large Dense Subgraphs in Massive Graphs. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi (Eds.). ACM, 721–732.

[30] Aristides Gionis and Charalampos E. Tsourakakis. 2015. Dense Subgraph Discovery: KDD 2015 Tutorial. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. ACM, 2313–2314.

[31] Andrew V. Goldberg. 1984. Finding a Maximum Density Subgraph.

[32] Andrew V. Goldberg and Robert E. Tarjan. 1990. Finding Minimum-Cost Circulations by Successive Approximation. *Mathematics of Operations Research* 15, 3 (August 1990), 430–466.

[33] Andrew V. Goldberg and Robert Endre Tarjan. 1990. Finding Minimum-Cost Circulations by Successive Approximation. *Math. Oper. Res.* 15 (1990), 430–466.

[34] 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*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 895–904.

[35] Shuguang Hu, Xiaowei Wu, and T-H. Hubert Chan. 2017. Maintaining Densest Subsets Efficiently in Evolving Hypergraphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17)*. ACM, 929–938.

[36] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 1311–1322.

[37] Samir Khuller and Barna Saha. 2009. On Finding Dense Subgraphs. In *Automata, Languages and Programming*. Springer Berlin Heidelberg, 597–608.

[38] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tompkins, and Eli Upfal. 2000. The Web as a Graph. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '00)*. ACM, New York, NY, USA, 1–10.

[39] Victor Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. 2010. *A Survey of Algorithms for Dense Subgraph Discovery*. 303–336.

[40] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection.

[41] 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.

[42] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, and Xiaolin Han. 2022. A Convex-Programming Approach for Efficient Directed Densest Subgraph Discovery. In *Proceedings of the 2022 International Conference on Management of Data*. 845–859.

[43] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2020. Efficient algorithms for densest subgraph discovery on large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1051–1066.

[44] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2021. On Directed Densest Subgraph Discovery. *TODS* 46, 4 (2021), 1–45.

[45] 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*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1051–1066.

[46] Pasin Manurangsi. 2018. Inapproximability of Maximum Biclique Problems, Minimum k-Cut and Densest At-Least-k-Subgraph from the Small Set Expansion Hypothesis. *arXiv* abs/1705.03581 (2018).

[47] Claire Mathieu and Michel de Rougemont. 2020. Large Very Dense Subgraphs in a Stream of Edges.

[48] M. E. J. Newman. 2001. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences* 98, 2 (2001), 404–409.

[49] Tore Opsahl. 2013. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks* 35, 2 (2013), 159–167.
[50] T. Opsahl, F. Agneessens, and J. Skvoretz. 2010. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks* 32, 3 (2010), 245–251.
[51] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally densest subgraph discovery. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 965–974.
[52] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*.
[53] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. 2010. Dense Subgraphs with Restrictions and Applications to Gene Annotation Graphs. In *Proceedings of the 14th Annual International Conference on Research in Computational Molecular Biology (RECOMB'10)*. Springer-Verlag, 456–472.
[54] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. 2020. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.* 29, 2-3 (2020), 595–618.
[55] Atish Das Sarma, Ashwin Lall, Danupon Nanongkai, and Amitabh Trehan. 2012. Dense Subgraphs on Dynamic Networks.
[56] 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)*. ACM, 181–193.
[57] Stephen B. Seidman. 1983. Network structure and minimum degree. *Social Networks* 5, 3 (1983), 269–287.
[58] Yossi Shiloach and Uzi Vishkin. 1982. An O(n2log n) parallel max-flow algorithm. *Journal of Algorithms* 3, 2 (1982), 128–146.
[59] Robert Endre Tarjan. 1984. A simple version of Karzanov's blocking flow algorithm. *Operations Research Letters* 2, 6 (1984), 265–268.
[60] Nikolaj Tatti. 2019. Density-Friendly Graph Decomposition. *ACM Trans. Knowl. Discov. Data* 13, 5 (2019), 54:1–54:29.
[61] Charalampos Tsourakakis and Tianyi Chen. 2021. Dense subgraph discovery: Theory and application. In *SDM Tutorial*.
[62] 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.
[63] Charalampos E. Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria A. Tsiarli. 2013. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy (Eds.). ACM, 104–112.
[64] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. 2011. The Anatomy of the Facebook Social Graph.
[65] S. Vishveshwara, K. V. Brinda, and Natarajan Kannan. 2002. PROTEIN STRUCTURE: INSIGHTS FROM GRAPH THEORY. *Journal of Theoretical and Computational Chemistry* 01 (2002), 187–211.
[66] Shijie Xu, Jiayan Fang, and Xiangyang Li. 2020. Weighted Laplacian Method and Its Theoretical Applications. *IOP Conference Series: Materials Science and Engineering* 768, 7 (2020), 072032.

Received October 2022; revised January 2023; accepted February 2023