The VLDB Journal (2026) 35:7
https://doi.org/10.1007/s00778-025-00958-0

REGULAR PAPER

Finding Locally Densest Subgraphs: Convex Programming with Edge and Triangle Density

Yi Yang[1] · Chenhao Ma[1] · Reynold Cheng[2] · Laks V. S. Lakshmanan[3] · Xiaolin Han[4]

Received: 19 May 2025 / Revised: 1 November 2025 / Accepted: 9 December 2025

Abstract
Finding the densest subgraph (DS) from a graph is a fundamental problem in graph data management and mining. The DS reveals closely connected entities and has been found to be useful in various application domains such as e-commerce, social science, and biology. However, in a big graph containing billions of edges, it is desirable to find more than one subgraph cluster that is not necessarily the densest, as each dense cluster reveals closely related vertices. In this paper, we study the locally densest subgraph (LDS), a recently proposed variant of DS. An LDS is a subgraph which is the densest among the "local neighbors". Given a graph G, a number of LDSs can be returned, reflecting different dense regions of G and thus giving more information than the DS. Existing solutions for LDS suffer from low efficiency. We thus develop a convex-programming-based solution that enables powerful pruning. We also extend our algorithm to handle triangle-based density and solve the triangle density-based LDS (LTDS) problem. Based on both existing and our proposed algorithms, we propose a unified framework for the LDS and LTDS problems. Extensive experiments on thirteen real large graph datasets show that our proposed algorithm is up to four orders of magnitude faster than the state-of-the-art.

Keywords Densest subgraph · Graph mining · Community detection

1 Introduction

In modern systems and platforms that manage a large number of objects and intricate relationships among them, graphs have been widely used to model such relationships [13, 18, 26, 28–32, 38, 39]. For example, the Facebook friendship network can be modeled as a graph by mapping users to vertices and friendships among users to edges connecting vertices [13]. Fig. 1 depicts a graph of friendship, where users a and f have an edge meaning that they are friends. In biology, graphs can be used to capture complex interactions among different proteins [52] and relationships in genomic DNA [24].

At the core of large scale graph data mining is the densest subgraph (DS) problem [5, 19, 27, 46], which is concerned with finding a "dense" subgraph from a graph G. For example, the DS of Fig. 1 is the subgraph induced by vertex subset S_1, because its density, i.e., the average number of edges over the number of vertices in the subgraph, is the highest among all possible subgraphs of G. The DS has been found useful in various application domains [57]. For example, dense subgraphs can be used to detect communities [12, 58] and discover fake followers [33] in social networks. In biology, the DS found can be used to identify regulatory motifs in genomic DNA [24] and find complex patterns in gene annotation graphs [48]. In graph databases, a DS is used to construct index structures for supporting reachability and distance queries [34]. In system optimization, DS plays

✉ Chenhao Ma
machenhao@cuhk.edu.cn

Yi Yang
yiyang3@link.cuhk.edu.cn

Reynold Cheng
ckcheng@cs.hku.hk

Laks V. S. Lakshmanan
laks@cs.ubc.ca

Xiaolin Han
xiaolinh@nwpu.edu.cn

1 The Chinese University of Hong Kong, Shenzhen, China

2 Department of Computer Science & Guangdong–Hong Kong-Macau Joint Laboratory & HKU Musketeers Foundation Institute of Data Science, The University of Hong Kong, Hong Kong, China

3 The University of British Columbia, Vancouver, Canada

4 Northwestern Polytechnical University, Xi'an, China

Published online: 04 January 2026

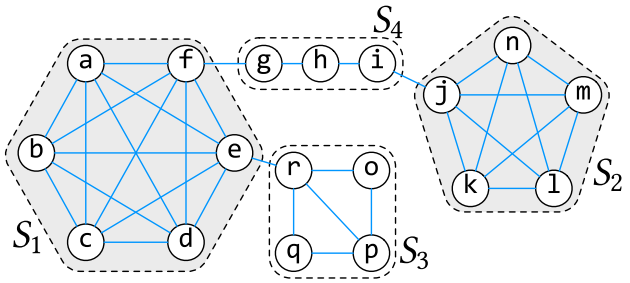

Fig. 1 An undirected graph, G

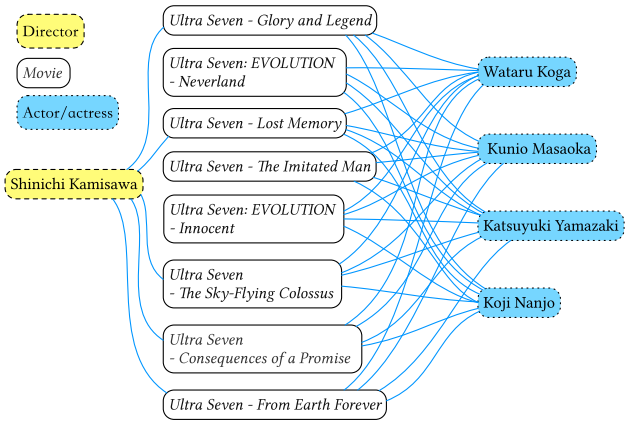

Fig. 2 An LDS about "Ultraman"

an important role in social piggybacking [26, 57], which improves the throughput of social networking systems (e.g., X, formerly Twitter).

Besides the classic edge-based setting, several alternative notions of density have been explored in the literature [44, 56, 58]. Among them, *triangle-based density* has gained particular attention in recent years. Unlike edges, triangles represent stronger and more cohesive connections between nodes, making them especially relevant for capturing community structures and patterns in graphs. Triangles frequently emerge across various domains. For instance, in social networks, a triangle models a tightly-knit group, as friends of friends are often friends themselves. Consequently, identifying subgraphs with high triangle density can reveal richer structures and relationships that edge-based measures may overlook. This characteristic makes triangle-dense subgraphs valuable for various data mining tasks, offering deeper insights by considering higher-order connectivity.

LDS model. Most existing studies [5, 10, 19, 27] on the DS problem focus on finding the densest subgraph. In fact, it is not uncommon to find more than one "dense subgraph" in a given graph. For example, in community detection, it is interesting to explore multiple communities in a social network, even if not all communities have the highest density. Similarly, while dense subgraphs may correspond to real-world phenomena such as echo chambers or protein complexes, actually detecting them entails exploring several dense subgraphs. Motivated by this, Qin et al [46] proposed the locally densest subgraph (LDS) model [46, 49, 57], which identifies several LDSs of a graph. An LDS needs to be *dense* and *compact*. Conceptually, an LDS is a subgraph with the highest density in its vicinity. Moreover, an LDS is "compact", in the sense that any subset of its vertices is highly connected to each other (formal definition in Section 3). For Fig. 1, the two subgraphs induced by vertex subsets S_1 and S_2, denoted $G[S_1]$ and $G[S_2]$ respectively, are two LDSs of G. On the other hand, the subgraph $G[S_1 \cup S_3]$ is not an LDS.

The LDSs have three important properties [46]:

- The LDS is *parameter-free*. As we will explain later, the two LDSs $G[S_1]$ and $G[S_2]$ can be obtained from G in Fig. 1 without setting density threshold or other parameters.
- LDSs of a given graph are pairwise disjoint, i.e., they share no vertices. In the above example, the two LDSs $G[S_1]$ and $G[S_2]$ are disjoint. Thus LDSs naturally allow us to identify all non-overlapping "dense regions" of a graph.
- The set of subgraphs found by the LDS problem is a superset of the subgraph found by the DS problem. For example, DS only identifies $G[S_1]$ in Fig. 1, with a density of 2.5. However, LDS returns $\{G[S_1], G[S_2]\}$ with density 2.5 and 2. Notice that $G[S_2]$ does not overlap with $G[S_1]$.

We have conducted a case study on a large movie graph provided by TCL, an electronic product provider in China. We found that the LDSs found are about different topics (e.g., western films, Chinese martial fiction, Danish comedies). Fig. 2 shows an LDS, with the third highest density, about the Japanese sci-fi series "Ultraman". The DS model can only find a subgraph about western films. The case study shows that the LDS model can find representative dense subgraphs of a graph.

Prior work. Based on the LDS model, Qin et al proposed a max-flow-based LDS solution with pruning techniques built on k-core, named `LDSflow` [46]. Here the k-core [51] of a graph is a maximal cohesive subgraph where each vertex has at least k neighbors in the subgraph. To find LDSs, `LDSflow` adopts a prune-and-verify framework. Specifically, pruning bounds for vertices based on k-core are derived, which are then used to prune vertices. The LDSs are then obtained by verifying the remaining vertices through max-flow computation on the flow network. Inspired by the LDS model, Samusevich et al and [49] generalized it to a triangle-based setting, which is referred to as the locally triangle-densest subgraphs (LTDS) model. A max-flow-based algorithm, named `LTDSflow`, was also proposed to identify the top-

k LTDSs with the largest triangle-based density. The main problem of `LDSflow` and `LTDSflow` is that they may not scale well on large graphs. For example, `LDSflow` needs around 17 hours to output 15 LDSs with the highest densities from a graph with around 3 million edges.

We make the following contributions.

1. *Propose compact number.* Given a vertex v, we define the *compact number* of v, which represents the degree of compactness of the most compact subgraph containing v.
   We further use compact numbers to theoretically link the LDS problem and a convex program for the DS problem, leveraging the fact that vertices in an LDS share the same compact number, and compact numbers can be computed by solving the convex program.
2. *Efficient convex-programming-based algorithm.*
   We propose an elegant prune-and-verify algorithm for the LDS problem based on convex programming, termed `LDScvx`. For pruning, we show that an iterative Frank-Wolfe algorithm [23] can provide tight upper and lower bounds for compact numbers, which can be used to prune from the graph vertices not contained by any LDS. For verification, `LDSflow` [46] performs min-cut computation based on a specific k-core of G. In `LDScvx`, we only need to compute the min-cut based on a smaller subgraph of the k-core by exploiting the upper and lower bounds of compact numbers.
3. *A convex-programming-based algorithm that generalizes the LDS solution to the LTDS problem.* We adapt our `LDScvx` solution for the LTDS problem. This extension preserves the efficiency and pruning strategies of `LDScvx` while ensuring adaptability to the structural differences between LDS and LTDS.
4. *A unified framework for the LDS problem.* Inspired by [64], we present a unifying summary framework for all LDS and LTDS solutions from a high-level perspective. Specifically, this framework consists of four stages: *Initial Reduction*, *Vertex Weight Updating*, *Graph Reduction and Division*, and *Candidate Subgraph Extraction and Verification*. Our framework allows us to compare our methods with existing solutions comprehensively.
5. *Extensive experiments.* We experimentally compare our CP-based algorithm with the existing algorithm on thirteen real large datasets with sizes of up to 1.6 billion edges. Our results show that `LDScvx` is up to four orders of magnitude faster than the state-of-the-art. We also report findings from a case study on a large movie graph.

Outline. The rest of the paper is organized as follows. We review the related work in Section 2. In Section 3, we formally present the LDS problem. Section 4 defines the compact number and discusses its relationship with LDS and convex programming. We present our LDS algorithm in Section 5 and extension for LTDS problem in Section 6. The unified framework and empirical results are shown in Section 7 and Section 8, respectively. Section 9 concludes the paper.

2 Related Work

The densest subgraph is regarded as one kind of cohesive subgraph. Other related topics include k-core [51], k-truss [60], clique, and quasi-clique [14]. More details on them can be found in [21, 22]. Tsourakakis and Chen [57] provides a comprehensive study of techniques and applications of the DS problem. In the following, we focus on densest subgraph discovery and its variants.

Densest subgraph discovery (DS). Goldberg [27] introduced the densest subgraph (DS) problem on undirected graphs, which aims to find the subgraph whose edge-density is the highest among all the subgraphs where the edge-density of a graph $G = (V, E)$ is defined as $\frac{|E|}{|V|}$. To solve the DS problem, Goldberg [27] proposed an exact algorithm based on flow network via solving $O(\log(n))$ min-cut problems. Later, Fang et al [19] improved the efficiency of the flow-based exact algorithm by locating the DS in a specific k-core. Exact algorithms can process small graphs in a reasonable time, but cannot scale well to large graphs. To remedy this issue, several approximation algorithms [5, 8, 10, 19, 63] have been developed.

Kannan and Vinay [35] extended the DS problem definition to directed graphs. Charikar [10] developed an exact polynomial-time algorithm to find the directed densest subgraph by solving $O(n^2)$ linear programs. Khuller and Saha [36] presented a max-flow-based exact algorithm. Ma et al [40–42] improved the max-flow-based algorithm by introducing the notion of $[x, y]$-core and exploiting a divide-and-conquer strategy. To further boost the efficiency of finding DS over large-scale directed graphs, approximation algorithms [10, 36, 40, 43, 50] for the directed DS problem are also developed. Based on existing DS algorithms for both undirected and directed graphs, Zhou et al [64] proposed a unified framework that integrates all algorithms from a high-level perspective.

Further, there are some variants of the DS problem focusing on different aspects. Asahiro et al [4] studied the DS problem with constraints on the size of the DS. However, the size constraints (e.g., at least k vertices or at most k vertices to be included) make the DS problem NP-hard [4]. Hence, Andersen and Chellapilla [2], Khuller and Saha [36], and Chekuri et al [11] studied efficient approximation algorithms on variants of the size-constrained DS. Tsourakakis extended the notion of density based on edges to k-clique-density, and studied the DS problem with k-clique-density [44, 53, 56].

Chang and Qiao [9] proposed a novel and efficient index to report all minimal DS's and enumerate all DS's, where the minimal DS is strictly denser than all of its proper subgraphs. Tatti et al. [54] and Danisch et al. [15] studied the density-friendly decomposition problem, which decomposes a graph into a chain of subgraphs, where each subgraph is nested within the next one, and the inner one is denser than the outer ones. Galbrun et al [25] and Dondi et al [16, 17] studied densest subgraphs with overlaps.

Locally densest subgraph (LDS). Qin et al [46] proposed a new DS model named locally densest subgraph (LDS). Based on this model, users can identify all the locally densest regions of a graph. Qin et al have shown that such subgraphs cannot be found by other DS models theoretically and empirically. Compared to the original DS model, Qin et al [46] showed that the subgraphs provided by the LDS model best represent different local dense regions of the graph, while the DS model only provides the densest subgraph. Furthermore, they compared the LDS model with a straightforward greedy approach for finding top-k densest regions based on the DS, which finds the DS [27] from the current graph, removes the DS from it, and repeats the process for k times. Unfortunately, this greedy approach has several shortcomings [46]: 1. The top-k results may not fully reflect the top-k densest regions. Especially when the graph has a vast dense region, subgraphs in other dense regions may have a low chance to appear. 2. A subgraph returned by the greedy approach can be partial and may be contained by a better subgraph. 3. This greedy approach is essentially a heuristic and does not admit any provable guarantees on the final result.

The above claims are also verified by the experimental results in [46]. With the LDS model justified, Qin et al [46] proposed a max-flow-based algorithm, `LDSflow`, to find the top-k LDSs from a graph.

However, we found that `LDSflow` [46] does not scale to large graphs because `LDSflow` needs to run the max-flow algorithm on the graph several times to find an LDS candidate and verify it, and the max-flow computation can be quite time-consuming. For example, the state-of-the-art max-flow algorithm, proposed by Orlin, has a time complexity of $O(nm)$ [45]. Building upon this foundation, Samusevich et al [49] extended the LDS model from edge-based density to triangle-based density, leading to the `LTDSflow` algorithm. In this work, we go beyond both prior methods by introducing a convex programming framework that unifies and generalizes them. Specifically, our proposed `LDScvx` and `LTDScvx` algorithms replace the expensive max-flow computation with a convex optimization formulation, enabling efficient discovery of locally densest subgraphs under both edge- and triangle-based densities. This formulation not only accelerates the computation but also provides a principled way to integrate pruning and verification within a unified optimization process.

Table 1 Notations and meanings

Notation	Meaning
$G = (V, E)$	a graph with vertex set V and edge set E
$G[S]$	the subgraph induced by S
$d_G(u)$	the degree of a vertex u in G
$\mathsf{density}(G)$	the density of graph G, i.e., $\frac{\lvert E\rvert}{\lvert V\rvert}$
$\phi(u), \phi_{G'}(u)$	compact number of u in G and G' respectively
$\overline{\phi}(u)$	the upper bound of $\phi(u)$
$\underline{\phi}(u)$	the lower bound of $\phi(u)$
$\mathsf{core}_G(u)$	the core number of u in G
$\mathsf{CP}(G)$	the convex program of G for densest subgraph
$\boldsymbol{\alpha}$	the weights distributed from edges to vertices
$\boldsymbol{r}$	the weights received by each vertex

In addition to the aforementioned works, two recent studies have further advanced the research on locally densest subgraph discovery. First, Trung et al [55] proposed a verification-free algorithm (`LDS-Opt`) that efficiently identifies top-k LDSs based on the concept of locally dense subgraphs introduced by Tatti and Gionis [54]. They showed that maximal λ-compact subgraphs correspond to connected components of locally dense subgraphs for certain λ values, enabling a hierarchical construction that eliminates the need for costly max-flow verification. Second, Xu et al [61] extended the LDS problem to the locally h-clique densest subgraph (LhCDS) setting and designed an exact *Iterative Propose–Prune–and–Verify* (`IPPV`) framework that jointly considers h-clique compactness and convex optimization. We acknowledge the importance of these two works and provide detailed discussions and empirical comparisons with them in Appendix B.

3 Preliminaries

This section formally defines the locally densest subgraph problem (LDS problem). Table 1 lists the notations used in this paper.

Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices and $m = |E|$ edges. For each vertex $u \in G$, we use $d_G(u)$ to represent its degree in G. Given a subset $S \subseteq V$, $E(S)$ denotes the set of edges induced by S, i.e., $E(S) = E \cap (S \times S)$. Hence, the subgraph induced by S is denoted by $G[S] = (S, E(S))$. Following the classic graph density definition [3, 5, 10, 27, 46], the density of a graph $G = (V, E)$, denoted by $\mathsf{density}(G)$, is defined as:

$$\mathsf{density}(G) = \frac{|E|}{|V|}. \tag{1}$$

Based on the density definition, the densest subgraph problem is to find the subgraph $G[S]$ of G such that $\mathsf{density}(G[S])$ is maximized.

Densest subgraph discovery is widely applied in many graph mining tasks (e.g., [24, 33, 40, 42, 48]). However, as pointed out by [46], it is usually not sufficient to find one dense subgraph, in many applications such as community detection [18]. Qin et al [46] proposed a locally densest subgraph model to provide multiple dense subgraphs, which are dense and compact. Eq. (1) gives the density of a subgraph. For compactness, Definition 3.1 defines what is a ρ-compact subgraph, which comes from the intuition that a graph is compact if any subset of vertices is highly connected to others in the graph [46].

Definition 3.1 (ρ-compact [46]) A graph $G = (V, E)$ is ρ-compact if and only if G is connected, and removing any subset of vertices $S \subseteq V$ will result in the removal of at least $\rho \times |S|$ edges in G, where ρ is a non-negative real number.

Definition 3.2 (Maximal ρ-compact subgraph [46]) A ρ-compact subgraph $G[S]$ of G is a maximal ρ-compact subgraph of G if and only if there does not exist a supergraph $G[S']$ of $G[S]$ ($S' \neq S$) in G such that $G[S']$ is also ρ-compact.

Remark The locally dense subgraph in [54] is similar to the maximal ρ-compact subgraph. But they are different because the ρ-compact subgraph needs to be connected.

Definition 3.3 (Locally densest subgraph [46]) A subgraph $G[S]$ of G is a locally densest subgraph (LDS) of G if and only if $G[S]$ is a maximal $\mathsf{density}(G[S])$-compact subgraph in G.

From Definition 3.3, we can find that 1. the definition itself is parameter-free; 2. an LDS is compact;

3. any supergraph of an LDS cannot be more compact than the LDS itself; 4. any subgraph $G[S']$ of an LDS $G[S]$ cannot be denser than the LDS itself [46].

These properties show that an LDS is indeed a *locally densest* subgraph. The third one comes from that an LDS $G[S]$ is a maximal ρ-compact subgraph. The last one can be proven by contradiction. Suppose that $G[S']$ has a density satisfying $\mathsf{density}(G[S']) > \mathsf{density}(G[S])$. If we remove vertex set $U = S \setminus S'$ from $G[S]$, the number of edges removed is $\mathsf{density}(G[S]) \times |S| - \mathsf{density}(G[S']) \times |S'| < \mathsf{density}(G[S]) \times (|S| - |S'|) = \mathsf{density}(G[S]) \times |U|$, which contradicts that $G[S]$ is $\mathsf{density}(G[S])$-compact. We further illustrate the LDS definition with the following example.

Example 3.1 (LDS) Consider the graph G shown in Fig. 1. The subgraph $G[S_1]$ with density $\frac{5}{2}$ is a maximal $\frac{5}{2}$-compact subgraph. Hence, $G[S_1]$ is an LDS. Similarly, $G[S_2]$ with density 2 is also an LDS as it is a maximal 2-compact subgraph. The subgraph $G[S_3]$ with density $\frac{5}{4}$ is a $\frac{5}{4}$-compact subgraph. But $G[S_3]$ is not an LDS because it is contained in $G[S_1 \cup S_3]$ which is $\frac{3}{2}$-compact (and also $\frac{5}{4}$-compact). $G[S_1 \cup S_3]$ is also not an LDS, because its density is $\frac{21}{10}$ but it is not a $\frac{21}{10}$-compact subgraph. The compactness of $G[S_1 \cup S_3]$ is $\frac{3}{2} = \frac{6}{4}$, because removing S_3 from $G[S_1 \cup S_3]$ will result in removing of 6 edges. □

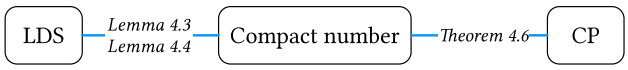

Fig. 3 Relation among LDS, Compact number, and CP

The LDS has a useful property that all the LDSs in a graph G are pairwise disjoint.

Lemma 3.1 *(Disjoint property [46]) Suppose $G[S]$ and $G[S']$ are two LDSs in G, we have $S \cap S' = \emptyset$.*

According to Lemma 3.1, the LDS model can be used to find all dense regions of a graph. However, most real-world applications (e.g., community detection) usually require finding the top-k dense regions of a graph [46]. Hence, we focus on finding the top-k LDSs with the largest densities following [46].

Problem 3.1 (LDS problem [46]) Given a graph G and an integer k, the LDS problem is to compute the top-k LDSs with the largest density in G.

4 From Compactness to CP

In this section, we first propose a new concept named *compact number*, inspired by the core number related to k-core [51], which is a cohesive subgraph model. Then, we present some interesting properties of LDS from the perspective of compact numbers. Afterward, we show that the compact numbers can be computed via a convex programming (CP) formulation of the densest subgraph problem. Hence, the compact number acts as a bridge between the LDS problem and the CP formulation, as shown in Fig. 3.

4.1 Compact Number and LDS

Inspired by the core number of k-core [51], we define the *compact number* of each vertex in graph G with respect to ρ-compact subgraphs of G.

Definition 4.1 (Compact number) Given a graph $G = (V, E)$, the compact number of each vertex $u \in V$, denoted by $\phi(u)$, is the largest ρ such that u is contained in a ρ-compact subgraph of G.

We use $\overline{\phi}(u)$ and $\underline{\phi}(u)$ to denote the upper and lower bounds of $\phi(u)$ in G, respectively. Besides, $\overline{\phi}_{G'}(u)$ denotes the upper bound of $\phi_{G'}(u)$ in G', G' is a subgraph of G.

Example 4.1 (Compact number) Consider vertex $\mathtt{q}$ of G in Fig. 1. The compact number of $\mathtt{q}$ is $\frac{3}{2}$, i.e., $\phi(\mathtt{q}) = \frac{3}{2}$, because $G[S_1 \cup S_3]$ is a $\frac{3}{2}$-compact subgraph containing $\mathtt{q}$ and there is no other subgraph containing $\mathtt{q}$ with a larger ρ. Removing S_3 from $G[S_1 \cup S_3]$ will remove 6 edges, so $G[S_1 \cup S_3]$ is $\frac{3}{2}$-compact.

Next, we discuss the relationship between the LDS and the compact numbers of vertices within or adjacent to the LDS via the following lemmas.

Lemma 4.1 *Given an LDS $G[S]$ in G, $\forall u \in S$, we have $\phi(u) = \mathsf{density}(G[S])$.*

Proof As $G[S]$ is a maximal $\mathsf{density}(G[S])$-compact subgraph, for each $u \in S$, there exists no other subgraph $G[S']$ containing u such that $G[S']$ is a ρ-compact subgraph with $\rho > \mathsf{density}(G[S])$. We prove the claim by contradiction. Suppose $G[S']$ is a ρ-compact subgraph with $\rho > \mathsf{density}(G[S])$ and $u \in S'$, we have $\mathsf{density}(S') \geq \rho > \mathsf{density}(G[S])$. First $S' \subseteq S$, because $G[S]$ is a maximal $\mathsf{density}(G[S])$-compact subgraph and $S' \cap S \neq \emptyset$. If we remove $U = S \setminus S'$ from $G[S]$, the number of edges removed is $|E(S)| - |E(S')| = \mathsf{density}(G[S]) \times |S| - \mathsf{density}(G[S']) \times |S'| < \mathsf{density}(G[S]) \times (|S| - |S'|) = \mathsf{density}(G[S]) \times |U|$. This contradicts that $G[S]$ is $\mathsf{density}(G[S])$-compact. Hence, $\mathsf{density}(G[S])$ is the compact number of all vertices in S. □

Lemma 4.2 *Given an LDS $G[S]$ in G, $\forall (u, v) \in E$, if $u \in S$ and $v \in V \setminus S$, we have $\phi(u) > \phi(v)$.*

Proof The lemma follows from the LDS definition. Suppose $\exists (u, v) \in E, u \in S, v \in V \setminus S$, $\phi(u) \leq \phi(v)$, which contradicts that $G[S]$ is a maximal $\mathsf{density}(G[S])$-compact subgraph. □

Lemmas 4.1and 4.2 indicate that the compact numbers of all vertices in an LDS $G[S]$ are exactly $\mathsf{density}(G[S])$ and the compact numbers of all vertices adjacent to vertices in $G[S]$ but not in $G[S]$ are less than $\mathsf{density}(G[S])$, respectively. We further illustrate the two lemmas via the following example.

Example 4.2 Consider the LDS $G[S_1]$ of G in Fig. 1. We can see that $\forall u \in S_1, \phi(u) = \mathsf{density}(G[S_1]) = \frac{5}{2}$, which fulfills Lemma 4.1. For vertices locating outside $G[S_1]$ but adjacent to some vertices in S_1, i.e., $\mathtt{g}$, $\mathtt{r}$, and $\mathtt{q}$, their compact numbers satisfies Lemma 4.2: $\phi(\mathtt{g}) = \frac{4}{3} < \frac{5}{2}$, $\phi(\mathtt{r}) = \phi(\mathtt{q}) = 2 < \frac{5}{2}$.

According to Lemmas 4.1 and 4.2, compact numbers are powerful to extract and verify LDSs from a graph G. If the compact numbers of all vertices are ready, we can partition the graphs into different subgraphs, where the vertices within the same subgraph share the same compact number and are connected, based on Lemma 4.1. Then, Lemma 4.2 can be used to select LDSs from all subgraphs. Hence, the efficient computation of compact numbers is a key issue. To tackle this issue, we show that compact numbers can be obtained by solving a convex program in the next subsection.

4.2 Compact Number and CP

In this subsection, we first review the convex program (CP) for densest subgraphs by Danisch et al [15]. Next, we theoretically prove that compact numbers can be obtained by solving the convex program.

$$\begin{aligned} \mathsf{CP}(G) \qquad & \min \sum_{u \in V} r_u^2 \\ & r_u = \sum_{(u,v) \in E} \alpha_{u,v}, && \forall u \in V \\ & \alpha_{u,v} + \alpha_{v,u} \geq 1, && \forall (u, v) \in E \\ & \alpha_{u,v}, \alpha_{v,u} \geq 0. && \forall (u, v) \in E \end{aligned} \tag{2}$$

The intuition of eq. (2) is that each edge $(u, v) \in E$ tries to distribute its weight, i.e., 1, between its two endpoints u and v such that the weight sum received by the vertices are as even as possible. Because in the DS $G[S]$ of G, it is possible to distribute all edge weights such that the weight sum received by each vertex in S is exactly $\mathsf{density}(G[S]) = \frac{|E(S)|}{|S|}$. Following the intuition, $\alpha_{u,v}$ in eq. (2) indicates the weight assigned to u from edge (u, v), and r_u is the weight sum received by u from its adjacent edges. Danisch et al [15] used r and α of eq. (2) to tentatively decompose the graph into a chain of subgraphs and applied max-flow to fine-grain and confirm the partitions such that each subgraph is nested within the next one with densities in descending order.

Before proving the compact numbers can be obtained via solving eq. (2), we briefly review the Frank-Wolfe-based iterative algorithm proposed by Danisch et al [15] for optimizing eq. (2). `Frank-Wolfe` (Algorithm 1) outlines the steps to optimize eq. (2). `Frank-Wolfe` first initializes $\boldsymbol{\alpha}$ and $\boldsymbol{r}$ (lines 2–3). Then, in each iteration, each edge $(u, v) \in E$ attempts to distribute its weight, i.e., 1, to the endpoint with a smaller r value (lines 4–10).

Next, we prove that the compact numbers can be extracted from the optimal solution of eq. (2).

Theorem 4.1 *Suppose $(\boldsymbol{r}^*, \boldsymbol{\alpha}^*)$ is an optimal solution of eq. (2). Then, each r_u^* in $\boldsymbol{r}^*$ is exactly the compact number of u, i.e., $\forall u \in V, r_u^* = \phi(u)$.*

Algorithm 1: Frank-Wolfe-based algorithm [15]

Function `Frank-Wolfe`$(G=(V,E), N\in\mathbb{Z}_+)$:
  foreach $(u,v)\in E$ **do** $\alpha^{(0)}_{u,v}\leftarrow\frac{1}{2}$, $\alpha^{(0)}_{v,u}\leftarrow\frac{1}{2}$;
  foreach $u\in V$ **do** $r^{(0)}_u\leftarrow\sum_{(u,v)\in E}\alpha^{(0)}_{u,v}$;
  for $i=1,\ldots,N$ **do**
    $\gamma_i=\frac{2}{i+2}$;
    foreach $(u,v)\in E$ **do**
      if $r^{(i-1)}_u<r^{(i-1)}_v$ **then** $\hat{\alpha}_{u,v}\leftarrow 1, \hat{\alpha}_{v,u}\leftarrow 0$;
      else $\hat{\alpha}_{u,v}\leftarrow 0, \hat{\alpha}_{v,u}\leftarrow 1$;
    $\boldsymbol{\alpha}^{(i)}\leftarrow(1-\gamma_i)\cdot\boldsymbol{\alpha}^{(i-1)}+\gamma_i*\hat{\boldsymbol{\alpha}}$;
    foreach $u\in V$ **do** $r^{(i)}_u\leftarrow\sum_{(u,v)\in E}\alpha^{(i)}_{u,v}$;
  return $(\boldsymbol{r}^{(i)},\boldsymbol{\alpha}^{(i)})$;

Proof For a vertex $u\in V$, let $X=\{v\in V|r^*_v>r^*_u\}$, $Y=\{v\in V|r^*_v=r^*_u\}$, and $Z=\{v\in V|r^*_v<r^*_u\}$. Clearly, $u\in Y$.

We first prove $G[X\cup Y]$ is a r^*_u-compact subgraph. Removing Y from $G[X\cup Y]$ will result in the removal of $r^*_u\times|Y|$ edges in $G[X\cup Y]$. The optimality of $\boldsymbol{r}^*$ implies that

1. $\forall(x,y)\in E\cap(X\times Y)$, $r^*_x>r^*_y$ and $\alpha_{x,y}=0$;
2. $\forall(y,z)\in E\cap(Y\times Z)$, $r^*_y>r^*_z$ and $\alpha_{y,z}=0$.

Otherwise, suppose $\exists(x,y)\in E\cap(X\times Y)$ such that $\alpha_{x,y}>0$ without loss of generality. There exists $r^*_x-r^*_y>\epsilon>0$ such that we can reduce $\alpha_{x,y}$ and increase $\alpha_{y,x}$ by ϵ, respectively. Hence, r^*_x is reduced and r^*_y is increased by ϵ, respectively. After such modification, the objective function be decreased by $2\epsilon(r^*_x-r^*_y-\epsilon)$, which contradicts the optimality of r^*.

Hence, $r^*_u\times|Y|=\sum_{(y,x)\in E\wedge y\in Y}\alpha_{y,x}=|((X\times Y)\cup(Y\times Y))\cap E|$, which is exactly the number of edges to be removed when removing Y from $G[X\cup Y]$. Besides, removing any $Q\subseteq X\cup Y$ from $G[X\cup Y]$ will result in the removal of at least $r^*_u\times|Q|$ edges, because $\sum_{(s,t)\in E(X\cup Y)\wedge s\in Q}1\geq\sum_{(s,t)\in E\wedge s\in Q}\alpha_{s,t}\geq r^*_u\times|Q|$, where the second inequality follows from the first condition of eq. (2). Hence, $G[X\cup Y]$ is a r^*_u-compact subgraph.

For any other subgraph $G[S]$ containing u, $G[S]$ is a ϕ-compact subgraph, where $\phi\leq r^*_u$. Clearly, $S\cap(Y\cup Z)\neq\emptyset$. Removing $S\cap(Y\cup Z)$ from $G[S]$ will result in the removal of no more than $r^*_u\times|S\cap(Y\cup Z)|$ edges, which can be proved by contradiction analogously. □

Theorem 4.1 shows that given an optimal solution $(\boldsymbol{r}^*,\boldsymbol{\alpha}^*)$ to eq. (2), the weight received by each vertex $\boldsymbol{r}^*_u$ is exactly the compact number of u. Example 4.3 further depicts Theorem 4.1 concretely.

Example 4.3 Consider the convex program eq. (2) for G in Fig. 1. We list the optimal solution $(\boldsymbol{r}^*,\boldsymbol{\alpha}^*)$ values in Table 2. Some values of $\boldsymbol{\alpha}^*$ are omitted, as they can be inferred from other r^*_u and $\alpha^*_{u,v}$ values. For each $u\in V$, r^*_u is exactly the compact number of u, $\phi(u)$.

Table 2 $(\boldsymbol{r}^*,\boldsymbol{\alpha}^*)$ to eq. (2) for G in Fig. 1

Vertices	r^*_u	Edges	$\alpha^*_{u,v}$
$u\in S_1$	$\frac{5}{2}$	$(u,v)\in E(S_1)\cup E(S_2)$	$\frac{1}{2}$
$u\in S_2$	2	(g, f), (i, j), (r, e)	1
$u\in S_3$	$\frac{3}{2}$	(g, h), (i, h)	$\frac{1}{3}$
g, h, i	$\frac{4}{3}$	...	

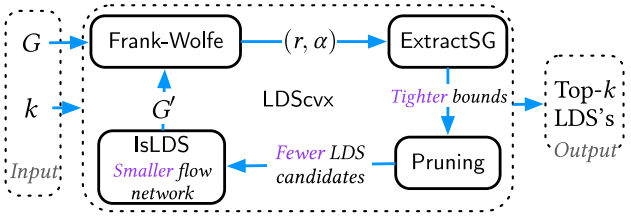

Fig. 4 Algorithm workflow

According to Corollary 4.9 of [15], it may need many iterations for `Frank-Wolfe` to obtain the optimal $(\boldsymbol{r}^*,\boldsymbol{\alpha}^*)$. Fortunately, we found that the approximate solution provided by `Frank-Wolfe` already helps prune the vertices not contained in LDSs and extract LDSs, which will be discussed in the next section.

5 Our LDS Algorithm

In this section, we introduce our convex programming based LDS algorithm, named `LDScvx`. Fig. 4 presents the workflow of `LDScvx`. First, we compute an approximate solution $(\boldsymbol{r},\boldsymbol{\alpha})$ via `Frank-Wolfe`; next, we extract stable groups, which can be used to bound the compact numbers, from G based on $(\boldsymbol{r},\boldsymbol{\alpha})$ via `ExtractSG`; afterward, we prune invalid vertices according to their compact numbers and generate LDS candidates via `Pruning`; finally, we verify the LDS candidates via `IsLDS`. If the verification failed, we repeat the above process to provide higher-quality $(\boldsymbol{r},\boldsymbol{\alpha})$ and compact number estimation.

5.1 Extract Stable Groups

In this subsection, we first introduce a new concept *stable group*, which can be used to provide upper and lower bounds of compact numbers, inspired by the stable subset in [15]. Then, we discuss how to extract stable groups from the approximate solution $(\boldsymbol{r},\boldsymbol{\alpha})$ provided by `Frank-Wolfe`.

Definition 5.1 (Stable group) Given a feasible solution $(\boldsymbol{r},\boldsymbol{\alpha})$ to $\mathsf{CP}(G)$, a stable group with respect to $(\boldsymbol{r},\boldsymbol{\alpha})$ is a non-empty subset $S\in V$, if the following conditions hold.

1. For any $v \in V \setminus S$, r_v satisfies either $r_v > \max_{u \in S} r_u$ or $r_v < \min_{u \in S} r_u$;
2. For any $v \in V$, if $r_v > \max_{u \in S} r_u$, we have that $\forall (v, u) \in E \cap (\{v\} \times S)$, $\alpha_{v,u} = 0$;
3. For any $v \in V$, if $r_v < \min_{u \in S} r_u$, we have that $\forall (u, v) \in E \cap (S \times \{v\})$, $\alpha_{u,v} = 0$.

Definition 5.1 defines stable group. We further use the stable groups w.r.t. $(\boldsymbol{r}^*, \boldsymbol{\alpha}^*)$ (depicted in Fig. 5) as an example to illustrate the properties of the stable groups. The definition indicates that if we sort all vertices $u \in V$ w.r.t. their r values in descending order, we can observe the following properties:

1. The vertices within the same stable group S form a consecutive subsequence of the whole sequence. For example, the stable groups in Fig. 5 give a partition to the entire sequence.
2. The weights of edges whose endpoints fall into different stable groups are assigned to the endpoints with smaller r values. In Fig. 5, we use arrows to denote weight assignments of edges across different stable groups. Note that edges within the same stable group are omitted. We can find that all arrows are pointed to the vertices with smaller r values.

Before discussing how to extract stable groups from $(\boldsymbol{r}, \boldsymbol{\alpha})$, we first prove by the following lemma that the stable groups help derive the upper and lower bounds of compact numbers.

Lemma 5.1 *Given a feasible solution $(\boldsymbol{r}, \boldsymbol{\alpha})$ to eq. (2) and a stable group S w.r.t. $(\boldsymbol{r}, \boldsymbol{\alpha})$, for all $u \in S$, we have that $\min_{v \in S} r_v \le \phi(u) \le \max_{v \in S} r_v$.*

Proof We prove the lemma by contradiction. According to Theorem 4.1, $\forall u \in V$, $\phi(u) = r_u^*$. Suppose there exists a vertex $u \in S$ such that $r_u^* = \phi(u) < \min_{v \in S} r_v \le r_u$. There must exist another vertex $x \in V$ such that $r_x^* = \phi(x) > r_x$. Here $r_x \ge \min_{v \in S} r_v$ according to the 3-rd condition in Definition 5.1. There exists $\epsilon > 0$ such that we could increase r_u^* by ϵ and decrease r_x^* by ϵ, via manipulating the corresponding $\boldsymbol{\alpha}^*$ values, to strictly decrease $\|\boldsymbol{r}^*\|_2^2$ (i.e., the objective function). This contradicts that $\boldsymbol{r}^*$ is the optimal solution to $\mathsf{CP}(G)$. □

According to Lemma 5.1, the minimum and maximum r values in the stable group S are the lower and upper bounds of compact numbers of vertices in S, respectively.

Now the key issue becomes how to extract stable groups from the approximate solution $(\boldsymbol{r}, \boldsymbol{\alpha})$ provided by `Frank-Wolfe`. Our stable groups closely connect to the stable subsets in [15]. Compared to our stable group, the stable subset does not allow vertices not in the subset having larger r values than vertices in the subset. For example, S_2 cannot be a stable subset, but $S_1 \cup S_2$ is a stable subset. Hence, the stable subset [15] can be treated as the union of several stable groups with largest r values. Due to the close relationship between our stable groups and stable subsets in [15], we adapt the stable subset extraction method [15] to extract our stable groups. In general, we first extract the tentative stable groups from $\boldsymbol{r}$, and then verify or merge the tentative ones via Definition 5.1 to give the stable groups. For the optimal solution $(\boldsymbol{r}^*, \boldsymbol{\alpha}^*)$ to $\mathsf{CP}(G)$, we can just group the vertices with the same r values to form the stable groups (refer Table 2 and Fig. 5). Although we cannot perform such aggregation based on $(\boldsymbol{r}, \boldsymbol{\alpha})$, the stable groups obtained from $(\boldsymbol{r}^*, \boldsymbol{\alpha}^*)$ can still provide useful heuristic for us.

Algorithm 2: Extract stable groups from $(\boldsymbol{r}, \boldsymbol{\alpha})$

```
Function ExtractSG(G = (V, E), r, α):
    sort vertices in V according to r: r_{u_1} ≥ r_{u_2} ≥ · ≥ r_{u_n};
    I ← {i | i = arg max_{i≤j≤n} density(G[V_{[1:j]}])};
    Ŝ ← partition V according to I;
    S ← ∅, S ← ∅;
    while Ŝ is not empty do
        S' ← pop out the first candidate from Ŝ;
        S ← S ∪ S';
        // via Definition 5.1
        if S is a stable group then
            put S into S, S ← ∅;
    foreach S ∈ S do
        foreach u ∈ S do φ̄(u) ← min{φ̄(u), max_{v∈S} r_v} ;
        foreach u ∈ S do φ̲(u) ← max{φ̲(u), min_{v∈S} r_v} ;
    return S, φ̄, φ̲
```

Consider the stable groups in Fig. 5. After sorting the vertices according to their r values, let $V_{[1:i]}$ denote the first i vertices in the sequence. We can find that the index i of the last vertex in each stable group is exactly $\arg\max_{j \ge i} \mathsf{density}(G[V_{[1:j]}])$, i.e., the subgraph induced by more vertices than the first i vertices cannot have a larger density. For example, the index of the last element in S_1 is 6, and the density of $G[V_{[1:6]}]$ is 3 and is the maximum among all subgraphs induced by $G[V_{[1:j]}]$, where $j \ge 6$. Hence, we use $\arg\max_{j \ge i} \mathsf{density}(G[V_{[1:j]}])$ to find indices for extracting stable group candidates. Note we break the tie via taking a larger index value for j, when two index values give the same density. Then, we verify each candidate by Definition 5.1. Specifically, we restrict all edges adjacent to the candidate stable group satisfying conditions (2) and (3) of Definition 5.1 by modifying $\boldsymbol{\alpha}$ and $\boldsymbol{r}$ and then check whether condition (1) of Definition 5.1 is fulfilled. If so, the candidate becomes a stable group. Otherwise, it may need to be merged with the next candidate.

Based on the above discussion, we present the algorithm to extract stable groups from $(\boldsymbol{r}, \boldsymbol{\alpha})$, named `ExtractSG`, in Algorithm 2. `ExtractSG` first sorts the vertices in V

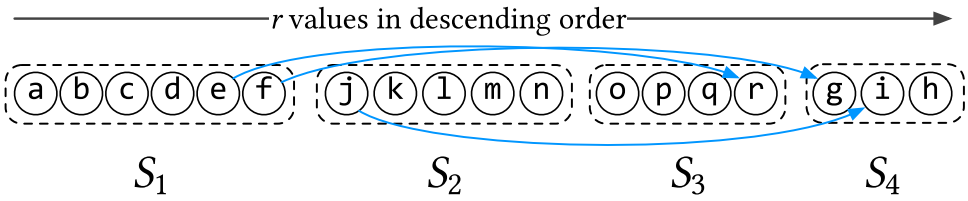

Fig. 5 Stable groups w.r.t. $(\boldsymbol{r}^*, \boldsymbol{\alpha}^*)$

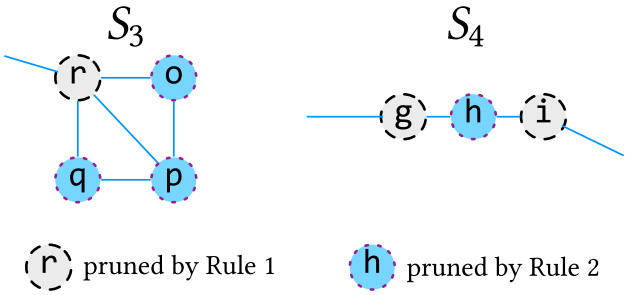

Fig. 6 Pruning rules illustration

according to their r values descendingly (line 2). Then, `ExtractSG` finds the indices I and extracts stable group candidates $\hat{\mathcal{S}}$ following the above heuristic (lines 3–4). Next, we check the candidate in $\hat{\mathcal{S}}$ one-by-one via Definition 5.1 (lines 6–10): if the candidate is a stable group, push it into the list of stable groups $\mathcal{S}$ (lines 9–10); otherwise, the current candidate S will be merged with the next candidate S' in the next iteration (line 8). After all stable groups in $\mathcal{S}$ are obtained, we update the upper and lower bounds of compact numbers according to Lemma 5.1 (lines 11–14). Finally, `ExtractSG` returns the stable groups $\mathcal{S}$ and updated upper and lower bounds of compact numbers (line 14).

5.2 Prune Invalid Vertices

In this subsection, we present how to prune the vertices, which are certainly not contained by any LDS, based on compact number bounds derived in Section 5.1.

We begin with a powerful corollary.

Corollary 5.1 *(Pruning rule 1) For any $u \in V$, if $\exists (u, v) \in E$, such that $\underline{\phi}(v) > \overline{\phi}(u)$, u is not contained by any LDS in G.*

Proof The corollary directly follows Lemma 4.2. □

Example 5.1 (Pruning rule 1) Reconsider the graph G in Fig. 1 and the stable groups in Fig. 5. For vertices in S_3, we can prune r, as shown in Fig. 6. Because for edge (e, r), we have $\overline{\phi}(\mathtt{r}) = \frac{3}{2} < \underline{\phi}(\mathtt{e}) = \frac{5}{2}$, respectively. Similarly, the two vertices g and i in S_4 are also pruned by Pruning rule 1 (Corollary 5.1).

Following Example 5.1, we can find that after removing r and q from G, denoting the graph after removal by G', there is only one edge adjacent to o and q, as shown in Fig. 6. We can find that $\overline{\phi}_{G'}(\mathtt{o}) = \overline{\phi}_{G'}(\mathtt{q}) = 1 < \underline{\phi}(\mathtt{o}) = \underline{\phi}(\mathtt{q}) = \frac{3}{2}$. It means that any LDS in G cannot contain o and q, because it needs to include some already pruned vertices, such as r, to obtain a $\underline{\phi}(\mathtt{o})$-compact subgraph containing o or q in G. Hence, we derive another pruning rule from the above case.

Lemma 5.2 *(Pruning rule 2) Let G' denote the graph after pruning vertices according to Corollary 5.1 and Lemma 5.2. For any vertex u in G', if $\overline{\phi}_{G'}(u) < \underline{\phi}(u)$, u is not contained by any LDS in G.*

Proof $\overline{\phi}_{G'}(u) < \underline{\phi}(u)$ means that only relying on the vertices in G' cannot form a $\underline{\phi}(u)$-compact subgraph containing u, which means that some already pruned vertices are needed. Hence, u cannot be contained by any LDS in G. □

To efficiently compute $\overline{\phi}_{G'}(u)$ in G', we use k-core [51], which is a cohesive subgraph model, following [46].

Definition 5.2 (k-core and core number [51]) The k-core of G is the maximal subgraph $G[S]$ such that for any $u \in S$, $d_{G[S]}(u) \geq r$. For any $u \in V$, the *core number* of u, denoted by $\mathsf{core}_G(u)$, is the largest k such that u is contained in the k-core of G.

Lemma 5.3 *Let G' denote the graph after pruning invalid vertices. $\mathsf{core}_{G'}(u)$ provides an upper bound of $\phi_{G'}(u)$.*

Proof sketch. The lemma follows from Lemma 4.7 of [46]. □

Following the above discussion about Example 5.1, Lemma 5.3 provides a useful approach to obtain the upper bounds of compact numbers of o and p after r and q are removed.

Example 5.2 (Pruning rule 2) After r is pruned from G in Example 5.1, we obtain the upper bounds of compact numbers of o, q, and p in the residual graph G' via Lemma 5.3: $\overline{\phi}_{G'}(\mathtt{o}) = \overline{\phi}_{G'}(\mathtt{q}) = \overline{\phi}_{G'}(\mathtt{p}) = 1$. Then, we apply Pruning rule 2 (Lemma 5.2) to remove o, q, and p from the graph, as shown in Fig. 6. Analogically, h in S_4 is also pruned.

Following the above two examples, we further compare our pruning rules with those in `LDSflow` [46]. `LDSflow` mainly used core numbers for pruning: for vertex u, if $(u, v) \in E$ and $\mathsf{core}_G(u) < \frac{\mathsf{core}_G(v)}{2}$, or $\mathsf{core}_{G'}(u) < \frac{\mathsf{core}_G(u)}{2}$, u can be pruned, where G' denotes the graph with

some vertices already pruned. From the perspective of compact numbers, the rationale behind the pruning in LDSflow is that they actually used core numbers to provide relatively loose upper and lower bounds for compact numbers.

Algorithm 3: Prune invalid vertices

```
1  Function Pruning(G = (V, E), S, φ̄, φ̲):
2      G' = (V', E') ← G;
3      foreach (u, v) ∈ E do
4          if φ̄(u) < φ̲(v) then remove u from G';
5      compute core_{G'}(u) for all vertices in G';
6      while ∃u ∈ V', core_{G'}(u) < φ̲(u) do
7          remove u from G';
8          update core numbers of vertices adjacent to u;
9      foreach stable group S ∈ S do S ← S ∩ V';
10     return S;
```

Based on the two pruning rules, i.e., Corollary 5.1 and lemma 5.2, we present our pruning algorithm, named Pruning, in Algorithm 3. We first replicate the graph G to G' (line 1). Then, we apply Pruning rule 1 (Corollary 5.1) to remove invalid vertices (lines 3–4). Next, we compute the core numbers for all vertices in G' (line 5). Afterwards, Pruning rule 2 (Lemma 5.2) is applied (lines 6–8). Finally, Pruning updates the stable groups by intersecting them with the vertices not been pruned (line 9), and returns the updated stable groups (line 10).

The following subsection will introduce how to extract and verify LDS from the updated stable groups.

5.3 Extract and Verify LDS

Here, we discuss how to extract and verify the LDS from the stable groups after pruning. First, we can find that the vertices within the stable group S with largest r values in $\mathcal{S}$ satisfy Lemma 4.2 w.r.t. the current valid vertices, otherwise they are pruned in Pruning. But, we are not sure whether $G[S]$ is the densest among all its subgraphs (i.e., $G[S]$ is self-densest) and whether there exists another larger $\mathsf{density}(G[S])$-compact subgraph of G containing $G[S]$, according to Definition 3.3.

We first examine whether $G[S]$ is self-densest because the computation cost for self-densest examination is smaller than checking whether it is a maximal $\mathsf{density}(G[S])$-compact subgraph.

Verifying whether $G[S]$ is the densest among all subgraphs of $G[S]$ is one step in the binary search of computing densest subgraph [27], i.e., checking whether there is a subgraph with higher density than $\mathsf{density}(G[S])$. Generally, we use IsDensest to verify the self-densest via computing the max-flow on the flow network generated based on $G[S]$ following [53].

If $G[S]$ is the DS of itself (i.e., IsDensest returns True), we need to further verify whether $G[S]$ is the maximal $\mathsf{density}(G[S])$-compact subgraph in G. We first review how $G[S]$ is verified as an LDS in [46], and next we give our improved verification algorithm.

Qin et al [46] first use breadth-first-search starting from $G[S]$ to traverse each vertice u with $\overline{\phi}(u) \geq \mathsf{density}(G[S])$. Recall that they use $\mathsf{core}_G(u)$ as $\overline{\phi}(u)$ (briefed in Section 5.2). We use G^t to denote the subgraph traversed. If there does not exist an already computed LDS in G^t, $G[S]$ is an LDS. Otherwise, Qin et al construct a flow network based on G^t, then use the min-cut algorithm to find all maximal $\mathsf{density}(G[S])$-compact subgraphs in G^t, and check whether $G[S]$ is maximal.

We can observe that the verification algorithm in [46] needs to compute the min-cut on the flow-network based on the vertices with $\overline{\phi}(u) \geq \mathsf{density}(G[S])$. We will show that only the vertices with $\overline{\phi}(u) \geq \mathsf{density}(G[S])$ and $\underline{\phi}(u) \leq \mathsf{density}(G[S])$, which form a subset of the set of vertices needed in [46], are needed to verify whether $G[S]$ is an LDS of G.

Algorithm 4 presents our improved verification algorithm, named IsLDS. IsLDS first initializes an empty queue Q, an empty vertex set U, an empty edge set L, and ρ with $\mathsf{density}(G[S])$ (lines 2–3). Next, the algorithm performs a breadth-first search starting from S (lines 4–13). Specifically, IsLDS uses Q to store the vertices to be traversed. Each time, it pops out the first vertex v from Q (line 5), and iterates all neighbors of v (lines 8–13). For each neighbor w, if $\underline{\phi}(w) > \rho$, w will not be added to U and Q, but a self loop of v is added to L (lines 10–11). If $\underline{\phi}(w) \leq \rho \leq \overline{\phi}(w)$, w is added into Q and U (lines 12–13). If IsLDS does not encounter a vertex with $\underline{\phi}(w) > \rho$ during the traversal, we can return True (line 14), which means that there does not exist an already computed LDS in the traversed subgraph. Otherwise, we construct a subgraph G^t with all edges induced by U and self loops in L (lines 15). Afterward, we compute all ρ-compact subgraphs in G^t via min-cut following [46] (line 16). Finally, we return True if $G[S]$ is maximal ρ-compact; otherwise, False is returned (line 17). We can observe that G^t only contains vertices with $\overline{\phi}(w) \geq \mathsf{density}(G[S]) \geq \underline{\phi}(w)$. Hence, the flow network generated in our algorithm is much smaller than that generated in [46].

Before proving the correctness of Algorithm 4, we use an example to illustrate the traversed subgraph G^t.

Example 5.3 Consider the graph in Fig. 7. Suppose $S = \{\mathtt{f}, \mathtt{g}, \mathtt{h}\}$ and we want to verify whether $G[S]$ is an LDS of G. Clearly, $G[S]$ is the DS of itself. We illustrate the scope of the subgraph G^t in Algorithm 4. Following Algorithm 4, U contains f, g, h, e and L consists of edge (e, e) because

Algorithm 4: Check whether $G[S]$ is an LDS of G

```
1  Function IsLDS(S, φ̄, φ, G = (V, E)):
2      Q ← an empty queue, ρ ← density(G[S]);
3      U ← ∅, L ← ∅, needFlow ← False;
4      foreach u ∈ S do
5          if u ∉ U then push u to Q, insert u into U;
6          while Q is not empty do
7              v ← pop out the front vertex in Q;
8              foreach (v, w) ∈ E do
9                  if w ∉ U then
10                     if φ(w) > ρ then
11                         add edge (v, v) to L, needFlow ← True;
12                     else if φ̄(w) > ρ then
13                         push w to Q, add w into U;
14     if not needFlow then return True;
15     G^t ← (U, E(U) ∪ L);
16     G' ← all ρ-compact subgraphs in G^t via min-cut;
17     return G[S] is a connected component in G';
```

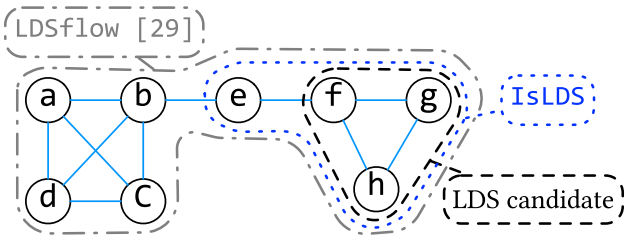

Fig. 7 LDS verification illustration

$\underline{\rho}(\mathtt{b}) > 1 = \mathsf{density}(G[S])$. Hence, G^t in our `IsLDS` contains 4 vertices and 5 edges, while the traversed subgraph in `LDSflow` [46] contains all eight vertices (i.e., `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`) as shown in Fig. 7, because core numbers of all vertices are larger than 1, i.e., $\mathsf{density}(G[S])$.

Theorem 5.1 *Given a graph G and a subgraph $G[S]$, where $G[S]$ is the DS of itself, $G[S]$ is an LDS of G if and only if Algorithm 4 returns* True.

Proof First, if $G[S]$ is an LDS, Algorithm 4 returns True. Because only the loops in L might increase the compact numbers in G^t compared to the compact numbers in G. Hence, $G[S]$ is still an LDS in G^t. Otherwise the maximal $\mathsf{density}(G[S])$-compact subgraph containing $G[S]$ must contain a vertex u with self loop, and then we can construct a larger $\mathsf{density}(G[S])$-compact subgraph in G by including vertices with $\underline{\phi}(w) > \mathsf{density}(G[S])$ connected to u. Hence, the contradiction proves the claim.

On the other direction, if $G[S]$ is not an LDS of G, we will find a larger $\mathsf{density}(G[S])$-compact subgraph containing $G[S]$ in G^t. Hence, $G[S]$ is also not an LDS in G^t. Thus, the algorithm returns False. □

Algorithm 5: Our LDS algorithm, `LDScvx`

```
Input  : A graph G = (V, E) and two integers k and N
Output : LDSs with top-k densities
1  preprocess G by pruning vertices via core numbers;
2  G' ← G;
3  stk ← an empty stack;
4  while k > 0 do
5      (r, α) ← Frank-Wolfe(G', N);
6      S, φ̄, φ ← ExtractSG(G', r, α);
7      S ← Pruning(G', S, φ̄, φ);
8      foreach S ∈ S reversely do push S into stk;
9      S ← pop out the top stable group from stk;
10     if IsDensest(G[S]) then
11         if IsLDS(S, φ̄, φ, G) then output G[S], k ← k − 1;
12         if stk is empty then break;
13         S ← pop out the top stable group from stk;
14     G' ← G[S];
```

By now, we have introduced all building blocks of our LDS algorithm. In the next subsection, we will present our LDS algorithm `LDScvx` by combining these components.

5.4 The Overall Algorithm `LDScvx`

Combining Algorithms 1 to 4 with reference to Fig. 4, we will obtain our LDS algorithm, named `LDScvx` in Algorithm 5.

In `LDScvx`, we first assign G to G' (line 1) and initialize an empty stack stk (line 2). Next, we extract the stable groups from the graph G' via `Frank-Wolfe`, `ExtractSG`, and `Pruning` (lines 4-6). Then, the algorithm pushes the stable groups in $\mathcal{S}$ reversely into stk (line 7). For stable groups in stk, the corresponding $\underline{\phi}$ value is decremented from top to bottom. Afterward, the first stable group in stk, which is also the one with the highest $\underline{\phi}$ value, is poped out (line 8) and is examined by `IsDensest` and `IsLDS` (line 9–10). If $G[S]$ is an LDS, we output it and decrease k by 1 (line 10). If $G[S]$ is not an LDS but is the DS of itself, we update S as the top stable group from stk (line 12). Next, we assign $G[S]$ to G' for the next iteration (line 13). The above process is repeated until top-k LDSs are found (line 3), or the stack is empty (line 11).

Next, we use an example to explain further the overall procedure of `LDScvx` (Algorithm 5).

Example 5.4 We still use the graph G in Fig. 1 as the example. Suppose we want to find top-2 LDSs from G. Assume we obtain $(\boldsymbol{r}^*, \boldsymbol{\alpha}^*)$ in Table 2 after `Frank-Wolfe`. Then, we will obtain the stable groups shown in Fig. 5, as well as the upper and lower bounds of compact numbers via `ExtractSG`. Next, in the `Pruning` process, the vertices in S_3 and S_4 will be pruned according to Corollary 5.1 and Lemma 5.2, which means that $\mathcal{S}$ contains S_1 and S_2. Afterward, S_2 and S_1 will be pushed into the stack stk. Now, the stable groups in stk satisfy that the compact numbers

of vertices in the stable group higher in stk are larger than those in the stable group lower in stk. Next, we pop out the top stable group S_1 from stk and verify that it is an LDS via `IsDensest` and `IsLDS`. We output $G[S_1]$ as the first LDS, pop out S_2 from stk, and repeat the above process. After S_2 is verified as an LDS, stk is empty, we break while loop (line 10 in Algorithm 5). In the end, we obtain two LDSs, $G[S_1]$ and $G[S_2]$.

Complexity. The time complexity of `LDScvx` is $O((N_{\mathsf{FW}}+N_{\mathsf{SG}})\cdot(n+m)+N_{\mathsf{Flow}}\cdot t_{\mathsf{Flow}})$, where N_{FW} is number of iterations that `Frank-Wolfe` needs, and $N_{\mathsf{SG}}\leq n$ is the number of stable groups in total, N_{Flow} is number of times `IsLDS` and `IsDensest` are called, and t_{Flow} denotes the time complexity of max-flow computation. Note that an iteration in `Frank-Wolfe` and verifying a stable group in `ExtractSG` both take $O(n+m)$ time cost. The memory complexity is $O(n+m)$.

6 The LTDS Problem and Our Solution

Samusevich et al [49] extended the LDS model from edge-based density to triangle-based density, termed locally triangle-densest subgraph (LTDS), and proposed a max-flow-based solution, named `LTDSflow`. We now study the LTDS problem and show how our previous `LDScvx` can be adapted.

6.1 The LTDS Problem

Following the classic definition [20, 53, 56, 59, 63], the triangle-based density of a graph $G=(V,E)$, denoted by $\mathsf{tr\text{-}density}(G)$, is defined as:

$$\mathsf{tr\text{-}density}(G)=\frac{|T|}{|V|}, \tag{3}$$

where T is the set of triangles in G. Based on the triangle-based density, we formally introduce the definition of LTDS below.

Definition 6.1 (ρ-tr-compact [49]) A graph $G=(V,E)$ is ρ-tr-compact if and only if G is connected, and removing any subset of vertices $S\subseteq V$ will result in the removal of at least $\rho\times|S|$ triangles in G, where ρ is a nonnegative real number.

Definition 6.2 (Maximal ρ-tr-compact subgraph [49]) A ρ-tr-compact subgraph $G[S]$ of G is a maximal ρ-tr-compact subgraph of G if and only if there does not exist a supergraph $G[S']$ of $G[S]$ with $S\subset S'$ in G such that $G[S']$ is also ρ-tr-compact.

Definition 6.3 (Locally triangle-densest subgraph [49]) A subgraph $G[S]$ of G is a locally triangle-densest subgraph (LTDS) of G if and only if $G[S]$ is a maximal $\mathsf{tr\text{-}density}(G[S])$-tr-compact subgraph in G.

Definitions 6.1 to 6.3 naturally follow from Definitions 3.1 to 3.3, indicating that LTDS shares many properties with LDS. We list some important properties in Lemma 6.1. This lemma ensures that LTDS is indeed the triangle-densest in its local region.

Lemma 6.1 *Given a graph $G=(V,E)$, an LTDS $G[S]$ of G has following properties:*

1. *any subgraph $G[S']$ of an LTDS $G[S]$ can not have a larger tr-density than $G[S]$;*
2. *any supergraph $G[S']$ of an LTDS $G[S]$, $G[S']$ is not ρ-tr-compact for any $\rho\geq\mathsf{tr\text{-}density}(G[S])$.*

Proof Please refer to the appendix. □

Problem 6.1 (LTDS problem [49]) Given a graph G and an integer k, the LTDS problem is to compute the top-k LTDS's with the largest density in G.

Example 6.1 (LTDS) Consider the graph G shown in Fig. 1. The subgraph $G[S_1]$ with tr-density $\frac{10}{3}$ is a maximal $\frac{10}{3}$-tr-compact subgraph. Hence, $G[S_1]$ is an LTDS. Similarly, $G[S_2]$ with tr-density 2 is also an LTDS as it is a maximal 2-tr-compact subgraph. The subgraph $G[S_3]$ with tr-density $\frac{1}{2}$ is a $\frac{1}{2}$-tr-compact subgraph. But $G[S_3]$ is not an LTDS because it is contained in $G[S_1\cup S_3]$ which is also $\frac{1}{2}$-tr-compact. $G[S_1\cup S_3]$ is also not an LTDS, because its tr-density is $\frac{11}{5}$ but it is not a $\frac{11}{5}$-tr-compact subgraph. The tr-compactness of $G[S_1\cup S_3]$ is $\frac{1}{2}=\frac{2}{4}$, because removing S_3 from $G[S_1\cup S_3]$ will result in removing of 2 triangles.

6.2 Our LTDS Algorithm

Frank-Wolfe-based algorithm. Eq. (4) presents the convex program for triangle-densest subgraphs [53]. This formulation is a natural extension of Eq. (2). The key difference between Eq. (4) and Eq. (2) lies in their objectives: the former focuses on distributing the weights of triangles, whereas the latter focuses on the weights of edges. Similarly, the Frank-Wolfe-based algorithm for Eq. (4) is also a straightforward generalization of Algorithm 1. The pseudo-code is presented in Algorithm 6.

Algorithm 6: Frank-Wolfe-based algorithm [15]

Function `Frank-Wolfe`$(G = (V, E, T), N \in \mathbb{Z}_+)$**:**
  foreach $t \in T$ **do** $\alpha_{u,t}^{(0)} \leftarrow \frac{1}{3}, \forall u \in t$;
  foreach $u \in V$ **do** $r_u^{(0)} \leftarrow \sum_{t \in T \wedge u \in t} \alpha_{u,t}^{(0)}$;
  for $i = 1, \ldots, N$ **do**
    $\gamma_i = \frac{2}{i+2}$;
    foreach $u \in t$ **do**
      $\hat{\alpha}_{u,t} \leftarrow 1$ if $u = x$ and 0 otherwise
    $\boldsymbol{\alpha}^{(i)} \leftarrow (1 - \gamma_t) \cdot \boldsymbol{\alpha}^{(i-1)} + \gamma_i * \hat{\boldsymbol{\alpha}}$;
    foreach $u \in V$ **do** $r_u^{(i)} \leftarrow \sum_{t \in T \wedge u \in t} \alpha_{u,t}^{(i)}$;
  return $(\boldsymbol{r}^{(i)}, \boldsymbol{\alpha}^{(i)})$;

$$
\begin{aligned}
\mathsf{CP}(G) \qquad & \min \sum_{u \in V} r_u^2 \\
& r_u = \sum_{u \in t, t \in T} \alpha_{u,t}, && \forall u \in V \\
& \sum_{u \in t} \alpha_{u,t} \geq 1, && \forall t \in T \\
& \alpha_{u,t} \geq 0, && \forall u \in V, \forall t \in T
\end{aligned} \tag{4}
$$

To bridge LTDS and Eq. (4), we introduce the definition of tr-compact number, generalized from Definition 4.1.

Definition 6.4 (Tr-compact number) Given an undirected graph $G = (V, E)$, the tr-compact number of each vertex $u \in V$, denoted by $\phi_{tr}(u)$, is the largest ρ, such that u is contained in a ρ-tr-compact subgraph of G.

Theorem 6.1 *Any connected subgraph $G[S]$ is an LTDS in G if and only if the following conditions are satisfied:*

1. $\forall u \in S$, $\phi_{tr}(u) = \mathsf{tr\text{-}density}(G[S])$;
2. $\forall (u, v) \in E$, *such that* $u \in S$ *and* $v \in V \setminus S$, *it holds that* $\phi_{tr}(u) > \phi_{tr}(v)$.

Proof Please refer to the appendix. □

Theorem 6.2 *Suppose* $(\mathbf{r^*}, \boldsymbol{\alpha}^*)$ *is an optimal solution of Eq. (4). Then, each* r_u^* *in* $\mathbf{r^*}$ *is exactly the tr-compact number of* u, *i.e.,* $\forall u \in V$, $r_u^* = \phi_{tr}(u)$.

Proof Please refer to the appendix. □

Theorem 6.1 enables us to efficiently extract and verify LTDS's using tr-compact numbers, while Theorem 6.2 ensures that Algorithm 6 can correctly compute feasible tr-compact numbers.

Extract stable groups and pruning. Inspired by the idea of k-core [51], Samusevich et al proposed tr-k-core to optimize the pruning process of `LTDSflow`.

Definition 6.5 (tr-k-core and tr-core number [49]) The tr-k-core of G is the maximal subgraph $G[S]$ such that for any $u \in S$, u is contained in at least k triangles. For any $u \in V$, the *tr-core number* of u, denoted by $\mathsf{tr\text{-}core}_G(u)$, is the largest k such that u is contained in the tr-k-core of G.

We borrow the idea of tr-core numbers to compute the initial bounds for tr-compact numbers. After initialization, Definition 5.1 and `ExtractSG` can be directly adapted to the triangle-based setting by substituting edge weights with triangle weights and replacing edge-based density with triangle-based density. Definition 6.6 gives a generalized definition of Definition 5.1.

Definition 6.6 (Triangle-based stable groups) Given a feasible solution $(\boldsymbol{r}, \boldsymbol{\alpha})$ to Eq. (4), stable group $S \in V$ with respect to $(\boldsymbol{r}, \boldsymbol{\alpha})$ is a non-empty subset if the following conditions hold:

1. For any $v \in V \setminus S$, r_v satisfies either $r_v > \max_{u \in S} r_u$ or $r_v < \min_{u \in S} r_u$;
2. For any $t \in T$, if $t \cap S \neq \emptyset$, $\sum_{r_u > \max_{v \in S} r_v} \alpha_{u,t} = 0$;
3. For any $t \in T$, if $t \cap S \neq \emptyset$, $\sum_{r_u < \min_{v \in S} r_v} \alpha_{u,t} = 1$.

The modified `ExtractSG` provides tighter bounds for tr-compact numbers compared to tr-core numbers [49]. The correctness of the bounds is guaranteed by Lemma 6.2. Similarly, `Pruning` can also be adapted.

Lemma 6.2 *Given a feasible solution* $(\boldsymbol{r}, \boldsymbol{\alpha})$ *to Eq. (4) and a triangle-based stable group S w.r.t.* $(\boldsymbol{r}, \boldsymbol{\alpha})$, *for all* $u \in S$, *we have that* $\min_{v \in S} r_v \leq \phi_{tr}(u) \leq \max_{v \in S} r_v$.

Proof sketch. We can prove the lemma by generalizing Lemma 5.1 for supporting triangles. □

Extract and verify LTDS. To verify the candidates returned by `Pruning`, we need to examine them from two perspectives based on the properties in Lemma 6.1. Since `IsDensest` effectively verifies self-densest property, the only remaining task is to determine whether the candidate subgraph $G[S]$ is a maximal $\mathsf{tr\text{-}density}(G[S])$-tr-compact subgraph. To efficiently solve this problem, we propose a different flow network construction method, named `ExtMaximal`, and a new LTDS algorithm, named `IsLTDS`, based on the original `IsLDS`, to address these issues.

Algorithm 7 presents our verification process for LTDS. From a high-level perspective, `IsLTDS` performs a breadth-first search starting from the candidate subgraph $G[S]$, aiming to identify all vertices and triangles that could be part of a $\mathsf{tr\text{-}density}(G[S])$-tr-compact subgraph. Afterward, with the help of `ExtMaximal`, we further verify whether $G[S]$ is maximal.

Specifically, an empty queue Q, an empty vertex set U, empty pattern sets P, $T_{visited}$ and tr-density ρ are first initialized (lines 2-3). Here, U and P are used to collect all vertices and triangles that would be included in a ρ-tr-compact subgraph, while $T_{visited}$ is used to avoid repeated triangle visits

Algorithm 7: Check whether $G[S]$ is an LTDS of G

```
1  Function IsLTDS(S, φ̄_tr, φ_tr, G = (V, E)):
2      Q ← an empty queue, ρ ← tr-density(G[S]);
3      U ← ∅, P ← ∅, T_visited ← ∅, needFlow ← False;
4      foreach u ∈ S do
5          push u to Q, insert u into U;
6      while Q is not empty do
7          v ← pop out the front vertex in Q;
8          foreach t ∈ T ∧ v ∈ t ∧ t ∉ T_visited do
9              valid ← True, T_visited ← T_visited ∪ t;
10             foreach w ∈ t do
11                 if φ̄_tr(w) < ρ then valid ← False;
12             if not valid then continue;
13             M ← ∅;
14             foreach w ∈ t do
15                 if φ_tr(w) ≤ ρ then
16                     add w into M;
17                     if w ∉ U then push w to Q, add w into U ;
18                 else
19                     needFlow ← True;
20             if M ≠ ∅ then P ← P ∪ {M};
21         foreach (v, w) ∈ E do
22             if w ∉ U then
23                 if φ_tr(w) > ρ then needFlow ← True;
24                 else if φ̄_tr(w) > ρ then
25                     push w to Q, add w into U;
26     if not needFlow then return True;
27     S′ ←ExtMaximal(U, P, ρ);
28     return G[S] is a connected component in G[S′]
```

Algorithm 8: Extract maximal ρ-tr-compact subgraphs

```
1  Function ExtMaximal(S, P, ρ):
2      initialize a flow network F = (V, E) with a source vertex s
         and a sink vertex t;
3      foreach u ∈ S do
4          add a node u into V;
5          add an edge (u, t) with capacity ρ into E;
6      foreach p ∈ P do
7          add a node p into V;
8          add an edge (s, p) with capacity 1 into E;
9          foreach u ∈ p do
10             add an edge (p, u) with capacity 1 into E;
11     S, T ← minimal s-t cut in F;
12     return (S ∩ S) \ s;
```

during the traversal. We use an additional boolean parameter $needFlow$ to indicate whether the flow-based function ExtMaximal should be executed.

Then, we push each vertex $u \in S$ into Q (lines 4-5). Next, we pop out the front vertex v from Q and begin our twofold verification (lines 7-25). In the first aspect, we search every unvisited triangle containing v to find valid triangles.

A valid triangle consists of three vertices, each with an upper bound of the tr-compact number greater than or equal to ρ.

Here, only if the triangle is valid, it can be part of the maximal tr-density$(G[S])$*-tr-compact subgraph, which helps reduce the search range and computational costs.* For each valid triangle t, we extend the vertex set U by adding vertices $w \in t$ that satisfy $\underline{\phi}_{tr}(w) \leq \rho$. A new vertex set M is maintained to store these vertices. If M is not empty, we add it to the pattern set P to extend it. *Vertices with* $\underline{\phi}_{tr}(w) > \rho$ *in* t *are excluded here to speed up the maximality verification. The proof of correctness can be found in the appendix.* In the second aspect, IsLTDS iterates all neighbors of v to further extend U (lines 21-25). If IsLTDS does not encounter a vertex with a lower bound of the tr-compact number greater than ρ, we directly return True (line 26). Otherwise, we must construct a well-designed flow network and use the max-flow algorithm to find the maximal ρ-tr-compact subgraph from $G[U]$. Finally, IsLTDS returns True if $G[S]$ is a connected component of $G[S']$.

Algorithm 8 describes our new flow network construction method. The intuition behind ExtMaximal is to distribute the weights of each valid triangle to the vertices in U, which is quite similar to the intuition behind Eq. (4). Specifically, each pattern $p \in P$ is connected to the source vertex s through an edge with capacity 1, representing the weight of a triangle. At the same time, all vertices in p are also connected to p with a capacity of 1. Moreover, each vertex is linked to the sink vertex t with a capacity of ρ. The correctness of ExtMaximal and IsLTDS is stated in Lemma 6.3 and Theorem 6.3, respectively.

Lemma 6.3 *Given a tr-density value ρ, along with the corresponding vertex set U and pattern set P generated in lines 6-25 of Algorithm 7, Algorithm 8 returns all vertices u whose tr-compact number is larger than or equal to ρ.*

Proof Please refer to the appendix. □

Theorem 6.3 *Given a graph G and a subgraph $G[S]$, where $G[S]$ is the TDS of itself, $G[S]$ is an LTDS of G if and only if Algorithm 7 returns* True.

Proof Please refer to the appendix. □

The overall algorithm LTDScvx. The workflow of LTDScvx can be naturally extended from LDScvx.

7 A Unified Framework for LDS Problem

Zhou et al [64] proposed a unified framework for the densest subgraph discovery (DSD) problem. This framework

Algorithm 9: The unified framework for LDS problem.

Input : A graph $G = (V, E)$ and an integer k
Output : Top-k valid subgraph

1 $stk \leftarrow$ an empty stack, $\boldsymbol{w} \leftarrow \emptyset, \overline{\phi} \leftarrow \emptyset, \underline{\phi} \leftarrow \emptyset$;
// The initial reduction method.
2 $V', E', \overline{\phi}, \underline{\phi} \leftarrow$ IR(G);
3 push V' into stk;
4 **while** $k > 0 \wedge stk \neq \emptyset$ **do**
5 $S \leftarrow$ pop out the first subset from stk;
// (1) The vertex weight updating method.
6 $\boldsymbol{w} \leftarrow$VWU($G$,$\boldsymbol{w}$, $\overline{\phi}$, $\underline{\phi}$);
// (2) The graph reduction and division method.
7 $S, stk, \overline{\phi}, \underline{\phi} \leftarrow$GRD($G$, $\boldsymbol{w}$, stk);
// (3) The candidate subgraph extraction and verification method.
8 $k \leftarrow$CSEV(G, S, k);

consists of three stages: *Graph Reduction*, *Vertex Weight Updating* (VWU), and *Candidate Subgraph Extraction and Verification* (CSEV). Inspired by this idea, we consolidate all LDS and LTDS solutions (LDSflow [46], LTDSflow [49], LDScvx, and LTDScvx) into a similar framework. Unlike the unified framework in [64], which was designed for the classical DSD problem, we further extend this paradigm to support both LDS and LTDS problems. Specifically, our Algorithm 9 introduces two new components, IR and GRD, and generalizes the original three-stage process to handle both edge and triangle-based local densities.

Algorithm 9 depicts this framework, including four components: *Initial Reduction* (IR), *Vertex Weight Updating* (VWU), *Graph Reduction and Division* (GRD), and *Candidate Subgraph Extraction and Verification* (CSEV). Specifically, given a graph G and an integer k, we initialize the upper and lower bounds of compact (resp. tr-compact) numbers using core (resp. tr-core) numbers (line 2). We then iteratively execute operations in the following three stages (lines 4-8):

1. Update the vertex weight vector $\boldsymbol{w}$ for each vertex;
2. Prune invalid vertices via upper and lower bounds of ϕ, update bounds of ϕ in residual graph, and divide the large graph into multiple smaller subgraphs for further computation;
3. Extract the candidate subgraph using vertex weight vector $\boldsymbol{w}$, and verify if the candidate subgraph meets the requirements.

We outline the four components of different algorithms in Table 3. For example, for our LDScvx, it uses k-core to initialize bounds of ϕ. Then, for the repeated searching process: **Stage (1)** employ Frank-Wolfe to optimize Eq. (2); **Stage (2)** prune invalid vertices based on Corollary 5.1 and Lemma 5.2, while adopting stable group to tighten the bounds and decompose the problem into multiple sub-problems; **Stage (3)** extract the top stable group as a candidate, and execute IsDensest and IsLDS to verify its optimality.

Comparison between the max flow-based and CP-based algorithms. With the help of our unified framework, we can thoroughly compare our method and the max flow-based ones. For the IR component, they share the same initialization. For VWU stage, the vertex weight vector $\boldsymbol{w}$ represents the weights assigned to different vertices. It holds different meanings and serves different purposes in different algorithms:

- In the max-flow-based algorithms, $\boldsymbol{w}$ denotes the flows from the vertices to the target node;
- In our CP-based algorithms, $\boldsymbol{w}$ represents the weight sum received by each vertex.

This stage reflects the main difference between these two types of algorithms. For GRD stage, with the help of stable groups, our CP-based algorithms provide *tighter* bounds via convex programming, which enables more vertices to be pruned and allows a division of the original graph. Furthermore, in the CSEV stage, LDScvx and LTDScvx construct *smaller* flow networks to verify the optimality of the candidate by exploiting tighter bounds.

Comparison between our framework and the DSD framework [64]. First, we divide the *Graph Reduction* stage into two parts, IR and GRD, since these two parts serve different purposes and utilize different notions, such as k-core and stable groups, in the LDS problem and its variant. In contrast, for the DSD problem, k-core is the only concept used in this stage. Second, since the LDS and LTDS problems aim to identify multiple dense regions in the given graph, both reduction and division are necessary. However, the DSD problem focuses solely on the densest part, making reduction the only essential step.

8 Experiments

8.1 Setup

We use thirteen real datasets [6, 7, 37, 47, 62] which are publicly available[1] except for TL. The TL dataset is provided by a television company TCL Technology. The dataset contains film information provided on its smart TV platform, mainly used for a case study. Other graph datasets cover various domains, including social networks (e.g., LiveJournal), e-commerce (e.g., Amazon), and video platforms (e.g.,

[1] https://networkrepository.com/, https://snap.stanford.edu/data/index.html, and https://law.di.unimi.it/datasets.php

Table 3 Overview of the four components of LDS/LTDS solutions.

Method	IR	Stage (1): VWU	Stage (2): GRD	Stage (3): CSEV
LDSflow	k-core	compute the maximum flow	❶ prune via Corollary 5.1 and Lemma 5.2; ❷ no updating; ❸ extract the residual graph.	❶ extract the minimum cut;
LTDSflow	tr-k-core		❶ no pruning; ❷ no updating; ❸ extract the residual graph.	❷ verify locally densest property.
LDScvx	k-core	optimize CP formulation	❶ prune via Corollary 5.1 and Lemma 5.2; ❷ update bounds via stable group;	❶ extract the top stable group; ❷ verify self-densest property;
LTDScvx	tr-k-core		❸ divide the graph via stable group.	❸ verify locally densest property.

Table 4 Graphs used in our experiments

Dataset	Category	$\|V\|$	$\|E\|$	$\|T\|$
PG	Social	10.7K	24.3K	54.8K
BK	Social	58.2K	214K	495K
TL	Movie	108K	168K	–
GW	Social	196K	950K	2.27M
AM	E-commerce	335K	926K	667K
YT	Video-sharing	1.13M	2.99M	3.05M
SK	Communication	1.70M	11.1M	28.8M
LJ	Social	4.00M	34.7M	–
OR	Social	3.07M	117M	–
IC	Web	7.41M	194M	–
AB	Web	22.7M	639M	–
IT	Web	41.3M	1.03B	–
LK	Hyperlink	52.6M	1.61B	–

YouTube). Table 4 summarizes the statistics. For datasets not involved in the LTDS-related experiments, the number of triangles is omitted (marked as "–") since $|T|$ is irrelevant to the corresponding edge-density setting. In addition, we further conduct experiments on four more datasets to validate the generality of our methods. Due to space limitations, their detailed statistics and results are presented in Appendix C.

We compare the following algorithms:

- LDScvx is our convex-programming based top-k LDS algorithm (Section 5.4).
- LDSflow [46] is the state-of-the-art top-k LDS algorithm based on max-flow.
- LTDScvx is our convex-programming based top-k LTDS algorithm (Section 6).

Table 5 Relative running time w.r.t. different N

N	50	100	150	200
LDScvx	1.64	1.10	1.12	1.20
LTDScvx	1.54	1.42	1.27	1.24

- LTDSflow [49] is the state-of-the-art top-k LTDS algorithm based on max-flow.

All the algorithms above are implemented in C++ with STL used, except for LDSflow, whose source codes are provided by the authors of [46]. In addition to these, we also conduct supplementary experiments comparing our methods with LDS-opt [55] and IPPV [61]; see Appendix B for detailed results. We run all the experiments on a machine having an Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz processor and 256GB memory, with Ubuntu installed.

8.2 Overall Evaluation of LDS and LTDS Algorithms

Setting of N. To choose the best setting of N, i.e., the number of Frank-Wolfe iterations, we tested the running time of LDScvx w.r.t. different values of N from 50 to 200 with k fixed to 5. Table 5 reports the average relative running time w.r.t. N over different datasets. The relative running time for a specific value of N on each dataset is obtained via dividing the running time by the minimum running time over all N values for the dataset. The mean is then obtained by averaging over all datasets. We can find that when $N = 100$ (resp. $N = 200$), we obtain the minimum average relative running time. Hence we use $N = 100$ (resp. $N = 200$) as the default parameter value in other experiments.

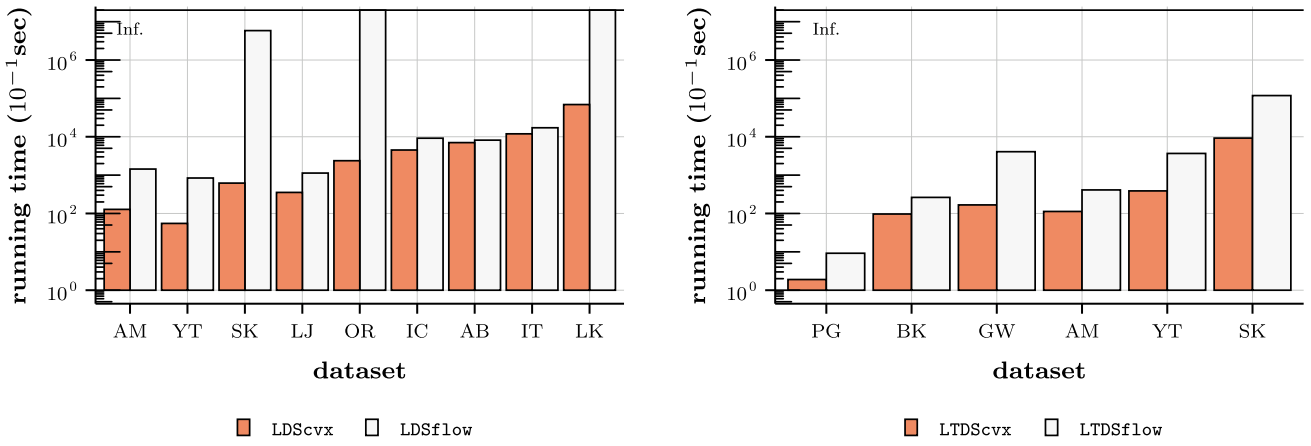

Fig. 8 Efficiency of LDScvx and LDSflow with $k = 5$

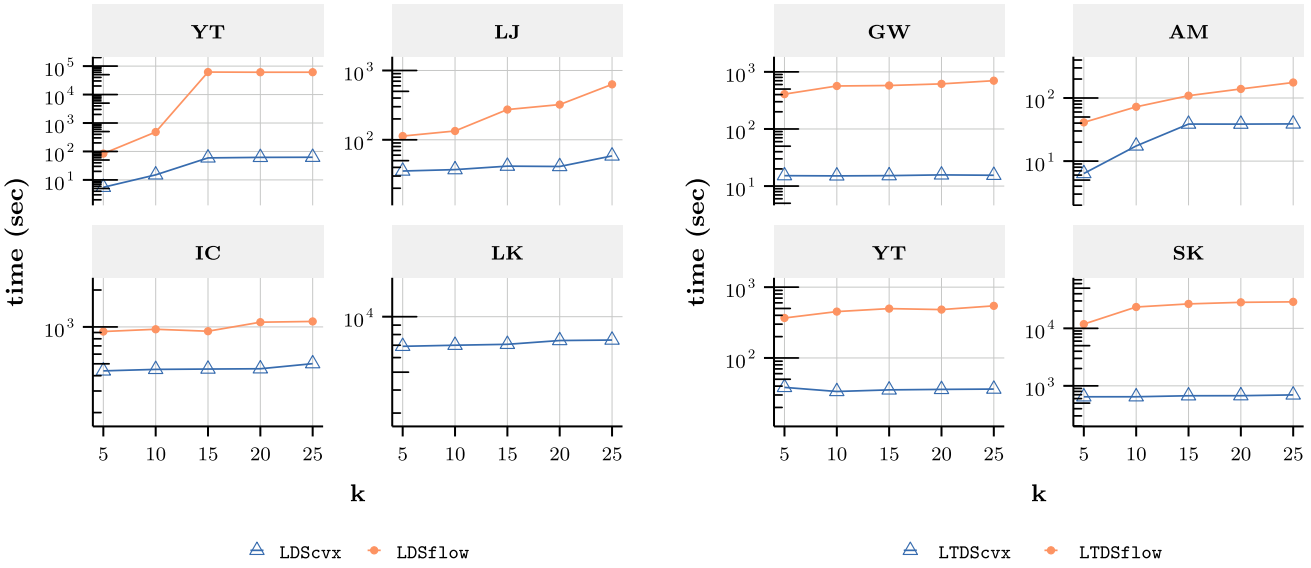

Fig. 9 Efficiency of LTDScvx and LTDSflow with $k = 5$

Evaluation of efficiency. In this experiment, we compare our top-k LDS algorithm LDScvx with the state-of-the-art LDSflow [46] w.r.t. running time.

We first fix $k = 5$ to overview the two algorithms on different datasets. For the edge-based setting, we conduct experiments on the nine largest datasets. In contrast, for the triangle-based setting, due to its higher computational complexity, we test on six smaller datasets, excluding TL, as it contains no triangles. Fig. 8 shows the efficiency results of the two algorithms. The datasets are ordered by graph size on the x-axis. Note that for some datasets, the bars of LDSflow touch the solid upper line, which means LDSflow cannot finish within 600 hours on those datasets. From Fig. 8, we can observe: first, the running time of LDSflow increases along with the graph size increasing; second, LDScvx is up to four orders of magnitude faster than LDSflow. For the LTDS algorithms, a similar trend can be observed in Fig.

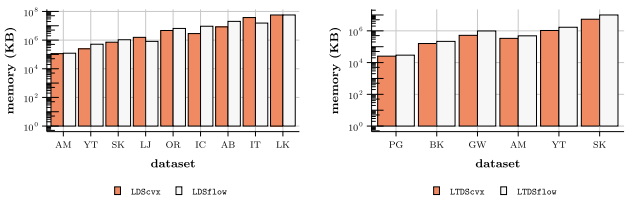

Fig. 10 Efficiency of LDScvx and LDSflow w.r.t. different k

9: LTDSflow generally requires significantly more time than LTDScvx, and LTDScvx is up to more than 20 times faster than LTDSflow on certain datasets. We reckon that the speedup comes from more vertices pruned due to tighter bounds, fewer candidates verified, and smaller flow networks as stated in Section 5.

We further provide the running time trends of these algorithms w.r.t. different k values on four representative datasets

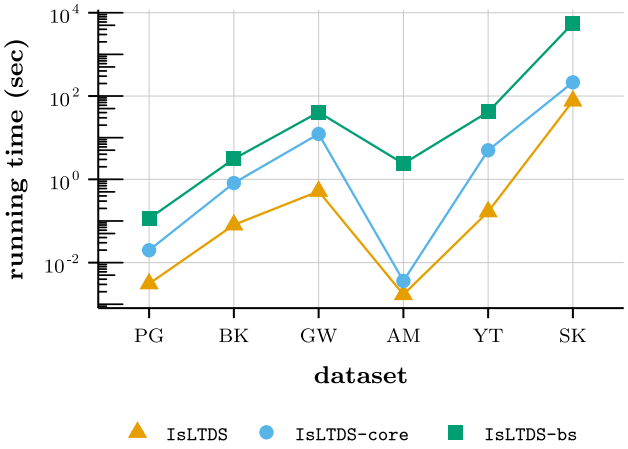

Fig. 11 Efficiency of LTDScvx and LTDSflow w.r.t. different k

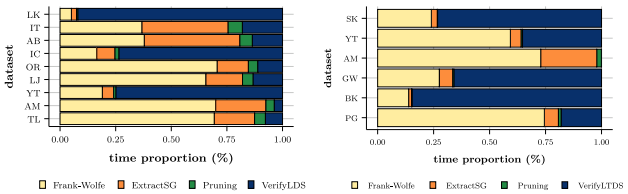

Fig. 12 Memory usage of LDS algorithms with $k = 5$

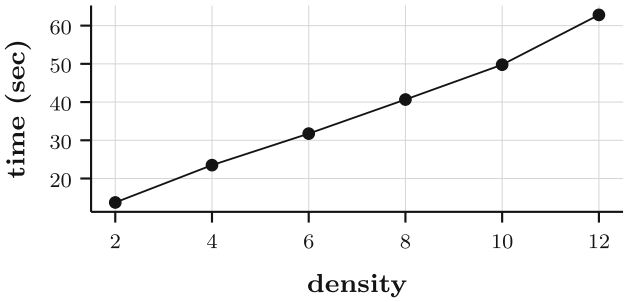

Fig. 13 Memory usage of LTDS algorithms with $k = 5$

Table 6 Numbers of failed LDS candidates on YT w.r.t. k.

Algorithm	$k = 10$	$k = 15$	increased times
LDScvx	37	84	2.27×
LDSflow	277	18399	66.42×

Table 7 Numbers of failed LTDS candidates on AM w.r.t. k.

Algorithm	$k = 10$	$k = 15$	increased times
LDScvx	0	0	—
LDSflow	0	0	—

in Figs. 10 and 11 due to space limit. For LDSflow, we do not show its trends on dataset LK because it cannot finish within 600 hours even with $k = 5$. We can see that when k increases, the running time of all algorithms increases. Moreover, the growth rates of LDScvx's and LTDScvx's running times with respect to k is generally smaller than those of LDSflow and LTDSflow.

Memory usage. Further, we test the memory usage of these algorithms. The maximum memory usage is tested via the Linux command `/usr/bin/time -v`. For the cases that LDSflow does not finish reasonably, we record the maximum resident memory during the running process. Fig. 12 and Fig. 13 report the maximum memory usage of these algorithms. The datasets are sorted on the x-axis in ascending order of graph size. We can observe that the memory usages of all algorithms increase along with the increasing graph size. Besides, the memory costs of convex-programming-based algorithms and max-flow-based algorithms are around the same scale because all algorithms take linear memory usage w.r.t. the graph size.

8.3 Comparison Under the Unified Framework

In this experiment, we comprehensively compare all algorithms under our summarized unified framework for LDS and LTDS problems. Since the IR methods are nearly identical across all algorithms, and the VWU methods are difficult to compare in isolation, we omit the discussion of these two components.

Evaluation of GRD methods. The primary difference between the GRD methods used in our convex-programming-based algorithms and those in existing max-flow-based solutions lies in the way LDScvx and LTDScvx derive stable groups to provide tighter bounds, thereby enabling more effective vertex pruning and a more efficient decomposition of the original graph.

To analyze the priority of stable groups, we focus on two special cases shown in Fig. 10 and Fig. 11:

- In Fig. 10, there is a significant increase in LDSflow (about two orders of magnitude) when k is increased from 10 to 15 on dataset YT;
- In Fig. 11, the running time of LTDScvx on dataset AM increases more noticeably with k compared to other datasets.

Here, we use the number of failed LDS and LTDS candidates as the evaluation metric for the GRD method, since better pruning and problem decomposition would lead to fewer but more accurate verifications.

We report the numbers of failed LDS candidates on YT with $k = 10$ and 15 for both algorithms in Table 6. We observe that the number of failed candidates in LDSflow increased around 66× when k increases from 10 to 15, which explains the surge of running time in Fig. 10. In contrast,

the number of failed candidates in LDScvx only increases by about 2×. This is why the running time of LDScvx does not increase much when k is increased from 10 to 15. Another observation from Table 6 is that the failed numbers for LDScvx are smaller than LDSflow on both k values, respectively. The reason is that we provide tight upper and lower bounds for compact numbers via convex programming and stable groups, and the tight bounds further enable more vertices to be pruned, which results in fewer LDS candidates to be examined. Apart from that, to examine an LDS candidate, our LDScvx only needs a subgraph of what is needed in LDSflow to calculate the max-flow by leveraging the lower bounds of compact numbers, according to Section 5.3. We believe the above two improvements explain why we are around three orders of magnitude faster on YT when $k = 15$.

We further analyze the special case observed on the AM dataset in Fig. 11, where the running time of LTDScvx increases more noticeably with k compared to other datasets. We report the numbers of failed LTDS candidates on AM with $k = 10$ and 15 for both algorithms in Table 7. Specifically, we find that when $k = 25$, the number of failed LTDS candidates for both LTDScvx and LTDSflow drops to zero. This indicates that the top-25 locally triangle-dense regions in AM can already be effectively pruned and decomposed using the tr-k-core number, without requiring the additional guidance from stable groups. As a result, the tighter bounds provided by our GRD method do not further contribute to vertex pruning or problem division in this case. This explains why LTDScvx does not exhibit a clear efficiency advantage and its running time increases with k, similar to LTDSflow.

These two special cases demonstrate that the efficiency of LDScvx and LTDScvx heavily depend on the effectiveness of the GRD method, particularly in providing tighter bounds for pruning and division. When the bounds are already tight (e.g., due to strong core structure), the advantage of stable groups diminishes, leading to a higher growth rate as k increases.

Evaluation of CSEV methods. Our main optimization for the CSEV method is that we construct smaller flow networks to verify the optimality of candidates. Here, we conduct an ablation study on LDScvx and LTDScvx to understand the effectiveness of IsLDS (Algorithm 4) and IsLTDS (Algorithm 7).

Recall that in IsLDS we only include the vertices satisfying $\overline{\phi}(u) \geq \mathsf{density}(G[S]) \geq \underline{\phi}(u)$ into the flow network computation, while its counterpart in LDSflow [46] includes all vertices satisfying $\overline{\phi}(u) \geq \mathsf{density}(G[S])$, named by IsLDS-core. We report the time used for verifying LDSs with IsLDS and IsLDS-core, respectively, on the nine datasets when $k = 5$ in Table 8. Here, the time of IsLDS-core is measured by replacing IsLDS with IsLDS-core in LDScvx. The time of IsLDS-core on LK is marked as ≥ 259200s because it cannot finish within three days. From the results, we observe that the verification process with IsLDS is up to 110× faster than that with IsLDS-core.

For the LTDS problem, we compare the following three verification strategies:

- IsLTDS: our proposed method, which includes only the vertices satisfying $\overline{\phi}_{tr}(u) \geq \mathsf{tr\text{-}density}(G[S]) \geq \underline{\phi}_{tr}(u)$ in the flow network construction;
- IsLTDS-core: a generalization of IsLDS-core to the triangle-based setting, including all vertices satisfying $\overline{\phi}_{tr}(u) \geq \mathsf{tr\text{-}density}(G[S])$;
- IsLTDS-bs (baseline) [49]: the method adopted in LTDSflow, which includes all vertices in G for verification.

We report the running time of the three verification methods on six datasets with $k = 5$ in Fig. 14. As shown in the figure, IsLTDS consistently achieves the lowest verification cost across all datasets. In particular, on large graphs such as SK and YT, IsLTDS reduces the verification time by up to two orders of magnitude compared to IsLTDS-bs, and is significantly faster than IsLTDS-core as well. This improvement is attributed to the tight upper and lower bounds derived from the stable group, which help exclude many irrelevant vertices from the flow network construction. As a result, IsLTDS performs verification over a much smaller subgraph, leading to substantial efficiency gains. Moreover, compared to IsLTDS-core, which only applies the upper bound constraint, IsLTDS leverages both bounds to further reduce the search space, especially when the bounds are tight. The comparison on AM further highlights this: while IsLTDS is significantly faster than the other two methods on most datasets, it shows little advantage over IsLTDS-core on AM. This observation is consistent with the special case discussed earlier, where we showed that all top-k candidates on AM can already be effectively pruned using the tr-k-core number alone. In such cases, the additional bounds provided by the stable group in IsLTDS do not further reduce the search space, making its verification cost comparable to that of IsLTDS-core.

8.4 Time Proportion Analysis

Here, we evaluate how the different building blocks of LDScvx and LTDScvx contribute to the whole running time after the graph is loaded and preprocessed. Fig. 15 reports the proportion of each part in the total running time for LDScvx: Frank-Wolfe (Algorithm 1), ExtractSG (Algorithm 2), Pruning (Algorithm 3), and VerifyLDS (Algorithm 4) with $k = 5$ over nine datasets. We can observe that the Frank-Wolfe computation is the most computationally expensive part on most datasets. For SK, LK and YT datasets, the time used

Table 8 Effect of `IsLDS` with $k = 5$

Dataset	`IsLDS`	`IsLDS-core`	Speedup
AM	0.3334s	0.3623s	1.09×
YT	2.6575s	80.9994s	30.48×
SK	58.7789s	18035.1864s	306.83×
LJ	2.1204s	2.3924s	1.13×
OR	18.4089s	723.6035s	39.31×
IC	285.4502s	288.9184s	1.01×
AB	60.2669s	62.0416s	1.03×
IT	147.9361s	188.8527s	1.28×
LK	2335.4461s	≥ 259200s	≥ 110.99×

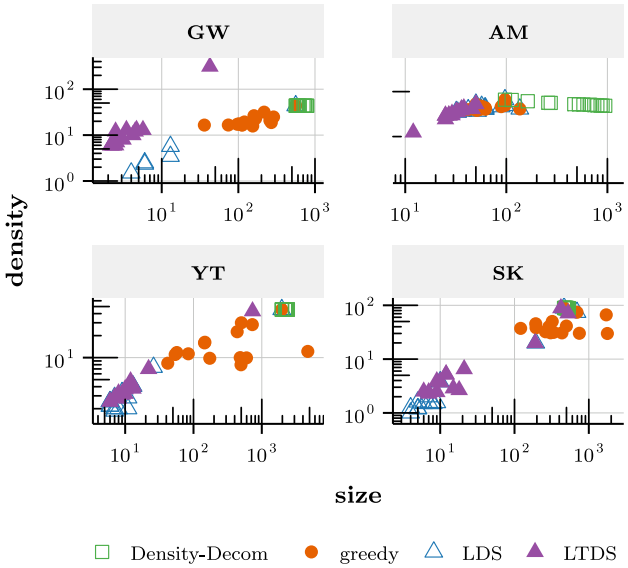

Fig. 14 Effect of `IsLTDS` with $k = 5$

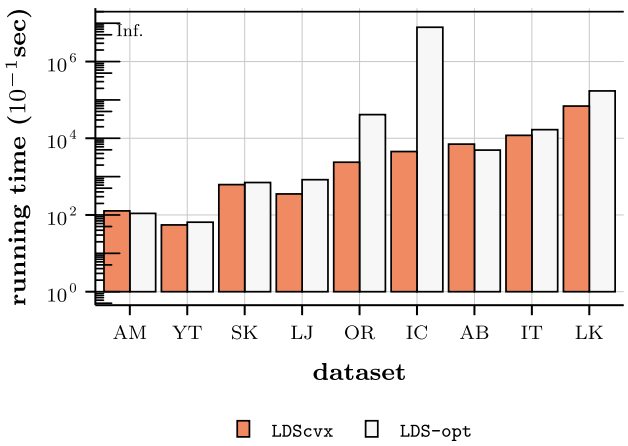

Fig. 15 Proportion of each part in total running time for `LDScvx`

by verifying LDS takes the majority. We further examine the number of failed LDS candidates (i.e., on which `IsLDS` returns False) on the nine datasets. Table 9 reports the results.

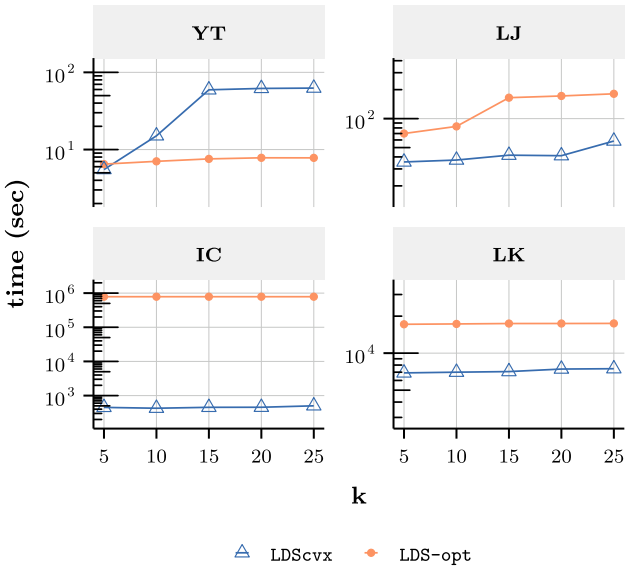

Fig. 16 Proportion of each part in total running time for `LTDScvx`

Table 9 Numbers of failed LDS candidates with $k = 5$.

Dataset	AM	YT	SK	LJ	OR	IC	AB	IT	LK
#failed	0	9	67	1	1	0	0	0	6

Table 10 Numbers of failed LTDS candidates with $k = 5$.

Dataset	PG	BK	GW	AM	YT	SK
#failed	0	8	18	0	7	21

We can find that the numbers of failed candidates on these datasets are much higher than other datasets, which means that these two datasets need more time to verify LDSs, which explains the results in Fig. 15 to some extent. For IC, the time used by verifying LDS is also high, because the first LDS in IC is quite large and takes relatively long time to verify.

For `LTDScvx`, the overall time breakdown in Fig. 16 is similar to that of `LDScvx`, with Frank-Wolfe and `VerifyLTDS` being the most time-consuming components across most datasets. The correlation between verification time and the number of failed candidates is further supported by Table 10, which shows that datasets with more failed candidates also exhibit higher `VerifyLTDS` proportions.

8.5 Scalability

We further tested the scalability w.r.t. the density via synthetics datasets. The six synthetic graphs are generated by Barabási-Albert (BA) model [1]. The numbers of vertices in all synthetic graphs are fixed to 1,000,000, and the densities are increased linearly from 2 to 12. In other words, the numbers of edges are from 2,000,000 to 12,000,000. We report

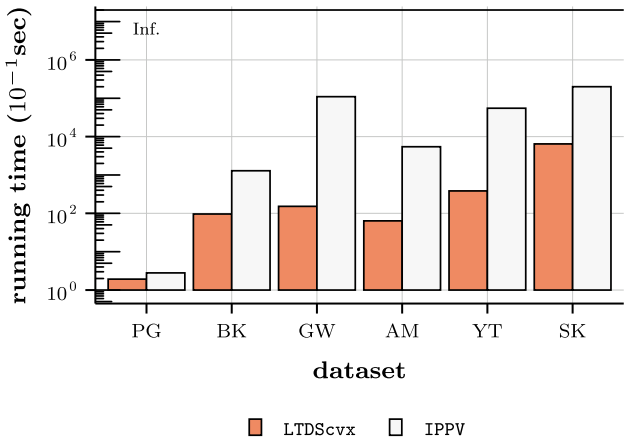

Fig. 17 The scalability w.r.t. the density

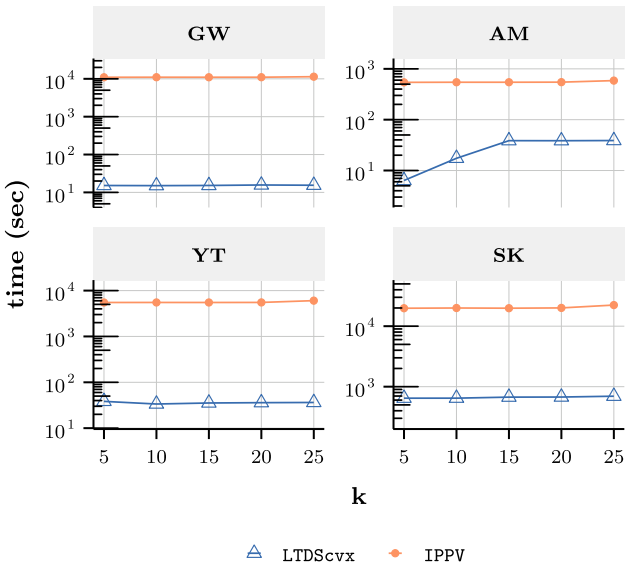

Fig. 18 Subgraph statistics: density w.r.t. size

the running time of LDScvx on the six synthetic datasets in Fig. 17. We observe that LDScvx scales well w.r.t. the graph density.

8.6 Subgraph Statistics

Fig. 18 reports the densities of the top-15 densest subgraphs w.r.t. the size (number of vertices) returned by four different models on four datasets, where Greedy iteratively computes a densest subgraph and removes it from the graph, and FDS denotes the density-friendly decomposition model [15, 54]. From Fig. 18, we can find that the densest subgraph can be found by all three algorithms, except LTDS, because the densest subgraph is also an LDS. But there are some dense subgraphs found by Greedy that do not qualify as LDSs. Hence, the subgraphs found by LDScvx have a wide range of densities and sizes. For FDS, we can find that the subgraphs have increasing sizes and decreasing densities. This is because FDS outputs a chain of subgraphs, where each subgraph is nested within the next one, and the inner one is denser than the outer ones. In contrast, LTDS operates under a triangle-based density setting, which evaluates subgraphs based on their triangle concentration rather than edge count. As a result, the subgraphs returned by LTDScvx tend to be smaller in size but possess high triangle densities. This difference explains why LTDS may not always identify the same densest subgraphs found by edge-based models, especially in cases where edge density is high but triangle concentration is relatively low.

8.7 Case Study

Here, we perform a case study on the TL dataset. The TL dataset is provided by a Chinese television company, TCL Technology, which contains three types of vertices: director, movie, and actor.

After examining the top-10 LDSs returned by our LDS algorithm, we found that the LDSs on the TL dataset are about different topics. For example, Fig. 2 shows the LDS with the third-highest density. This LDS contains eight movies related to a famous Japanese sci-fiction series "Ultraman" with four actors and one director. In this LDS, the four actors participated in all eight films, and the director directed five of them. The LDS with the second-highest density is a subgraph about Chinese martial fiction. Other LDSs cover topics about western films, Danish comedies, and cartoons, while the DS model can only find a large subgraph about western films. Hence, from the output of the case study, we reckon that the subgraphs returned by the LDS algorithm are good representations of *different* local dense regions in the graph.

9 Conclusion

In this paper, we study the problem of finding the top-k locally densest subgraphs (LDS's) in a graph for identifying the local dense regions from the graph. The LDS's are usually compact and dense. To facilitate the LDS discovery, we propose a new concept, named *compact number* for each vertex, which denotes the compactness of the most compact subgraph containing the corresponding vertex. By leveraging the compact number and its relations with the LDS problem and a specific convex program, we derive a convex-programming based algorithm LDScvx following the pruning-and-verify paradigm. For pruning, we derive tight upper and lower bounds of compact numbers and design powerful pruning techniques based on the bounds. For verification, we minimize the size of the subgraph needed for max-flow computation via further exploiting the lower

bounds of compact numbers. Extensive experiments on six datasets show that LDScvx is up to four orders of magnitude faster than the state-of-the-art algorithm. The case study performed on a movie dataset reveals that the output subgraphs by our LDS algorithm are high-quality representations of local dense regions of the graph.

In the future, it is interesting to see whether the LDS model can be extended to other graph types, e.g., bipartite graphs and directed graphs. Besides, extending the edge-based density to clique-based density for the LDS problem is also worth investigating.

A Additional Proofs

Proof of Lemma 6.1 These two properties can be proved by contradiction. For the first property, suppose the tr-density of $G[S]$ is equal to ρ, and $G[S']$ is a subgraph of $G[S]$ with a larger tr-density ρ'. Then removing $S \setminus S'$ from $G[S]$ would result in the removal of $|T(G[S])| - |T(G[S'])| = \rho|S| - \rho'|S'| < \rho|S \setminus S'|$ triangles, which contradicts Definition 6.1. For the second property, this can be directly derived from Definition 6.2. □

Proof of Theorem 6.1. We first explore the relationship between the LTDS and the tr-compact numbers of vertices either within or adjacent to the LTDS through the following claims. □

Claim 1 Given an LTDS $G[S]$ in G, $\forall u \in S$, we have $\phi_{tr}(u) = \text{tr-density}(G[S])$.

Proof We prove the claim by contradiction. Let ρ denote the value of $\text{tr}-\text{density}(G[S])$. Since $G[S]$ is ρ-tr-compact, $\forall u \in S$, $\phi_{tr}(u) \geq \rho$. Suppose there exists a vertex $v \in S$ with $\phi_{tr}(v) > \rho$. Then, we can find a $\phi(v)$-tr-compact subgraph $G[S']$ that contains v. Because $G[S]$ is a maximal ρ-tr-compact subgraph and $S' \cap S \neq \emptyset$, we have $S' \subseteq S$, which contradicts with Lemma 6.1. □

Claim 2 Given an LTDS $G[S]$ in G, $\forall (u, v) \in E$, if $u \in S$ and $v \in V \setminus S$, we have $\phi_{tr}(u) > \phi_{tr}(v)$.

Proof We prove the claim by contradiction. Let ρ denote the value of $\text{tr}-\text{density}(G[S])$. Suppose there exists an edge $(u, v) \in E$ such that $u \in S, v \in V \setminus S$, and $\phi_{tr}(u) \leq \phi_{tr}(v) = \rho$. Then we can find a ρ-tr-compact subgraph $G[S']$ of G that contains v. $G[S \cup S']$ is a connected supergraph of g and is also $\phi_{tr}(u)$-tr-compact, which contradicts with Lemma 6.1. □

The sufficiency direction of Theorem 6.1 can be concluded from Lemma 6.1, while the necessity direction directly follows from Claims 1and 2.

Proof of Theorem 6.2 We use the following claims to formally explain the ideal weight distribution of Eq. (4). □

Claim 3 Suppose $(\mathbf{r}^*, \boldsymbol{\alpha}^*)$ is an optimal solution of Eq. (4).For a vertex $u \in V$, let $X = \{v \in V | r_v^* > r_u^*\}$, $Y = \{v \in V | r_v^* = r_u^*\}$, $Z = \{v \in V | r_v^* < r_u^*\}$. The optimality of $\mathbf{r}^*$ implies that:

1. $\forall (a, b, c) \in T \cap (X \times X \times Y)$, $r_a^* > r_c^*$, $r_b^* > r_c^*$, and $\alpha^*_{c,(a,b,c)} = 1$.
2. $\forall (a, b, c) \in T \cap (X \times Y \times Y)$, $r_a^* > r_b^* = r_c^*$, and $\alpha^*_{b,(a,b,c)} + \alpha^*_{c,(a,b,c)} = 1$.
3. $\forall (a, b, c) \in T \cap (Z \times Z \times Y)$, $r_a^* < r_c^*$, $r_b^* < r_c^*$, and $\alpha^*_{a,(a,b,c)} + \alpha^*_{b,(a,b,c)} = 1$.
4. $\forall (a, b, c) \in T \cap (Z \times Y \times Y)$, $r_a^* < r_b^* = r_c^*$, and $\alpha^*_{a,(a,b,c)} = 1$.

Proof We prove *1* by contradiction: Suppose $\exists t \in T \cap (X \times X \times Y)$, $r_a^* > r_c^*$, $r_b^* > r_c^*$, $\alpha_{c,(a,b,c)} < 1$. Since $\alpha_{a,t} + \alpha_{b,t} + \alpha_{c,t} = 1$, we have $\alpha_{a,t} + \alpha_{b,t} > 0$. Obviously there exists some nonnegative ϵ and μ, so that $\alpha'_{a,t} = \alpha_{a,t} - \epsilon$, $\alpha'_{b,t} = \alpha_{b,t} - \mu$, and $\alpha'_{c,t} = \alpha_{c,t} + \epsilon + \mu$, satisfies $\alpha'_{a,t} + \alpha'_{b,t} + \alpha'_{c,t} = 1$ and $\alpha'_{a,t}, \alpha'_{b,t}, \alpha'_{c,t} \in [0, 1]$. We change $\alpha_{a,t}$, $\alpha_{b,t}$, and $\alpha_{c,t}$ into $\alpha'_{a,t}, \alpha'_{b,t}$, and $\alpha'_{c,t}$, respectively. After such modifications, the objective function will change $\epsilon \cdot (-2r_a^* + 2r_c^* + 2\epsilon + \mu) + \mu \cdot (-2r_b^* + 2r_c^* + 2\mu + \epsilon)$. We can always find $\epsilon > 0$ or $\mu > 0$, which satisfies $2r_a^* - 2r_c^* > 2\epsilon + \mu$, and $2r_b^* - 2r_c^* > 2\mu + \epsilon$, to strictly decrease the objective function. This contradicts that $\mathbf{r}^*$ is the optimal solution to Eq. (4). Equation 2-4 can be proved similarly. □

Claim 3 shows the optimal weight distribution of different kinds of triangles, i.e., each triangle distributes all its weight to vertices with the lowest r^* value.

Claim 4 Suppose $(\mathbf{r}^*, \boldsymbol{\alpha}^*)$ is an optimal solution of $CP(G)$. For a vertex $u \in V$, each connected component of $G[X \cup Y]$ is r_u^*-tr-compact.

Proof According to Claim 3, $|T(G[X \cup Y])| = |T \cap (X \times X \times X)| + |T \cap (X \times X \times Y)| + |T \cap (X \times Y \times Y)| + |T \cap (Y \times Y \times Y)| = \sum_{u \in X \cup Y} r_u^*$, and removing any subset $S \subseteq X \cup Y$ will result in the removal of at least $r_u^* \times |S|$ triangles, this is because

$$
\begin{aligned}
\sum_{t \in T(G[X \cup Y]) \wedge t \cap S \neq \emptyset} 1 &= \sum_{t \in T(G[X \cup Y]) \wedge t \cap S \neq \emptyset} \sum_{v \in t} \alpha_{v,t} \\
&\geq \sum_{t \in T(G[X \cup Y]) \wedge t \cap S \neq \emptyset} \sum_{v \in t \wedge v \in S} \alpha_{v,t} \\
&= \sum_{v \in S} r_v^* \\
&\geq r_u^* \times |S|
\end{aligned}
$$

Hence, the claim holds. □

Based on Claim 3, Claim 4 builds an initial relationship between tr-compactness and Eq. (4). Now, we begin to prove Theorem 6.2. Claim 4 guarantees that any $u \in V$ is contained in at least one r_u^*-tr-compact subgraph. For any other subgraph $G[S] \nsubseteq G[X \cup Y]$ containing u, $G[S]$ is a ϕ-tr-compact subgraph, where $\phi \leq r_u^*$. Clearly, $S \cap (Y \cup Z) \neq \emptyset$. Removing $S \cap (Y \cup Z)$ from $G[S]$ will result in the removal of no more than $r_u^* \times |S \cap (Y \cup Z)|$ triangles. Hence, the theorem holds.

Proof of Lemma 6.2 Based on Definition 6.6, we prove $\phi_{tr}(u) \geq \min_{v \in S} r_v$ by contradiction. According to Theorem 6.2, $\forall u \in V$, $\phi_{tr}(u) = r_u^*$. Suppose there exists a vertex $u \in S$ such that $r_u^* = \phi_{tr}(u) < \min_{v \in S} r_v \leq r_u$. According to Definition 6.6, we have $\sum_{v \in S} r_v = \sum_{v \in S} r_v^*$, so there must exist another vertex $x \in S$ such that $r_x^* = \phi(x) > r_x$, which implies $r_x^* \geq \min_{v \in S} r_v > r_u^*$.We can choose ϵ such that $0 < \epsilon < r_x^* - r_u^*$. If we increase r_u^* by ϵ, and decrease r_x^* by ϵ, the objective value $||\boldsymbol{r}^*||_2^2$ will decrease by $2\epsilon(\epsilon + r_u^* - r_x^*)$. This contradicts that $\mathbf{r}^*$ is the optimal solution to Eq. (4), so $\phi_{tr}(u) \geq \min_{v \in S} r_v$. $\phi_{tr}(u) \leq \max_{v \in S} r_v$ can be proved in the same way. □

Proof of Lemma 6.3 For the sake of the proof, we introduce the following definitions and notations. For a given set of vertices S', let $t_i(S')$ be the number of patterns that involve exactly i vertices not from S', $i \in \{1, 2, 3\}$, and t_u be the number of patterns that involve vertex u. Let $A = \mathcal{S} \cap S$.

We begin the proof with a structural claim for the optimal min-cut of the flow network $\mathcal{F}$ in Algorithm 8. □

Claim 5 Consider any min-cut $(\mathcal{S}, \mathcal{T})$ in the network $\mathcal{F}$, the cost of the min-cut is equal to $|P| - t_0(A) + \rho|A|$.

Proof For patterns satisfying $p \cap A = p$, node p corresponding to the specific pattern p has to be in $\mathcal{S}$. If not, then we could reduce the cost of the min-cut by moving p to $\mathcal{S}$. There are six additional cases we considered, one per each type of pattern with respect to set A.

1. $p = \{u, v, w\} \in P, u, v, w \notin A$.
2. $p = \{u, v, w\} \in P, u, v \notin A, w \in A$.
3. $p = \{u, v, w\} \in P, u \notin A, v, w \in A$.
4. $p = \{u, v\} \in P, u, v \notin A$.
5. $p = \{u, v\} \in P, u \notin A, v \in A$.
6. $p = \{u\} \in P, u \notin A$.

For cases 3, 5, 6, node p corresponding to this specific p can be either in $\mathcal{S}$ or $\mathcal{T}$, and the cost of edges from source s to $\mathcal{T} \cap S$ is equal to $t_1(A)$. For cases 2, 4, node p corresponding to this specific p has to be in $\mathcal{T}$, and the cost of edges from source s to $\mathcal{T} \cap S$ is equal to $t_2(A)$. For case 1, node p corresponding to this specific p has to be in $\mathcal{T}$, and the cost of edges from source s to $\mathcal{T} \cap S$ is equal to $t_3(A)$. Furthermore, the cost of edges from $\mathcal{S}$ to sink t equals $\rho|A|$. Summing up these cost terms, the total cost is equal to $t_1(A) + t_2(A) + t_3(A) + \rho|A| = |P| - t_0(A) + \rho|A|$. □

Next, we derive the following claim, which aims to build a relationship between S, P, and the original graph.

Claim 6 Removing any subset S' from S, such that $\forall u \in S'$, $\phi_{tr}(u) \geq \rho$, will result in the removal of at least $\rho|S'|$ patterns from P.

Proof According to Definition 6.4, $\phi_{tr}(u) \geq \rho$ suggests that u is contained in at least one ρ-tr-compact subgraph of G. For all triangles containing u and possibly contained in a ρ-tr-compact subgraph, we have generated a corresponding $p \in P$ in Algorithm 7. Suppose the claim holds to the contrary; this would contradict Definition 6.1. □

Claim 7 Given an optimal min-cut $(\mathcal{S}, \mathcal{T})$ in the network $\mathcal{F}$ that maximize $|A|$, vertex $u \in A$ if and only if $\phi_{tr}(u) \geq \rho$.

Proof Let B denote the vertex set of all vertices u such that $u \in S \cap \phi_{tr}(u) \geq \rho$. We prove that A and B are the same.

(1) We prove that $A \subseteq B$. Suppose to the contrary that some vertices $u \in A$ have a tr-compact number smaller than ρ, then according to Claim 6,there exists a subset $S' \subseteq A$ such that removing S' from A would result in the removal of less that $\rho|S'|$ patterns. Thus, we have $(|P| - t_0(A \setminus S') + \rho|A \setminus S'|) - (|P| - t_0(A) + \rho|A|) = t_0(A) - t_0(A \setminus S') - \rho|S'| < 0$, which contradicts that $(\mathcal{S}, \mathcal{T})$ is optimal.

(2) We prove that $B \subseteq A$. Suppose to the contrary that B is not a subset of A. According to *(1)*, we can derive that $A \subsetneq B$. According to Claim 6, removing $B \setminus A$ from B would result in the removal of at least $\rho|B \setminus A|$ patterns. Thus, we have $(|P| - t_0(B) + \rho|B|) - (|P| - t_0(A) + \rho|A|) = t_0(A) - t_0(B) + \rho|B \setminus A| <= 0$. To maximize $|A|$, $B \setminus A$ has to be $\emptyset$, which contradicts our assumption.

According to *(1)* and *(2)*, the claim is proved. □

Lemma 6.3 follows directly from Claim 7.

Proof of Theorem 6.3 First, if $G[S]$ is an LTDS, Algorithm 7 returns True. Because according to Lemma 6.3, Algorithm 8 returns all vertices $u \in U$ such that their tr-compact number is larger than or equal to ρ. $G[S]$ is a connected component in $G[S']$ suggests that $G[S]$ is a maximal ρ-tr-compact subgraph in G. Otherwise the maximal density$(G[S])$-tr-compact subgraph containing $G[S]$ must contain a vertex u with corresponding patterns, and then we can construct a larger density$(G[S])$-tr-compact subgraph in G by including vertices with $\underline{\phi}_{tr}(w) >$ density$(G[S])$ connected to u. Hence, the contradiction proves the claim.

On the other direction, if $G[S]$ is not an LTDS of G, we will find a larger density$(G[S])$-tr-compact subgraph containing $G[S]$. Thus, the algorithm returns False. □

B Comparison with LDS-opt and IPPV

In this appendix, we provide a detailed comparison between our proposed methods and two representative algorithms in the literature – LDS-opt [55] and IPPV [61]. For clarity, we decompose both LDS-opt and IPPV under our unified framework, which consists of four components–IR, VWU, GRD, and CSEV. Table 11 summarizes how each method instantiates these stages, highlighting their key algorithmic differences.

B.1 Comparision with LDS-opt

LDS-opt is a verification-free approach for identifying top-k LDSs, which builds upon the concept of locally dense subgraphs introduced by Tatti and Gionis [54]. Specifically, Trung et al [55] proved that the maximal λ-compact subgraphs of a graph G correspond to the connected components of certain locally dense subgraphs of G. This correspondence enables a hierarchical construction of all maximal λ-compact subgraphs, from which LDSs can be directly obtained as the leaf nodes of the hierarchy–thus avoiding the costly max-flow verification required in LDSflow, particularly for relatively large k values.

Comparison under the unified framework. The main distinctions between LDS-opt and our proposed method lie in the VWU, GRD, and CSEV components, while both methods share a similar initialization step (IR). In VWU, our method optimizes the convex programming formulation to update vertex weights, whereas LDS-opt still requires maximum flow computations to discover the compact parts, which is the same as LDSflow. In GRD, LDS-opt adopts locally dense subgraphs [54] to perform graph division, whereas our method relies on the notion of stable groups. Our stable groups are conceptually related to the stable subsets introduced by Danisch et al [15], which were defined in the convex programming formulation of locally dense subgraph decomposition. Both concepts identify vertex sets that are locally optimal under the convex relaxation of density. However, stable subsets impose a stricter condition that no vertex outside the subset can have a larger r value than any vertex inside, thus often merging several locally tight regions into one. In contrast, our stable groups relax this constraint and allow multiple disjoint regions with locally maximal r values to coexist. Hence, a stable subset can be viewed as the union of several stable groups with the highest r values. This finer granularity enables our method to obtain tighter local bounds and a more precise graph division, which in turn leads to smaller flow networks and reduced verification costs. Compared to LDS-opt, which leverages locally dense subgraph structures for graph decomposition, our stable-group-based algorithm achieves both stronger theoretical tightness and higher empirical scalability, as we will discuss later. Finally, in CSEV, although LDS-opt avoids explicit max-flow verification and is therefore considered verification-free, it still checks the locally dense property–an intrinsic condition of locally dense subgraphs [54]–which, however, remains a lightweight and flow-free operation in our framework.

Experimental Comparison. Empirical evaluations further reveal distinct behavioral and performance characteristics between the two approaches: Fig. 19 presents the efficiency of LDScvx and LDS-opt[2] on nine datasets. These two methods demonstrate comparable performance on seven datasets, with speedups ranging from 0.70× to 2.49×. However, on the OR and IC datasets, our method achieves significantly higher efficiency, outperforming LDS-opt by 17.4× and 1740×, respectively. This superior performance mainly comes from our use of stable groups for graph division in the GRD stage. By enabling finer-grained decomposition, stable groups can more quickly locate promising candidates and substantially reduce the scale of the flow network required for verification. This design effectively narrows the gap in verification overhead between our method and the verification-free approach, while simultaneously improving the efficiency of both the candidate extraction and VWU stages.

Fig. 20 demonstrates the performance of LDScvx and LDS-opt on four datasets with k ranging from 5 to 25. While our method remains efficient on three datasets, it is noticeable that LDS-opt performs particularly well on the YT dataset. This observation is consistent with the claim in [55], where LDS-opt shows better scalability for larger k values. This advantage arises from its verification-free design and hierarchical search structure, which allows it to avoid repeated verification for overlapping local regions when k increases. In contrast, our LDScvx still conducts verification via flow network, leading to slightly higher overhead when the number of candidate subgraphs grows.

B.2 Comparision with IPPV

Comparison under the unified framework. The IPPV method [61] was recently proposed for efficiently detecting locally h-clique densest subgraphs (LhCDS), a generalization of the locally densest subgraph (LDS) problem from edge-based density to h-clique densities. IPPV jointly considers the h-clique compact number and the LhCDS definition to design a multi-stage detection pipeline. Specifically, it first derives upper and lower bounds of clique compact numbers, then applies a convex-programming-based refinement to tighten these bounds in the *ProposeCL* stage. Next, in the *Prune* stage, it employs tentative graph decomposition to handle overlapping cliques across subgraphs. Finally, the *VerifyLhCDS* stage introduces both basic and

[2] The implementation of LDS-opt is provided by the authors of [55].

Table 11 Overview of the four components of LDS-opt and IPPV

Method	IR	Stage (1): VWU	Stage (2): GRD	Stage (3): CSEV
LDS-opt	k-core	compute the maximum flow	❶ no pruning; ❷ update bounds via locally dense subgraph [54]; ❸ divide the graph via locally dense subgraph [54];	❶ extract the minimum cut; ❷ verify locally dense property.
IPPV	(k, ψ_h)-*core*	optimize CP formulation	❶ prune via Proposition 5 [61]; ❷ update bounds via stable h-clique group [61]; ❸ divide the graph via stable h-clique group [61];	❶ extract the top stable h-clique group [61]; ❷ verify self-densest property; ❸ verify locally densest property.

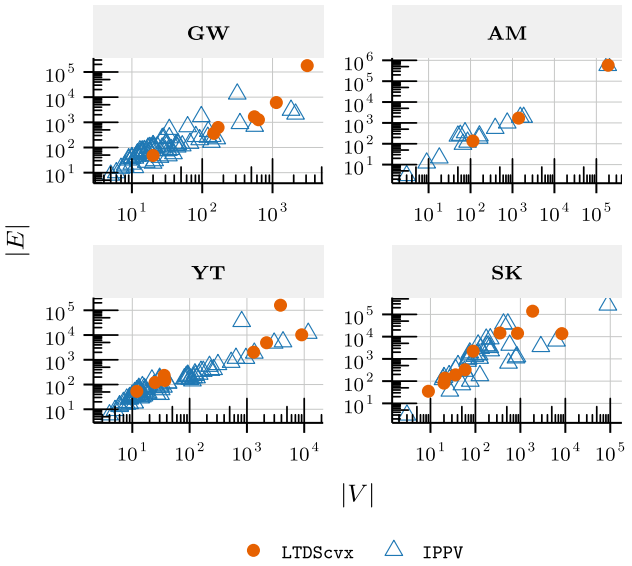

Fig. 19 Efficiency of LDScvx and LDS-opt with $k = 5$

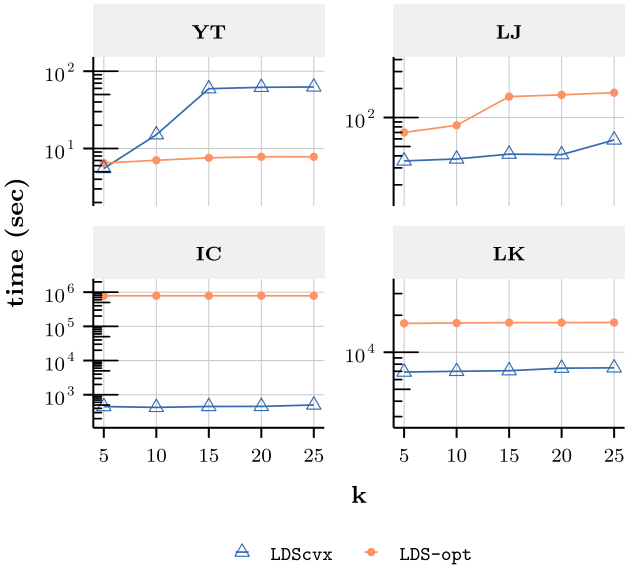

Fig. 20 Efficiency of LDScvx and LDS-opt w.r.t. different k values

fast verification schemes, where the fast version constructs smaller-scale flow networks to validate candidates with reduced cost. This design ensures that IPPV achieves both exactness and high efficiency in top-k LhCDS discovery. This method can also be naturally incorporated into our proposed unified framework, as shown in Table 11. Specifically, both our convex-programming-based methods and IPPV adopt Frank–Wolfe-style iterative optimization to refine vertex weights during the VWU stage. However, our proposed triangle-based *stable group* demonstrates significantly better empirical performance than the *stable h-clique group* adopted in IPPV [61], when h is fixed to 3, leading to substantially reduced computational cost and faster convergence in practice.

Experimental comparison. Fig. 21 and Fig. 22 compare the efficiency of LTDScvx and IPPV[3] with the value of h fixed to 3 on six datasets. As shown in Fig. 21, LTDScvx consistently outperforms IPPV across all datasets, achieving up to 722× speedup on GW, 143× on YT, and 85× on AM. Even on the smaller datasets such as PG and BK, our method remains competitive, with runtime reductions ranging from 1.4× to 13.4×. These results highlight the strong scalability of LTDScvx when compared to IPPV, whose running time grows rapidly on large graphs due to the cost of iteratively verifying local h-clique compactness and handling overlapping dense regions.

Fig. 22 further illustrates the effect of varying k from 5 to 25. While IPPV exhibits nearly constant runtime, our LTDScvx shows stable and significantly lower computation

[3] The implementation of IPPV is provided by the authors of [61].

cost across all k values. This advantage stems from the use of our *triangle-based stable group* strategy. By decomposing the graph more effectively, LTDScvx reduces the scale of the flow network needed for verification and locates candidates more precisely and efficiently.

We further compare the performance of the stable h-clique group used in IPPV and our proposed triangle-based stable group, as illustrated in Fig. 23. The results show that the subgraphs generated by LTDScvx after the first GRD stage are typically smaller and more compact than those produced by IPPV. This indicates that our triangle-based stable groups can partition the graph into finer-grained, density-aware components, which leads to more accurate candidate localization and substantially reduces the size of subsequent verification tasks. In contrast, the stable h-clique group in IPPV tends to form larger intermediate subgraphs due to the high overlap among h-cliques, resulting in heavier computation and redundant verification. Therefore, our LTDScvx achieves both higher efficiency and better scalability by tightening the decomposition granularity and minimizing unnecessary candidate propagation.

Remark We observe that in our experiments, IPPV actually runs even slower than the baseline LTDSflow, which appears inconsistent with the performance reported in [61]. However, we emphasize that the implementations of both IPPV and LTDSflow used in our evaluation are provided by the respective authors and are executed under their default parameter settings. This discrepancy may arise from differences in hardware environments, graph characteristics, or parameter sensitivity between the reported datasets and ours.

Limitation and potential extension. We acknowledge that one limitation of our current work is that LTDScvx has not been extended to support the general h-clique densest subgraph problem, as addressed by IPPV [61]. The key challenge in generalizing from the edge-based to the h-clique-based setting lies in effectively modifying our convex programming formulation for higher-order motifs. Among them, the triangle represents the simplest and most fundamental case of h-cliques, which already captures the essential computational and structural complexity of higher-order extensions. Although we do not explicitly present an algorithm for arbitrary h values, our method can be readily adapted to the h-clique scenario by redefining the compactness and stability conditions under higher-order interactions. Moreover, our empirical results demonstrate that LTDScvx significantly outperforms IPPV when $h = 3$, suggesting that our convex programming–based approach provides a promising and scalable foundation for general h-clique extensions in future work.

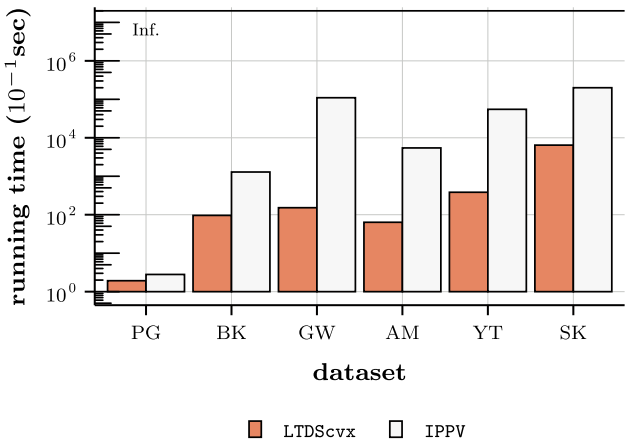

Fig. 21 Efficiency of LTDScvx and IPPV with $k = 5$

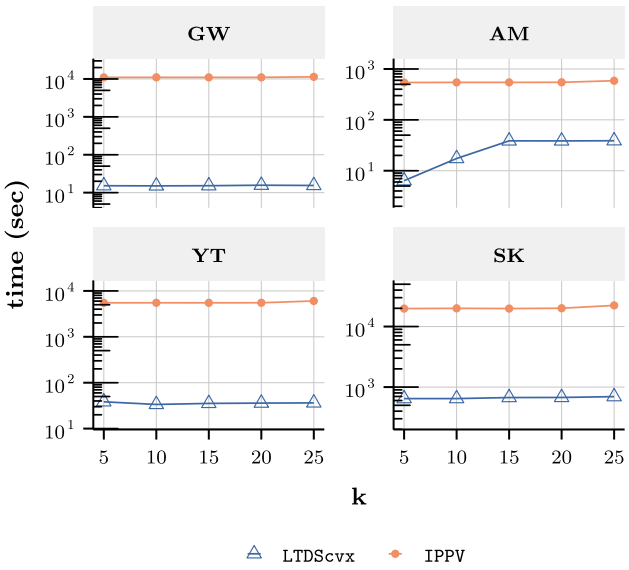

Fig. 22 Efficiency of LTDScvx and IPPV w.r.t. different k values

C Additional Experiments for LTDS Problem on More Datasets

To further examine the robustness and generalization ability of LTDScvx across diverse graph structures, we conduct additional experiments on four real-world datasets selected from [61]. As summarized in Table 12, these datasets vary significantly in both scale and density, covering multiple network types including social, communication, and collaboration graphs. This extended evaluation complements the main experiments, which focused on six representative datasets, and provides a more comprehensive assessment of the performance of LTDScvx on both sparse and dense real-world graphs. For completeness, we focus on comparing LTDScvx and LTDSflow, while the performance of IPPV can be found in its original paper [61].

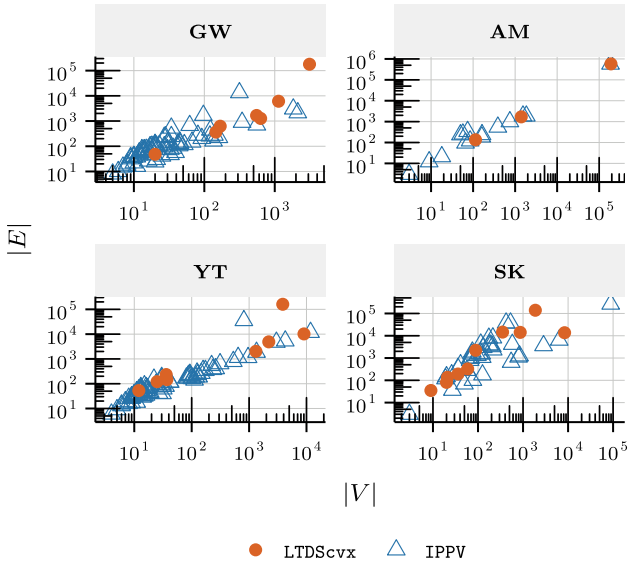

Fig. 23 Sizes of candidate subgraphs produced after the first GRD stage

Table 12 Additonal datasets.

Dataset	Category	$\|V\|$	$\|E\|$	$\|T\|$
PP	Social	5.91K	41.7K	174K
EN	Communication	36.7K	184K	727K
DB	Collaboration	317K	1.05M	2.22M
FX	Social	2.52M	7.92M	7.90M

As shown in Table 13, LTDScvx consistently outperforms LTDSflow across all four datasets and all tested k values. On relatively small social and communication networks such as PP and EN, LTDScvx achieves up to 15.5× and 4.4× speedups, respectively, mainly owing to the reduced number of max-flow invocations brought by our convex relaxation scheme. For the medium-scale collaboration graph DB, the advantage remains clear, with a stable improvement between 3× and 12×, demonstrating that the method scales robustly as the graph becomes denser. On the large-scale social graph FX, the improvement is particularly remarkable: LTDScvx achieves over two orders of magnitude faster runtime ($> 100\times$), highlighting its strong scalability and efficiency when dealing with massive networks containing millions of triangles. Overall, these results provide further evidence that the convex-programming formulation and the triangle-based stable group enable LTDScvx to maintain both efficiency and stability across diverse graph sizes and densities.

Acknowledgements Chenhao Ma was partially supported by NSFC under Grant 62302421, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2025A1515010439, and the Guangdong Provincial Key Laboratory of Big Data Computing, The Chinese University of Hong Kong, Shenzhen. Xiaolin han was supported by NSFC under Grant 62302397, the China Postdoctoral Science Foundation under Grant Number 2025M774364, and Shaanxi Post-doctoral Research Project (2025BSHSDZZ106). Reynold Cheng was supported by the Research Grant Council of Hong Kong (RGC Project HKU 17202325), the University of Hong Kong (Project 2409100399), and the HKU Faculty Exchange Award 2024 (Faculty of Engineering). Lakshmanan's research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada and an NSERC Alliance Grant.

Table 13 Efficiency comparison of LTDScvx and LTDSflow on four additional datasets

Dataset	k	LTDScvx (s)	LTDSflow (s)	Speedup
PP	5	0.94	8.55	9.10×
	10	0.81	8.95	11.05×
	15	0.56	8.66	15.46×
	20	0.86	8.65	10.06×
	25	0.85	8.66	10.19×
EN	5	10.54	33.52	3.18×
	10	9.50	38.49	4.05×
	15	9.57	40.17	4.20×
	20	9.68	40.98	4.23×
	25	9.73	42.75	4.39×
DB	5	20.89	76.52	3.66×
	10	22.11	130.34	5.90×
	15	22.20	161.06	7.25×
	20	23.15	222.42	9.61×
	25	24.19	290.94	12.03×
FX	5	27.54	3287.72	119.38×
	10	28.23	3344.07	118.46×
	15	27.79	3307.98	119.03×
	20	27.35	3320.51	121.41×
	25	28.92	3373.15	116.64×

References

1. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Rev. Mod. Phys. **74**(1), 47 (2002)
2. Andersen, R., Chellapilla, K.: Finding dense subgraphs with size bounds. In: International workshop on algorithms and models for the web-graph, Springer, pp 25–37 (2009)
3. Asahiro, Y., Iwama, K., Tamaki, H., Tokuyama, T.: Greedily finding a dense subgraph. J. Algorithms **34**(2), 203–221 (2000)
4. Asahiro, Y., Hassin, R., Iwama, K.: Complexity of finding dense subgraphs. Discret. Appl. Math. **121**(1–3), 15–26 (2002)
5. Bahmani, B., Kumar, R., Vassilvitskii, S.: Densest subgraph in streaming and mapreduce. arXiv preprint arXiv:1201.6567 (2012)
6. Boldi, P., Vigna, S.: The webgraph framework i: Compression techniques. In: Proceedings of the 13th international conference on World Wide Web, ACM, pp 595–602 (2004)
7. Boldi, P., Rosa, M., Santini, M., Vigna, S.: Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In: Proceedings of the 20th international conference on World Wide Web, ACM, pp 587–596 (2011)

8. Boob, D., Gao, Y., Peng, R., Sawlani, S., Tsourakakis, C., Wang, D., Wang, J.: Flowless: Extracting densest subgraphs without flow computations. Proceedings of The Web Conference **2020**, 573–583 (2020)
9. Chang, L., Qiao, M.: Deconstruct densest subgraphs. Proceedings of The Web Conference **2020**, 2747–2753 (2020)
10. Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: International Workshop on Approximation Algorithms for Combinatorial Optimization, Springer, pp 84–95 (2000)
11. Chekuri, C., Quanrud, K., Torres, M.R.: Densest subgraph: Supermodularity, iterative peeling, and flow. In: SODA, SIAM, pp 1531–1555 (2022)
12. Chen, J., Saad, Y.: Dense subgraph extraction with application to community detection. IEEE Trans. Knowl. Data Eng. **24**(7), 1216–1230 (2010)
13. Ching, A., Edunov, S., Kabiljo, M., Logothetis, D., Muthukrishnan, S.: One trillion edges: Graph processing at facebook-scale. Proceedings of the VLDB Endowment **8**(12), 1804–1815 (2015)
14. Conte, A., De Matteis, T., De Sensi, D., Grossi, R., Marino, A., Versari, L.: D2k: scalable community detection in massive networks via small-diameter k-plexes. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp 1272–1281 (2018)
15. Danisch, M., Chan, T.H.H., Sozio, M.: Large scale density-friendly graph decomposition via convex programming. In: Proceedings of the 26th International Conference on World Wide Web, pp 233–242 (2017)
16. Dondi, R., Hosseinzadeh, M.M., Guzzi, P.H.: A novel algorithm for finding top-k weighted overlapping densest connected subgraphs in dual networks. Appl. Network Sci. **6**(1), 1–17 (2021)
17. Dondi, R., Hosseinzadeh, M.M., Mauri, G., Zoppis, I.: Top-k overlapping densest subgraphs: approximation algorithms and computational complexity. J. Comb. Optim. **41**(1), 80–104 (2021)
18. Dourisboure, Y., Geraci, F., Pellegrini, M.: Extraction and classification of dense communities in the web. In: Proceedings of the 16th international conference on World Wide Web, pp 461–470 (2007)
19. Fang, Y., Yu, K., Cheng, R., Lakshmanan, L.V., Lin, X.: Efficient algorithms for densest subgraph discovery. Proceedings of the VLDB Endowment 12(11) (2019)
20. Fang, Y., Yu, K., Cheng, R., Lakshmanan, L.V., Lin, X.: Efficient algorithms for densest subgraph discovery. arXiv preprint arXiv:1906.00341 (2019)
21. Fang, Y., Huang, X., Qin, L., Zhang, Y., Zhang, W., Cheng, R., Lin, X.: A survey of community search over big graphs. VLDB J. **29**(1), 353–392 (2020)
22. Fang, Y., Luo, W., Ma, C.: Densest subgraph discovery on large graphs: Applications, challenges, and techniques. Proceedings of the VLDB Endowment 15 (2022)
23. Frank, M., Wolfe, P., et al.: An algorithm for quadratic programming. Naval research logistics quarterly **3**(1–2), 95–110 (1956)
24. Fratkin, E., Naughton, B.T., Brutlag, D.L., Batzoglou, S.: Motifcut: regulatory motifs finding with maximum density subgraphs. Bioinformatics **22**(14), e150–e157 (2006)
25. Galbrun, E., Gionis, A., Tatti, N.: Top-k overlapping densest subgraphs. Data Min. Knowl. Disc. **30**(5), 1134–1165 (2016)
26. Gionis, A., Junqueira, F.P., Leroy, V., Serafini, M., Weber, I.: Piggybacking on social networks. In: VLDB 2013-39th International Conference on Very Large Databases, vol 6, pp 409–420 (2013)
27. Goldberg, A.V.: Finding a maximum density subgraph. University of California Berkeley (1984)
28. Han, X.: Traffic incident detection: a deep learning framework. In: 2019 20th IEEE International Conference on Mobile Data Management (MDM), IEEE, pp 379–380 (2019)
29. Han, X., Grubenmann, T., Cheng, R., Wong, S.C., Li, X., Sun, W.: Traffic incident detection: A trajectory-based approach. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), IEEE, pp 1866–1869 (2020)
30. Han, X., Cheng, R., Grubenmann, T., Maniu, S., Ma, C., Li, X.: Leveraging contextual graphs for stochastic weight completion in sparse road networks. In: Proceedings of the 2022 SIAM International Conference on Data Mining (SDM), SIAM, pp 64–72 (2022)
31. Han, X., Cheng, R., Ma, C., Grubenmann, T.: Deeptea: effective and efficient online time-dependent trajectory outlier detection. Proceedings of the VLDB Endowment **15**(7), 1493–1505 (2022)
32. Han, X., Dell'Aglio, D., Grubenmann, T., Cheng, R., Bernstein, A.: A framework for differentially-private knowledge graph embeddings. Journal of Web Semantics **72**(100), 696 (2022)
33. Hooi, B., Song, H.A., Beutel, A., Shah, N., Shin, K., Faloutsos, C.: Fraudar: Bounding graph fraud in the face of camouflage. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 895–904 (2016)
34. Jin, R., Xiang, Y., Ruan, N., Fuhry, D.: 3-hop: a high-compression indexing scheme for reachability query. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, pp 813–826 (2009)
35. Kannan, R., Vinay, V.: Analyzing the structure of large graphs. Forschungsinst. für Diskrete Mathematik (1999)
36. Khuller, S., Saha, B.: On finding dense subgraphs. In: International colloquium on automata, languages, and programming, Springer, pp 597–608 (2009)
37. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (2014)
38. Li, X., Cheng, R., Chang, K.C.C., Shan, C., Ma, C., Cao, H.: On analyzing graphs with motif-paths. Proceedings of the VLDB Endowment **14**(6), 1111–1123 (2021)
39. Ma, C., Cheng, R., Lakshmanan, L.V., Grubenmann, T., Fang, Y., Li, X.: Linc: a motif counting algorithm for uncertain graphs. Proceedings of the VLDB Endowment **13**(2), 155–168 (2019)
40. Ma, C., Fang, Y., Cheng, R., Lakshmanan, L.V., Zhang, W., Lin, X.: Efficient algorithms for densest subgraph discovery on large directed graphs. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp 1051–1066 (2020)
41. Ma, C., Fang, Y., Cheng, R., Lakshmanan, L.V., Zhang, W., Lin, X.: Efficient directed densest subgraph discovery. ACM SIGMOD Rec. **50**(1), 33–40 (2021)
42. Ma, C., Fang, Y., Cheng, R., Lakshmanan, L.V., Zhang, W., Lin, X.: On directed densest subgraph discovery. ACM Transactions on Database Systems (TODS) **46**(4), 1–45 (2021)
43. Ma, C., Fang, Y., Cheng, R., Lakshmanan, L.V., Han, X.: A convex-programming approach for efficient directed densest subgraph discovery. In: Proceedings of the 2022 International Conference on Management of Data, pp 845–859 (2022)
44. Mitzenmacher, M., Pachocki, J., Peng, R., Tsourakakis, C., Xu, S.C.: Scalable large near-clique detection in large-scale networks via sampling. In: KDD, pp 815–824 (2015)
45. Orlin, J.B.: Max flows in o (nm) time, or better. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing, pp 765–774 (2013)
46. Qin, L., Li, R.H., Chang, L., Zhang, C.: Locally densest subgraph discovery. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 965–974 (2015)
47. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI, https://networkrepository.com (2015)
48. Saha, B., Hoch, A., Khuller, S., Raschid, L., Zhang, X.N.: Dense subgraphs with restrictions and applications to gene annotation

graphs. In: Annual International Conference on Research in Computational Molecular Biology, Springer, pp 456–472 (2010)
49. Samusevich, R., Danisch, M., Sozio, M.: Local triangle-densest subgraphs. In: 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, pp 33–40 (2016)
50. Sawlani, S., Wang, J.: Near-optimal fully dynamic densest subgraph. In: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, pp 181–193 (2020)
51. Seidman, S.B.: Network structure and minimum degree. Social networks **5**(3), 269–287 (1983)
52. Stelzl, U., Worm, U., Lalowski, M., Haenig, C., Brembeck, F.H., Goehler, H., Stroedicke, M., Zenkner, M., Schoenherr, A., Koeppen, S., et al.: A human protein-protein interaction network: a resource for annotating the proteome. Cell **122**(6), 957–968 (2005)
53. Sun, B., Danisch, M., Chan, T.H., Sozio, M.: Kclist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. Proceedings of the VLDB Endowment (PVLDB) (2020)
54. Tatti, N., Gionis, A.: Density-friendly graph decomposition. In: Proceedings of the 24th International Conference on World Wide Web, pp 1089–1099 (2015)
55. Trung, T.B., Chang, L., Long, N.T., Yao, K., Binh, H.T.T.: Verification-free approaches to efficient locally densest subgraph discovery. In: 2023 IEEE 39th International Conference on Data Engineering (ICDE), IEEE, pp 1–13 (2023)
56. Tsourakakis, C.: The k-clique densest subgraph problem. In: Proceedings of the 24th international conference on world wide web, pp 1122–1132 (2015)
57. Tsourakakis, C., Chen, T.: Dense subgraph discovery: Theory and application. In: SDM Tutorial (2021)
58. Tsourakakis, C., Bonchi, F., Gionis, A., Gullo, F., Tsiarli, M.: Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 104–112 (2013)
59. Tsourakakis, C.E.: A novel approach to finding near-cliques: The triangle-densest subgraph problem. arXiv preprint arXiv:1405.1477 (2014)
60. Wang, J., Cheng, J.: Truss decomposition in massive networks. Proceedings of the VLDB Endowment 5(9) (2012)
61. Xu, X., Liu, H., Lv, X., Wang, Y., Li, D.: An efficient and exact algorithm for locally h-clique densest subgraph discovery. Proceedings of the ACM on Management of Data **2**(6), 1–26 (2024)
62. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. Knowl. Inf. Syst. **42**(1), 181–213 (2015)
63. Zhou, Y., Guo, Q., Fang, Y., Ma, C.: A counting-based approach for efficient k-clique densest subgraph discovery. Proceedings of the ACM on Management of Data **2**(3), 1–27 (2024)
64. Zhou, Y., Guo, Q., Yang, Y., Fang, Y., Ma, C., Lakshmanan, L.: x In-depth analysis of densest subgraph discovery in a unified framework. arXiv preprint arXiv:2406.04738 (2012)