Verification-Free Approaches to Efficient Locally Densest Subgraph Discovery

Tran Ba Trung
Hanoi University of Science and Technology
batrung97@gmail.com

Lijun Chang ✉
The University of Sydney
lijun.chang@sydney.edu.au

Nguyen Tien Long
Hanoi University of Science and Technology
long.nt180129@sis.hust.edu.vn

Kai Yao
The University of Sydney
kyao8420@uni.sydney.edu.au

Huynh Thi Thanh Binh
Hanoi University of Science and Technology
binhht@soict.hust.edu.vn

***Abstract*—Finding dense subgraphs from a large graph is a fundamental graph mining task with many applications. The notion of locally densest subgraph (LDS) is recently formulated to identify multiple dense subgraphs that cover different regions of a large graph. Informally, an LDS is a subgraph with the highest density in its local region. The state-of-the-art algorithm for computing top-k LDSes with the highest densities is LDS. It iteratively computes the densest subgraph and removes it from the graph, where all the computed densest subgraphs form the candidates of LDSes. Then, each candidate is verified through a costly maximum flow computation. Although advanced pruning techniques are proposed in LDS, the verification step is still time consuming especially for not-so-small k values. In this paper, we aim to improve the efficiency of finding top-k LDSes by designing verification-free approaches. Our algorithms are based on our observation that the set of maximal λ-compact subgraphs for all possible λ values form a hierarchical structure, and LDSes are simply leaves in the hierarchical structure. Thus, we propose a divide-and-conquer algorithm LDS-DC as well as an optimized algorithm LDS-Opt to efficiently identify top-k LDSes without constructing the entire hierarchical structure. Both of our algorithms have lower time complexities than LDS. Extensive empirical studies on real graphs show that our optimized algorithm LDS-Opt outperforms LDS for all k values, and the improvement is up-to several orders of magnitude.**

***Index Terms*—Locally Densest Subgraphs, Locally Dense Subgraphs, Maximal λ-compact Subgraphs**

I. Introduction

Finding dense subgraphs from a large graph is a fundamental graph mining task [1], [2], [3]. As real-world graphs are typically globally sparse, dense subgraphs usually indicate semantically important regions of a graph. Dense subgraph mining has been used in many applications, e.g., detecting communities in social networks [4], [5], identifying stories in social media [6], spotting spam links in web graphs [7], and finding regulatory motifs in biological graphs [8].

A widely adopted notion of graph density is the *average-degree density*. Specifically, the density of an undirected graph $g = (V(g), E(g))$, denoted $\rho(g)$, is measured by the ratio of its number of edges to its number of vertices which is equal to half of its average degree, i.e., $\rho(g) = \frac{|E(g)|}{|V(g)|}$. Given a large graph $G = (V, E)$, the *densest subgraph* of G is the subgraph g that maximizes the density among all of G's subgraphs. The densest subgraph of an undirected graph G with n vertices and m edges can be computed in $\mathcal{O}(nm \log \frac{n^2}{m})$ time through network flow techniques [9], [10].

Lijun Chang is the corresponding author.

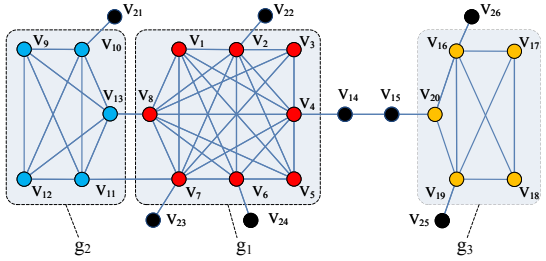


Fig. 1: An example graph

Reporting only a single densest subgraph however is often not sufficient for applications, as it covers only one region of a large input graph. In view of this, Qin et al. [11] recently formulated the notion of *locally densest subgraph (LDS)* aiming to find multiple dense subgraphs that cover different regions of a large graph. It is defined based on the notion of *(maximal) λ-compact subgraphs*. A graph is λ-compact if (1) it is connected and (2) removing any subset S of vertices and their associated edges from it will result in the removal of at least $\lambda|S|$ edges. A subgraph of G is a maximal λ-compact subgraph of G if it is the largest subgraph that is λ-compact. LDSes are those maximal λ-compact subgraphs for λ being equal to the density, i.e., a subgraph g of G is an LDS if it is a maximal $\rho(g)$-compact subgraph of G. The definition of LDS is parameter-free, and has the following elegant properties: (1) any subgraph of an LDS is not denser than itself, and (2) any proper supergraph of an LDS is not as compact as itself. Intuitively, an LDS is a subgraph with the highest density in its local region. Note that LDSes are vertex-induced subgraphs and are disjoint. For the graph in Figure 1, g_1 and g_3 are the only two LDSes, which are representative for the two dense regions of the graph.

2375-026X/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDE55515.2023.00008

The state-of-the-art algorithm for computing top-k LDSes with the highest densities is LDS proposed in [11], whose time complexity is $\mathcal{O}(m^2 n \log^2 n)$. The general idea is iteratively computing the densest subgraph and removing it from the graph; note that, as the graph keeps shrinking, the computed densest subgraphs are not necessarily the densest in the input graph G. It is proved in [11] that connected components of all the computed densest subgraphs are the candidates of LDSes. For each candidate g (i.e., connected component of the computed densest subgraphs), it verifies whether g is indeed an LDS: g is an LDS if it is a connected component of $F(\lambda)$ for $\lambda = \rho(g)$. This is based on the fact that the set of connected components of $F(\lambda)$ is the set of all maximal λ-compact subgraphs in G. Here, $F(\lambda)$ is defined as the subgraph (or vertex subset) S that maximizes $|E(S)| - \lambda|S|$ where tie is broken by taking the largest vertex subset and $E(S)$ denotes the set of edges of G with both end-points in S; $F(\lambda)$ can be computed via network flow [11]. Consider the graph in Figure 1, g_1 is the densest subgraph in the input graph and g_1 is also an LDS. After removing g_1 from the graph, g_2 becomes the densest subgraph; however, g_2 is not an LDS since $g_1 \cup g_2$ is maximal $\rho(g_2)$-compact. After further removing g_2 from the graph, g_3 becomes the densest subgraph and is an LDS. Pruning techniques are also proposed in [11] to improve the efficiency of LDS. Firstly, vertices that are guaranteed to be not in any LDS are removed from the graph; this speeds up the computation of densest subgraphs. Secondly, for verification, $F(\lambda)$ is computed on the $\lceil\lambda\rceil$-core of G instead of on the entire graph G, where l-core is the largest subgraph of G whose minimum degree is at least l. For large λ values, the $\lceil\lambda\rceil$-core is small and thus verification is relatively efficient. However, for small λ values, the $\lceil\lambda\rceil$-core remains large, which makes verification expensive. As a result, LDS becomes inefficient when it needs to verify candidates with relatively low density.

In this paper, we propose verification-free approaches for efficiently computing top-k LDSes. Instead of first generating densest subgraphs and then verifying them through computing $F(\lambda)$ on G, we compute maximal λ-compact subgraphs and obtain LDSes from them. This is based on our observation that the set of LDSes is a subset of all maximal λ-compact subgraphs for all possible λ values. To eliminate the need of expensive verification, we prove that (1) a maximal λ-compact subgraph is an LDS if and only if it does not contain any other maximal λ'-compact subgraph as a proper subgraph, and (2) the set of maximal λ-compact subgraphs for all possible λ values form a hierarchical structure; consequently, LDSes are simply leaves in the hierarchical structure. To efficiently construct the hierarchical structure, we prove that maximal λ-compact subgraphs are the same as connected components of locally dense subgraphs that are computed in [12].[1] However, directly invoking the algorithm of [12] to construct the hierarchical structure and then reporting the leaves as LDSes is inefficient for finding top-k LDSes. In view of this, we propose a divide-and-conquer algorithm LDS-DC by following the general idea of [12] while integrating top-k LDSes identification into the process such that we can stop as soon as k LDSes have been identified. We prove that the identified k LDSes have the highest densities, and that the time complexity of LDS-DC is $\mathcal{O}(n^2 m)$. Our empirical studies show that LDS-DC runs much faster than LDS for relatively large k values (e.g., $k \geq 20$), but may be outperformed by LDS when k is smaller. To resolve the inefficiency of LDS-DC for very small k values, we further propose an optimized approach LDS-Opt which consistently outperforms LDS-DC, especially for small k values.

Our main contributions are summarized as follows:

- We are the first to demonstrate the relationship between LDSes and locally dense subgraphs.
- We propose a verification-free algorithm LDS-DC that has a lower time complexity than the state of the art for finding top-k LDSes.
- We further propose an optimized algorithm LDS-Opt which is suitable for finding top-k LDSes for all k values.
- We conduct extensive experiments on real graphs to demonstrate the efficiency of our verification-free algorithms. The results show that LDS-DC outperforms the state-of-the-art algorithm LDS for $k \geq 20$, while LDS-Opt outperforms LDS for all k values.

The remainder of the paper is organized as follows. Section II defines the problem and presents preliminaries. We briefly review the state-of-the-art algorithm LDS and discuss its inefficiency in Section III. We formally characterize LDSes from maximal λ-compact subgraphs in Section IV, and propose two verification-free algorithms LDS-DC and LDS-Opt in Section V. The results of our empirical studies are presented in Section VI, and related works are discussed in Section VII. Finally, Section VIII concludes the paper.

II. Preliminaries

In this paper, we consider an unweighted and undirected graph $G = (V, E)$ with $n = |V(G)|$ vertices and $m = |E(G)|$ undirected edges. For each vertex $v \in V$, we use $N_G(v) = \{u \in V \mid (v, u) \in E\}$ to denote the set of neighbors of v in G. The degree of v in G is denoted $d_G(v) = |N_G(v)|$. Given a vertex subset $X \subseteq V$, we use $E_G(X)$ to denote the set of edges of G whose both end-points are in X, i.e., $E_G(X) = \{(u, v) \in E \mid u, v \in X\}$. The subgraph of G induced by X is denoted by $G[X]$, i.e., $G[X] = (X, E_G(X))$. When the context is clear, we omit the subscript G from the notations.

Given an arbitrary graph g, we use $V(g)$ and $E(g)$, respectively, to denote its vertex set and its edge set. The *density* of g, denoted $\rho(g)$, is defined as half of its average degree, i.e.,

$$\rho(g) = \frac{|E(g)|}{|V(g)|}$$

Definition II.1 (λ-compact [11])**.** *Given a graph g and a positive value λ, g is λ-compact if (1) it is connected, and*

[1] Note that, to avoid confusion of the concept of locally ***densest*** subgraph defined in [11] and the concept of locally ***dense*** subgraph defined in [12], we always use the abbreviation LDS to refer to locally densest subgraph while do not abbreviate locally dense subgraph in this paper.

(2) removing any subset S of vertices and their associated edges from it will result in the removal of at least $\lambda|S|$ edges.

It is easy to see that any λ-compact graph is also λ'-compact for any $\lambda' < \lambda$. A subgraph g of G is a *maximal λ-compact subgraph* of G if every proper supergraph of g in G is not λ-compact. The concept of locally densest subgraph is defined based on maximal λ-compact subgraph, as follows.

Definition II.2 (Locally Densest Subgraph (LDS) [11])**.** *A subgraph g of G is a locally densest subgraph if g is a maximal $\rho(g)$-compact subgraph in G.*

LDSes of a graph satisfy the following properties [11]:

- The density of any subgraph of an LDS g is at most $\rho(g)$, i.e., any subgraph of an LDS is not denser than itself.
- Any proper supergraph of an LDS g is not $\rho(g)$-compact, i.e., any proper supergraph of an LDS is not as compact as itself.
- LDSes are disjoint, i.e., $V(g) \cap V(g') = \emptyset$ for any two distinct LDSes g and g'.

It is easy to see from the definition that both a maximal λ-compact subgraph of G and an LDS of G are vertex-induced subgraphs of G. For presentation simplicity, we also use a vertex subset to refer to the corresponding vertex-induced subgraph.

Example II.1. *Consider the graph in Figure 1. g_1 is the subgraph induced by vertices $\{v_1, v_2, \ldots, v_8\}$ which consists of 8 vertices and 24 undirected edges. Thus, the density of g_1 is $\rho(g_1) = \frac{24}{8} = 3$. g_1 is an LDS since itself is a maximal 3-compact subgraph.*

g_2 is the subgraph induced by vertices $\{v_9, v_{10}, \ldots, v_{13}\}$ which consists of 5 vertices and 10 undirected edges. Thus, the density of g_2 is $\rho(g_2) = 2$. Although g_2 does not have any subgraph that is denser than itself, g_2 is not an LDS. This is because g_2 is not a maximal 2-compact subgraph; instead, its proper supergraph $g_1 \cup g_2$ is a maximal 2-compact subgraph.

g_3 is the subgraph induced by vertices $\{v_{16}, v_{17}, \ldots, v_{20}\}$ consisting of 5 vertices and 8 undirected edges. Its density is $\rho(g_3) = \frac{8}{5}$. g_3 is an LDS, since itself is a maximal $\frac{8}{5}$-compact subgraph.

Problem Statement. Given an unweighted and undirected graph G and an integer k, we study the problem of finding the k LDSes of G that have the highest densities.

Frequently used notations are summarized in Table I.

A. Locally Dense Subgraphs

We will show in Section V-A that LDSes are closely related to locally dense subgraphs defined in [12]. Thus, in this subsection, we briefly review the definition and properties of locally dense subgraph. Note that, to avoid confusion of the concept of locally ***densest*** subgraph defined in [11] and the concept of locally ***dense*** subgraph defined in [12], we always use the abbreviation LDS to refer to locally densest subgraph while do not abbreviate locally dense subgraph in this paper.

TABLE I: Frequently used notations

Notation	Meaning
$G = (V, E)$	A graph G with vertex set V and edge set E
X, Y, Z, S, U, W	Vertex subsets of V
$E(X)$	The set of edges of G whose both end-points are in X, i.e., $E(X) = \{(u, v) \in E \mid u, v \in X\}$
$G[X]$	The subgraph of G induced by vertex subset X, i.e., $G[X] = (X, E(X))$
$\rho(g)$	The density of graph $g = (V(g), E(g))$, i.e., $\rho(g) = \frac{\lvert E(g)\rvert}{\lvert V(g)\rvert}$
$E(X, Y)$	The cross edges between disjoint vertex set X and Y, i.e., $E(X, Y) = \{(u, v) \in E \mid u \in X, v \in Y\}$
$\rho(X, Y)$	The outer density of X with respect to Y, i.e., $\rho(X, Y) = \frac{\lvert E(X)\rvert + \lvert E(X,Y)\rvert}{\lvert X\rvert}$ for disjoint vertex sets X and Y; $\rho(X, Y) = \rho(X \backslash Y, Y)$ otherwise
$B_0, B_1, \ldots, B_r$	Locally dense subgraphs
$F(\lambda)$	The vertex subset S that maximizes $\lvert E(S)\rvert - \lambda\lvert S\rvert$, i.e., $F(\lambda) = \operatorname{argmax}_{S \subseteq V} \lvert E(S)\rvert - \lambda\lvert S\rvert$

Firstly, we need the definition of outer density. Given a graph $G = (V, E)$ and two disjoint vertex subsets X and Y of V, the *outer density* of X with respect to Y, denoted $\rho(X, Y)$, is defined as

$$\rho(X, Y) = \frac{|E(X)| + |E(X, Y)|}{|X|}$$

where $E(X, Y) = \{(u, v) \in E \mid u \in X, v \in Y\}$ denotes the set of *cross edges* between X and Y. Note that, $\rho(X, \emptyset) = \rho(X)$ and $\rho(\emptyset, X) = 0$. In the case that X and Y are overlapping, the outer density will be expanded as:

$$\rho(X, Y) = \rho(X \setminus Y, Y)$$

Locally dense subgraph is defined based on the notion of outer density as follows.

Definition II.3 (Locally Dense Subgraph [12])**.** *Given a graph $G = (V, E)$, a vertex subset $W \subseteq V$ is a locally dense subgraph if there are no $X \subseteq W$ and $Y \subseteq V \setminus W$ such that $\rho(X, W \setminus X) \leq \rho(Y, W)$.*

It is shown in [12] that locally dense subgraphs are nested, i.e., for any two locally dense subgraphs U and W, either $U \subseteq W$ or $W \subseteq U$. Thus, the set of locally dense subgraphs can be arranged into a sequence

$$\emptyset = B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_r = V$$

where $r \leq n$. Moreover, it satisfies the property that

$$\rho(B_i, B_{i-1}) > \rho(B_{i+1}, B_i), \text{ for } 0 < i < r.$$

Example II.2. *Consider the graph in Figure 1, there are five non-empty locally dense subgraphs. $B_1 = \{v_1, v_2, \ldots, v_8\}$, $B_2 = B_1 \cup \{v_9, v_{10}, \ldots, v_{13}\}$, $B_3 = B_2 \cup \{v_{16}, v_{17}, \ldots, v_{20}\}$, $B_4 = B_3 \cup \{v_{14}, v_{15}\}$, and $B_5 = V$. $\rho(B_1, \emptyset) = \rho(B_1) = 3$, $\rho(B_2, B_1) = \frac{12}{5}$, $\rho(B_3, B_2) = \frac{8}{5}$, $\rho(B_4, B_3) = \frac{3}{2}$, and $\rho(B_5, B_4) = 1$.*

III. Inefficiency of the State-of-the-art Approach

The state-of-the-art approach for computing top-k LDSes is LDS proposed in [11]. The general idea of LDS is based on the following two lemmas.

Lemma III.1. *[11] Any densest subgraph component of G is an LDS of G, where a densest subgraph component is a connected component of the densest subgraph.*

Lemma III.2. *[11] Let g be an LDS of G. Any LDS g' ($g' \neq g$) in G is still an LDS in G' where G' is the residual graph of G after removing g.*

Thus, LDS iteratively computes the densest subgraph of G', which is initialized as G, and removes the densest subgraph from G', until top-k LDSes have been obtained or G' becomes empty. For each connected component g of the computed densest subgraphs, it verifies whether g is an LDS based on the following lemma, which says that g is an LDS if it is a connected component of $F(\lambda)$ for $\lambda = \rho(g)$. Here, $F(\lambda)$ is defined as the subgraph (or vertex subset) S that maximizes $|E(S)| - \lambda|S|$, where tie is broken by taking the largest vertex subset, i.e., $F(\lambda) = \text{argmax}_{S \subseteq V} |E(S)| - \lambda|S|$.

Lemma III.3. *[11] If G contains maximal λ-compact subgraphs, then the set of connected components of $F(\lambda) = \text{argmax}_{S \subseteq V} |E(S)| - \lambda|S|$ is the set of all maximal λ-compact subgraphs in G.*

Algorithm 1: LDS [11]

Input: A graph $G = (V, E)$ and an integer k
Output: Top-k LDSes $\mathcal{R}$

1 $G' \leftarrow \mathsf{prune}(G)$;
2 Initialize a priority queue $\mathcal{H} \leftarrow \emptyset$;
3 **for each** *connected component g of G'* **do**
4 $\quad \mathcal{H}.\mathsf{push}(g, \bar{\rho}^*(g), \text{false})$;
5 $\mathcal{R} \leftarrow \emptyset$;
6 **while** $\mathcal{H} \neq \emptyset$ **and** $|\mathcal{R}| < k$ **do**
7 $\quad (g, ub, \text{densest}) \leftarrow \mathcal{H}.\mathsf{pop}()$;
8 $\quad$ **if** $\text{densest} = \text{true}$ **then**
9 $\quad\quad$ **if** $\mathsf{verify}(g, G)$ **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \{g\}$;
10 $\quad$ **else**
11 $\quad\quad g' \leftarrow$ any connected component of the densest subgraph of g;
12 $\quad\quad \mathcal{H}.\mathsf{push}(g', \rho(g'), \text{true})$;
13 $\quad\quad G' \leftarrow$ the residual graph of g after removing g';
14 $\quad\quad G' \leftarrow \mathsf{prune}(G')$;
15 $\quad\quad$ **for each** *connected component g of G'* **do**
16 $\quad\quad\quad \mathcal{H}.\mathsf{push}(g, \bar{\rho}^*(g), \text{false})$;
17 **return** $\mathcal{R}$;

The pseudocode of LDS is shown in Algorithm 1. For time efficiency consideration, it processes the graph in a component-by-component manner where the connected components are stored in a priority queue $\mathcal{H}$ (Lines 3–4 and 15–16), and conducts vertex pruning by removing from the graph all vertices that are guaranteed to be not in any LDS (Lines 1 and 14). Specifically, $\mathcal{H}$ stores both connected components and densest subgraph components, which are distinguished via the third element (i.e., densest) of each entry; that is, $\text{densest} = \text{true}$ means that this entry stores a densest subgraph component (Line 8). The priority of a component g in $\mathcal{H}$ is an upper bound of the densest subgraph in g, denoted $\bar{\rho}^*(g)$ (Lines 4 and 16); if g itself is a densest subgraph component, then the upper bound is defined as $\rho(g)$.

LDS works as follows. It initially puts all the connected components of the pruned graph G' into the priority queue $\mathcal{H}$ (Lines 2–4). Then, it iteratively computes the next LDS that has the highest density (Lines 6–16). To do so, it pops from $\mathcal{H}$ the component g that has the highest upper bound (Line 7). If g is a densest subgraph component (Line 8), it verifies whether g is an LDS by calling $\mathsf{verify}(g, G)$ (Line 9); note that, if g is an LDS, then due to the nature of $\mathcal{H}$, g is guaranteed to be the next LDS that has the highest density. Otherwise, g is not a densest subgraph component (Line 10), it computes the densest subgraph in g, and let g' be any connected component of this densest subgraph (Line 11). Then, it inserts into $\mathcal{H}$ the densest subgraph component g' (Line 12), as well as the connected components of the residual graph of g after removing g' and conducting vertex pruning (Lines 13–16).

Inefficiency of LDS. Despite that LDS incorporates several advanced pruning techniques, it is still inefficient in the following two aspects. Firstly, to compute the densest subgraph of g, it needs to compute $F(\lambda)$ on g for multiple λ values by binary searching on λ [10]. Secondly, to verify whether g is truely an LDS, it needs to compute $F(\lambda)$ on G for $\lambda = \rho(g)$. The inefficiency of LDS becomes severer when there are many false-positives (i.e., we need to conduct the above computation for many more subgraphs g), and/or when λ is small which is the case when k becomes large.

We remark that verifying whether g is an LDS becomes extremely time consuming when $\rho(g)$ is small, as it needs to compute $F(\lambda)$ on G for a small value $\lambda = \rho(g)$. It is proved in [11] that when computing $F(\lambda)$ on G, we only need to work on the $\lceil\lambda\rceil$-core of G, where the l-core of G is the largest subgraph of G whose minimum degree is at least l; this is intuitive, since the minimum degree of a λ-compact subgraph must be at least λ. For large λ values, the $\lceil\lambda\rceil$-core of G is small and thus the computation of $F(\lambda)$ is relatively efficient. However, for small λ values, the $\lceil\lambda\rceil$-core of G is large, which makes the computation of $F(\lambda)$ time consuming. This motivates us to design verification-free approaches for computing LDSes in Section V.

IV. Characterizing LDSes from Maximal λ-Compact Subgraphs

According to Definition II.2, an LDS of G must be a maximal λ-compact subgraph of G for some λ. Thus, the set of all LDSes of G is a subset of all maximal λ-compact subgraphs of G for all possible λ values. In this section, we characterize LDSes from maximal λ-compact subgraphs, which will enable us to design efficient algorithms to explore all LDSes in the next section. In the following, we first in Section IV-A prove

the condition for a maximal λ-compact subgraph to be an LDS, then in Section IV-B prove the hierarchical structure of all maximal λ-compact subgraphs for all possible λ-values, and finally in Section IV-C show that LDSes are simply leaves of the hierarchical structure.

A. Condition for a Maximal λ-Compact Subgraph to be an LDS

To prove the condition for a maximal λ-compact subgraph to be an LDS, we first define the notion of *compactness* of a graph and build the connection between compactness and the density.

Definition IV.1 (Compactness)**.** *The compactness of a connected graph g, denoted $\eta(g)$, is the largest λ such that g is λ-compact. The compactness of a disconnected graph is defined to be* 0*.*

Lemma IV.1. *For any connected graph g, $\eta(g) \leq \rho(g)$.*

Proof. According to the definition of compactness, g is $\eta(g)$-compact. Thus, $|E(g)| \geq \eta(g) \times |V(g)|$ since the set of vertices to be removed can be $V(g)$, and consequently $\rho(g) = \frac{|E(g)|}{|V(g)|} \geq \eta(g)$. □

Lemma IV.2. *For any connected graph g, $\eta(g) < \rho(g)$ if and only if there is a subgraph g' of g such that $\rho(g') > \rho(g)$.*

Proof. ($\Longrightarrow$) If $\eta(g) < \rho(g)$, then there is a vertex subset S of g whose removal from g will result in the removal of less than $\rho(g) \times |S|$ edges. Let g' be the resulting graph of g obtained by removing vertices of S and their associated edges. Then $|E(g')| > |E(g)| - \rho(g) \times |S| = \rho(g) \times (|V(g)| - |S|) = \rho(g) \times |V(g')|$. Thus, $\rho(g') = \frac{|E(g')|}{|V(g')|} > \rho(g)$.

($\Longleftarrow$) If there is a subgraph g' of g such that $\rho(g') > \rho(g)$, then we also have $\rho(g'') > \rho(g)$ where g'' denotes the subgraph of g induced by $V(g')$. Let S be $V(g) \setminus V(g'')$. Then, removing S from g will result in the removal of $|E(g)| - |E(g'')|$ edges. Note that, $|E(g)| = \rho(g) \times |V(g)|$ and $|E(g'')| = \rho(g'') \times |V(g'')| > \rho(g) \times |V(g'')|$. Thus, $|E(g)| - |E(g'')| < \rho(g) \times (|V(g)| - |V(g'')|) = \rho(g) \times |S|$. Consequently, g cannot be $\rho(g)$-compact, and $\eta(g) < \rho(g)$. □

Corollary IV.1. *For any connected graph g, $\eta(g) = \rho(g)$ if and only if the density of every subgraph of g is at most $\rho(g)$.*

Proof. This directly follows from Lemma IV.2. □

Now, we are ready to prove the condition for a maximal λ-compact subgraph of G to be an LDS of G.

Lemma IV.3. *For any maximal λ-compact subgraph g of G, g is an LDS if and only if g contains no λ'-compact subgraph for $\lambda' > \eta(g)$.*

Proof. First, we prove by contradiction that if g is an LDS, then g contains no λ'-compact subgraph for $\lambda' > \eta(g)$. Suppose there is such a subgraph g' of g that is λ'-compact for $\lambda' > \eta(g)$, i.e., $\eta(g') > \eta(g)$. As g is an LDS, from Definition II.2 and Lemma IV.1, we have $\eta(g) = \rho(g)$; thus $\eta(g') > \rho(g)$. From Lemma IV.1, we know that $\rho(g') \geq \eta(g')$. Thus $\rho(g') > \rho(g)$, which implies that $\eta(g) < \rho(g)$ following Lemma IV.2; contradiction. Thus, g contains no λ'-compact subgraph for $\lambda' > \eta(g)$.

Second, we prove that if g is not an LDS, then g must have a λ'-compact subgraph for $\lambda' > \eta(g)$. Since g is not an LDS, we have $\eta(g) < \rho(g)$ according to Definition II.2 and Lemma IV.1. Then, from Lemma IV.2 we know that g must have a subgraph g' such that $\rho(g') > \rho(g)$. Let g'' be the densest subgraph of g; note that, $\rho(g'') \geq \rho(g') > \rho(g)$. From Corollary IV.1, we have $\eta(g'') = \rho(g'')$. Consequently, $\eta(g'') > \rho(g) > \eta(g)$.

Now, the lemma follows. □

B. Hierarchical Structure of Maximal λ-Compact Subgraphs

In this subsection, we prove that the set of maximal λ-compact subgraphs of G for all possible λ-values form a hierarchical structure.

We first prove in the lemma below that the union of two overlapping or adjacent λ-compact subgraphs is still λ-compact. For presentation simplicity, we only consider vertex-induced subgraphs since all maximal λ-compact subgraphs are vertex-induced subgraphs.

Lemma IV.4. *Given a graph G and two λ-compact subgraphs X and Y of G, if $X \cap Y \neq \emptyset$ or $E(X, Y) \neq \emptyset$, then $X \cup Y$ is also λ-compact.*

Proof. Firstly, we consider the case of $X \cap Y \neq \emptyset$. Then, $G[X \cup Y]$ is connected. Let's consider an arbitrary subset $S \subseteq X \cup Y$. Let $S_X = S \cap (X \setminus Y)$ and $S_Y = S \cap Y$; then, $S_X \cap S_Y = \emptyset$ and $S_X \cup S_Y = S$. Let $\bar{S}$ be $(X \cup Y) \setminus S$. If we remove S from $X \cup Y$, then the set of edges that will be removed is $E(S) \cup E(S, \bar{S})$. We have

$$
\begin{aligned}
&|E(S) \cup E(S, \bar{S})| \\
=&|E(S)| + |E(S, \bar{S})| \\
=&|E(S_X \cup S_Y)| + |E(S_X \cup S_Y, \bar{S})| \\
=&|E(S_X) \cup E(S_X, S_Y) \cup E(S_Y)| + |E(S_X, \bar{S}) \cup E(S_Y, \bar{S})| \\
=&\big(|E(S_X)| + |E(S_X, S_Y)| + |E(S_X, \bar{S})|\big) + \\
&\big(|E(S_Y)| + |E(S_Y, \bar{S})|\big) \\
\geq&\big(|E(S_X)| + |E(S_X, S_Y \cap X)| + |E(S_X, \bar{S} \cap X)|\big) + \\
&\big(|E(S_Y)| + |E(S_Y, \bar{S} \cap Y)|\big) \\
=&(|E(S_X)| + |E(S_X, X \setminus S_X)|) + \\
&(|E(S_Y)| + |E(S_Y, Y \setminus S_Y)|)
\end{aligned}
$$

where the last equality follows from the fact that $S_X \cup S_Y \cup \bar{S} = X \cup Y$. As both X and Y are λ-compact, it follows that $|E(S_X)| + |E(S_X, X \setminus S_X)| \geq \lambda \times |S_X|$ and $|E(S_Y)| + |E(S_Y, Y \setminus S_Y)| \geq \lambda \times |S_Y|$. Thus, $|E(S) \cup E(S, \bar{S})| \geq \lambda \times (|S_X| + |S_Y|) = \lambda \times |S|$. Consequently, $X \cup Y$ is λ-compact.

Secondly, we consider the case of $X \cap Y = \emptyset$ and $E(X, Y) \neq \emptyset$. Then, $G[X \cup Y]$ is connected. By using a similar argument as above, $|E(S) \cup E(S, \bar{S})| \geq \lambda |S|$ holds for any subset $S \subseteq X \cup Y$, and thus $X \cup Y$ is λ-compact. □

Now, we prove the hierarchical structure of maximal λ-compact subgraphs for all possible λ values in the following lemma.

Lemma IV.5. *Given a graph G, any two distinct maximal λ-compact subgraphs X and Y are disjoint (i.e., $X \cap Y = \emptyset$ and $E(X,Y) = \emptyset$). For any maximal λ-compact subgraph X and any maximal λ'-compact subgraph X' with $\lambda > \lambda'$, either $X \subseteq X'$, or X and X' are disjoint (i.e., $X \cap X' = \emptyset$ and $E(X,X') = \emptyset$).*

Proof. The disjointness of X and Y directly follows from Lemma IV.4. We prove the relationship between X and X' by contradiction. Suppose that $X \not\subseteq X'$, and either $X \cap X' \neq \emptyset$ or $E(X,X') \neq \emptyset$. Then, from Lemma IV.4, we know that $X \cup X'$, which is not the same as X', is also λ'-compact since X is λ'-compact for $\lambda' < \lambda$; this contradicts that X' is a maximal λ'-compact subgraph. □

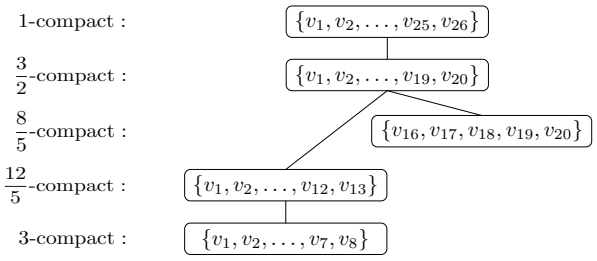


Fig. 2: An example of hierarchical structure of all maximal λ-compact subgraphs

For example, Figure 2 illustrates the hierarchical structure of all maximal λ-compact subgraphs for all possible λ values for the graph in Figure 1. Note that, $\{v_1, v_2, \ldots, v_{13}\}$ is both $\frac{12}{5}$-compact and $\frac{8}{5}$-compact; we only show it as $\frac{12}{5}$-compact in the figure, since $\frac{12}{5}$ is its compactness.

C. Put It Together

Now, we are ready to characterize LDSes in the hierarchical structure of all maximal λ-compact subgraphs for all possible λ values.

Theorem IV.6. *For any maximal λ-compact subgraph g of G, g is an LDS if and only if g contains no maximal λ'-compact subgraph as a proper subgraph for any λ'.*

Proof. From Lemma IV.3, we know that g is an LDS if and only if it contains no λ'-compact subgraph for $\lambda' > \eta(g)$. As g is a maximal λ-compact subgraph, from Lemma IV.5 we know that for $\lambda' > \eta(g)$, g contains a λ'-compact subgraph if and only if g contains a *maximal* λ'-compact subgraph. Thus, g is an LDS if and only if it contains no *maximal* λ'-compact subgraph for $\lambda' > \eta(g)$; note that, any maximal λ'-compact subgraph for $\lambda' > \eta(g)$ must be a proper subgraph of g since g is not λ'-compact. On the other hand, for $\lambda' \leq \eta(g)$, it is trivial that any proper subgraph of g is not a maximal λ'-compact subgraph. Thus, the theorem holds. □

It is worth pointing out that Theorem IV.6 is different from Lemma IV.3. Specifically, Theorem IV.6 specifies that g contains no ***maximal*** λ'-compact subgraph, while Lemma IV.3 does not enforce maximality. It is from Theorem IV.6 that we can design verification-free approaches for computing LDSes.

From Theorem IV.6, we can conclude that the leaves of the hierarchical structure of all maximal λ-compact subgraphs for all possible λ values are exactly the LDSes. Thus, from Figure 2, we can see that $\{v_1, v_2, \ldots, v_8\}$ and $\{v_{16}, v_{17}, \ldots, v_{20}\}$ are the two LDSes.

V. Verification-Free Approaches

In this section, we design verification-free approaches for efficiently computing LDSes by following Theorem IV.6. We first propose a divide-and-conquer approach in Section V-A, and then design optimization techniques in Section V-B.

A. A Divide-and-Conquer Approach

We have proved in Section IV that LDSes are simply leaves of the hierarchical structure of all maximal λ-compact subgraphs for all possible λ values. Thus, the remaining problem for computing LDSes is how to efficiently construct the hierarchical structure. We prove in the following lemma that maximal λ-compact subgraphs of G are the same as connected components of locally dense subgraphs of G; thus, we can use the locally dense subgraph computation techniques of [12] to construct the hierarchical structure. Recall that (1) $F(\lambda) = \operatorname{argmax}_{S \subseteq V} |E(S)| - \lambda |S|$ where tie is broken by taking the largest vertex subset, and (2) the set of locally dense subgraphs can be arranged into a sequence $\emptyset = B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_r = V$.

Lemma V.1. *Maximal λ-compact subgraphs of G are connected components of locally dense subgraphs of G.*

Proof. We build the connection between maximal λ-compact subgraphs and locally dense subgraphs through $F(\lambda)$. Firstly, it is proved in [11] that the set of connected components of $F(\lambda)$ is exactly the set of all maximal λ-compact subgraphs of G. Secondly, it is proved in [12] that $F(\lambda)$ for any λ satisfying $\rho(B_{i+1}, B_i) < \lambda \leq \rho(B_i, B_{i-1})$ is the locally dense subgraph B_i; note that, this also means that $F(\lambda)$ remains the same for all λ values in the range $(\rho(B_{i+1}, B_i), \rho(B_i, B_{i-1})]$. Thirdly, $F(\lambda) = \emptyset$ for any $\lambda > \rho(B_1, \emptyset) = \rho(B_1)$ as B_1 is the densest subgraph of G [12]. Thus, the lemma follows. □

Following Lemma V.1, we could first invoke the algorithm of [12] to construct the hierarchical structure of all maximal λ-compact subgraphs for all possible λ values, and then report the leaves of the hierarchical structure as LDSes. However, this would be inefficient for reporting top-k LDSes as it first constructs the entire hierarchical structure. Moreover, the implementation of [12] involves computing maximum flow over a graph with floating value capacities which may lead to inaccurate results [13]. In view of this, we propose a divide-and-conquer algorithm in this subsection.

Firstly, we prove the following lemma, which is central to our algorithm design.

Lemma V.2. *Let $\emptyset = B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_r = V$ be the sequence of locally dense subgraphs. Consider i and j such that $0 \leq i < j \leq r$, and let $B_l = F(\lambda)$ for $\lambda = \rho(B_j, B_i)$.*

- *If $i + 1 < j$, then $i < l < j$.*
- *If $i + 1 = j$, then $l = j$.*

Proof. Let's first consider the case that $i + 1 < j$. Recall that locally dense subgraphs satisfy the property that $\rho(B_i, B_{i-1}) > \rho(B_{i+1}, B_i)$. Let's denote $\rho(B_i, B_{i-1})$ by $\frac{a_i}{b_i}$. Then, we have $\frac{a_i}{b_i} > \frac{a_{i+1}}{b_{i+1}} > \cdots > \frac{a_{j-1}}{b_{j-1}} > \frac{a_j}{b_j}$. As $\rho(B_j, B_i) = \sum_{x=i+1}^{j} \frac{a_x}{b_x}$, a simple calculation shows that $\rho(B_j, B_i) > \frac{a_j}{b_j} = \rho(B_j, B_{j-1})$. Similarly, we also have $\rho(B_j, B_i) < \frac{a_{i+1}}{b_{i+1}} = \rho(B_{i+1}, B_i)$. Thus, for $B_l = F(\lambda)$ with $\lambda = \rho(B_j, B_i)$, we have $i + 1 \leq l \leq j - 1$.

Now, let's consider the case that $i + 1 = j$. We have $\lambda = \rho(B_j, B_i) = \rho(B_{i+1}, B_i) > \rho(B_{i+2}, B_{i+1}) = \rho(B_{j+1}, B_j)$ and thus $l = i + 1 = j$. □

Note that, Lemma V.2 is similar to Proposition 10 of [12] but different in setting the value for λ. Specifically, Proposition 10 of [12] considers $F(\lambda)$ for $\lambda = \rho(B_j, B_i) + \frac{1}{n^2}$ where there is an additional term of $\frac{1}{n^2}$, and thus leads to a different statement for the second bullet point: if $i + 1 = j$, then $l = i$. The advantage of our Lemma V.2 will become clear when we discuss the computation of $F(\lambda)$ shortly.

Corollary V.1. *Given any two locally dense subgraphs X and Y such that $X \subsetneq Y$, let $Z = F(\lambda)$ where $\lambda = \rho(Y, X)$.*

- *If $Z = Y$, then there is no locally dense subgraph W such that $X \subsetneq W \subsetneq Y$.*
- *If $Z \neq Y$, then $X \subsetneq Z \subsetneq Y$ and Z is a locally dense subgraph.*

Proof. This directly follows from Lemma V.2. □

Following Corollary V.1, we can compute all locally dense subgraphs in G in a divide-and-conquer manner. As a special case, we know that $B_0 = \emptyset$ and $B_r = V$. Thus, we can initially let $X = \emptyset$ and $Y = V$, and compute $F(\lambda)$ for $\lambda = \rho(Y, X)$. If $F(\lambda) \neq Y$, then we divide the problem into two sub-problems, one for X and $F(\lambda)$ which will compute all locally dense subgraphs in-between X and $F(\lambda)$, and another for $F(\lambda)$ and Y which will compute all locally dense subgraphs in-between $F(\lambda)$ and Y. Otherwise, we reach a base case that there is no more locally dense subgraph in-between X and Y.

However, to efficiently obtain the top-k LDSes, we cannot afford to first construct the entire hierarchical structure of maximal λ-compact subgraphs. Thus, we propose to integrate top-k LDS identification into the process and stop as soon as k LDSes have been identified. Given the sequence of locally dense subgraphs $\emptyset = B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_r = V$, we know from Theorem IV.6 and Lemma V.1 that LDSes are those connected components of B_i that do not include any vertex of B_{i-1} for $0 < i \leq r$. However, obtaining connected components of $G[B_i]$ may be time consuming since B_i could be large. We prove in the following lemma that we can check connected components of $G[B_i \backslash B_{i-1}]$ instead.

Lemma V.3. *Given the sequence of locally dense subgraphs $\emptyset = B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_r = V$, LDSes of G are those connected components of $G[B_i \backslash B_{i-1}]$ that have no edge to B_{i-1}, for $0 < i \leq r$.*

Proof. This can be easily seen from the fact that a connected component of $G[B_i]$ does not include any vertex of B_{i-1} if and only if it is a connected component of $G[B_i \backslash B_{i-1}]$ that have no edge to B_{i-1}, since B_{i-1} is a subset of B_i. □

Algorithm 2: LDS-DC(X, Y, k)

Input: Two locally dense subgraphs X and Y such that $X \subsetneq Y$
Output: Top-k LDSes W such that $W \subseteq Y \backslash X$

```
1  R ← ∅;
2  λ ← ρ(Y, X);
3  F(λ) ← LD(λ, X, Y);
4  if F(λ) = Y then
5      for each connected component W of G[Y\X] do
6          if E(W, X) = ∅ then R ← R ∪ {W};
7  else
8      if |R| < k then R ← R ∪ LDS-DC(X, F(λ), k − |R|);
9      if |R| < k then R ← R ∪ LDS-DC(F(λ), Y, k − |R|);
10 return R;
```

Based on Lemma V.3 and the above discussions, the pseudocode of our divide-and-conquer algorithm for computing top-k LDSes is shown in Algorithm 2, where invoking LDS-DC$(\emptyset, V, k)$ returns the top-k LDSes in G. It works as follows. Given two locally dense subgraphs X and Y such that $X \subsetneq Y$, it first computes $F(\lambda)$ for $\lambda = \rho(Y, X)$ by invoking LD which will be introduced shortly. If $F(\lambda) = Y$, then we reach the first case of Corollary V.1 (i.e., $X = B_{i-1}$ and $Y = B_i$ for some i), and thus we know that connected components of $G[Y \backslash X]$ that do not have any edge to X are LDSes (Lines 5–6). Note that, checking whether $E(W, X)$ is empty or not for a connected component W of $G[Y \backslash X]$ can be conducted for free when obtaining the connected components of $G[Y \backslash X]$; thus, we call our approach as *verification-free*. If $F(\lambda) \neq Y$, then we reach the second case of Corollary V.1, and thus we split the problem into two sub-problems and recursively solve them (Lines 8–9). The correctness of Algorithm 2 directly follows from the above discussions, and the following lemma which ensures that the LDSes are enumerated in non-increasing density order.

Lemma V.4. *Given the sequence of locally dense subgraphs $\emptyset = B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_r = V$,*

- *all LDSes in $B_i \backslash B_{i-1}$ have the same density, for $0 < i \leq r$,*
- *all LDSes in $B_i \backslash B_{i-1}$ have strictly higher density than all LDSes in $B_{i+1} \backslash B_i$, for $0 < i < r$.*

Proof. It is proved in [12] that $F(\lambda)$ for any λ satisfying $\rho(B_{i+1}, B_i) < \lambda \leq \rho(B_i, B_{i-1})$ is the locally dense subgraph B_i. This suggests that $F(\lambda)$ remains to be B_i for all λ values in the range $(\rho(B_{i+1}, B_i), \rho(B_i, B_{i-1})]$. Thus, the compactness

of any connected component W of $G[B_i \backslash B_{i-1}]$ that has no edge to B_{i-1} (i.e., any LDS W in $B_i \backslash B_{i-1}$) must be $\rho(B_i, B_{i-1})$; note that, the compactness of W cannot be larger than $\rho(B_i, B_{i-1})$, since otherwise it will be in $F(\lambda')$ for $\lambda' > \rho(B_i, B_{i-1})$ contradicting that W is not in B_{i-1}. Hence, the lemma follows. □

Algorithm 3: LD(λ, X, Y)

Input: A fraction value $\lambda = \frac{a}{b}$, and a locally dense subgraph X and another subgraph Y such that $X \subsetneq F(\lambda) \subseteq Y$
Output: Locally dense subgraph $F(\lambda)$

1 Let $g \leftarrow G[Y \backslash X]$;
2 Assign a weight of b to each edge of g;
3 Add a source vertex s and a sink vertex t to g;
4 **for each** *vertex* $v \in Y \backslash X$ **do**
5 Add an undirected edge (s, v) with weight $(|E(v, Y \backslash X)| + 2|E(v, X)|) \times b$ into g;
6 Add an undirected edge (v, t) with weight $2 \times a$ into g;
7 Compute a minimum cut $(\{s\} \cup S, \{t\} \cup \bar{S})$ between s and t in g such that $|S|$ is maximized;
8 **return** $X \cup S$;

The pseudocode for computing the set of maximal λ-compact subgraphs (i.e., $F(\lambda)$) of $G = (V, E)$ is shown in Algorithm 3. Here, λ is represented as a fractional value $\frac{a}{b}$. In addition, it takes two other inputs — a locally dense subgraph X and another subgraph Y such that $X \subsetneq F(\lambda) \subseteq Y$ — which will be used for optimizing the computation. Specially, when Y is also a locally dense subgraph and $Y \supsetneq X$ and $\lambda = \rho(Y, X)$ which is the case of invoking Algorithm 3 by Algorithm 2, the condition $X \subsetneq F(\lambda) \subseteq Y$ is satisfied. Note that, we present Algorithm 3 in this general form because this general form is needed by our optimized algorithm that will be discussed in Section V-B. The inputs X and Y are used for optimizing the computation of $F(\lambda)$ as follows. Firstly, all vertices of $V \backslash Y$ can be excluded from the computation since we know that they will not be in the result. Secondly, we can also exclude X from the computation while retaining the edges between $u \in Y \backslash X$ and X as self-loops on u. As a result, we only need to conduct the computation on the subgraph $G[Y \backslash X]$, which can be small.

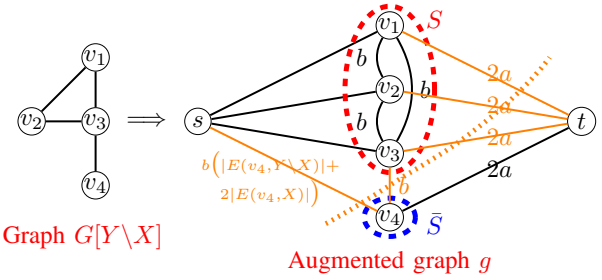


Fig. 3: Augmented graph for computing $F(\lambda)$ (for presentation simplicity, weights between s and $\{v_1, v_2, v_3\}$ are omitted)

Specifically, $F(\lambda)$ is computed via a minimum cut on the augmented graph that is constructed from $G[Y \backslash X]$ by Algorithm 3, which is also shown in Figure 3. Let $(\{s\} \cup S, \{t\} \cup \bar{S})$ be an s-t cut (i.e., a partitioning of the vertex set) in the augmented graph, see Figure 3. The value of the cut (i.e., total weight of cross partition edges) is

$$
\begin{aligned}
&\big(\textstyle\sum_{v \in \bar{S}} b(|E(v, Y \backslash X)| + 2|E(v, X)|)\big) + b|E(S, \bar{S})| + 2a|S| \\
=&\big(\textstyle\sum_{v \in Y \backslash X} b(|E(v, Y \backslash X)| + 2|E(v, X)|)\big) + b|E(S, \bar{S})| + \\
&2a|S| - \big(\textstyle\sum_{v \in S} b(|E(v, Y \backslash X)| + 2|E(v, X)|)\big) \\
=&2b\big(|E(Y \backslash X)| + |E(Y \backslash X, X)|\big) + b|E(S, \bar{S})| + 2a|S| - \\
&\big(2b|E(S)| + b|E(S, \bar{S})| + 2b|E(S, X)|\big) \\
=&2b\big(|E(Y \backslash X)| + |E(Y \backslash X, X)| + |E(X)|\big) + 2a|S| - \\
&\big(2b|E(S)| + 2b|E(S, X)| + 2b|E(X)|\big) \\
=&2b|E(Y)| + 2a|S| - 2b|E(S \cup X)| \\
=&2b|E(Y)| - 2a|X| - 2b\big(|E(S \cup X)| - \lambda|S \cup X|\big)
\end{aligned}
$$

Since $2b|E(Y)| - 2a|X|$ is constant given λ, X and Y, minimizing the value of the cut is equivalent to maximizing $|E(S \cup X)| - \lambda|S \cup X|$ for $S \subseteq Y \backslash X$. Consequently, Algorithm 3 correctly computes $F(\lambda)$.

Compared with the algorithm of [12], Algorithm 3 is different in the following two aspects. Firstly, the input Y does not need to be a locally dense subgraph. Secondly, all the edge weights of the augmented graph are integers, while [12] uses float-value weights (because $\lambda = \rho(Y, X) + \frac{1}{n^2}$) which creates a challenge for computing the minimum cut via maximum flow [13]. Note that, in Algorithm 3, the edge weights are at most $3n^2$ since $b \leq n$ which can be easily stored in 64bit integers. While it is also possible to convert the edge weights of the augmented graph of [12] to integers by multiplying them by the denominator of the fractional representation of $\rho(Y, X) + \frac{1}{n^2}$, the resulting edge weights cannot be stored in 64bit integers since they can be up to $3n^4$.

Theorem V.5. *The time complexity of running* LDS-DC$(\emptyset, V, k)$ *is* $\mathcal{O}(n^2 m)$

Proof. This can be proved in a similar way to the proof of Proposition 12 of [12]; we omit the details. The general idea is that Algorithm 3 will be invoked for at most $2r - 3$ times and each invocation takes time $\mathcal{O}(nm)$, where $r \leq n$ is the total number of locally dense subgraphs. We remark that the total time complexity of Lines 5–6 of Algorithm 2 during the whole running process is $\mathcal{O}(m)$, by noting that those connected components are disjoint. □

Example V.1. *Consider the graph G in Figure 1, Figure 4 illustrates the detailed running process of* LDS-DC$(\emptyset, V, 2)$. *Initially, $X = \emptyset$, $Y = V = \{v_1, v_2, \ldots, v_{26}\}$, and $\lambda = \rho(Y, X) = \frac{53}{26}$, which are shown in the root node of Figure 4. By invoking* LD, *we obtain $F(\frac{53}{26}) = \{v_1, v_2, \ldots, v_{13}\}$, which is not the same as Y. Thus, we split the problem into two subproblems, which are denoted by the two children of the root node in Figure 4.*

Let's first consider the subproblem with $X = \emptyset$ and $Y = F(\frac{53}{26}) = \{v_1, v_2, \ldots, v_{13}\}$. We have $\lambda = \frac{36}{13}$ and $F(\frac{36}{13}) = \{v_1, v_2, \ldots, v_8\}$ which is not the same as Y. Thus, we futher split this subproblem into two subproblems, which are denoted

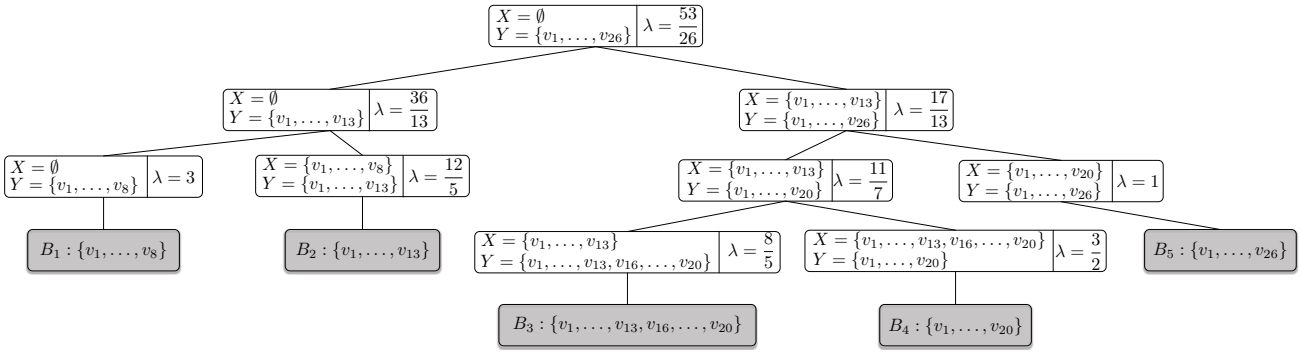


Fig. 4: Running example of LDS-DC

by its two children in Figure 4. For the left sub-problem, we have $X = \emptyset$, $Y = \{v_1, v_2, \ldots, v_8\}$, $\lambda = 3$ and $F(3) = \{v_1, \ldots, v_8\} = Y$; thus, we know that there is no locally dense subgraph in-between $X = \emptyset$ and $Y = \{v_1, v_2, \ldots, v_8\}$. Consequently, $B_1 = \{v_1, v_2, \ldots, v_8\}$ is the first locally dense subgraph and it is also an LDS. for the right sub-problem, we have $X = \{v_1, v_2, \ldots, v_8\}$, $Y = \{v_1, v_2, \ldots, v_{13}\}$, $\lambda = \frac{12}{5}$ and $F(\lambda) = \{v_1, v_2, \ldots, v_{13}\} = Y$; consequently, $B_2 = \{v_1, v_2, \ldots, v_{13}\}$ is the second locally dense subgraph. As the connected component $\{v_9, v_{10}, \ldots, v_{13}\}$ of $Y \backslash X$ have edges to $X = \{v_1, \ldots, v_8\}$, $\{v_9, v_{10}, \ldots, v_{13}\}$ is not an LDS.

Now, let's consider the second subproblem of the initial problem (i.e. the root). $X = \{v_1, v_2, \ldots, v_{13}\}$, $Y = \{v_1, v_2, \ldots, v_{26}\}$, $\lambda = \frac{17}{13}$ and $F(\frac{17}{13}) = \{v_1, v_2, \ldots, v_{20}\} \neq Y$. We thus split the problem into two subproblems and go into the left subproblem which has $X = \{v_1, v_2, \ldots, v_{13}\}$, $Y = F(\frac{17}{13}) = \{v_1, v_2, \ldots, v_{20}\}$, $\lambda = \frac{11}{7}$ and $F(\frac{11}{7}) = \{v_1, v_2, \ldots, v_{13}, v_{16}, v_{17}, \ldots, v_{20}\} \neq Y$. Consequently, we further split it into two subproblemws and go into the left one with $X = \{v_1, v_2, \ldots, v_{13}\}$, $Y = F(\frac{11}{7}) = \{v_1, v_2, \ldots, v_{13}, v_{16}, v_{17}, \ldots, v_{20}\}$, $\lambda = \frac{8}{5}$, and $F(\frac{8}{5}) = \{v_1, v_2, \ldots, v_{13}, v_{16}, v_{17}, \ldots, v_{20}\}$ which is the same as Y. Thus, $B_3 = \{v_1, v_2, \ldots, v_{13}, v_{16}, v_{17}, \ldots, v_{20}\}$ is the third locally dense subgraph. $Y \backslash X = \{v_{16}, v_{17}, \ldots, v_{20}\}$ is the second LDS since it has no edge to $X = \{v_1, v_2, \ldots, v_{13}\}$. Now, we have identified the top-2 LDSes, and we stop.

B. An Optimized Approach

From Example V.1, we can see that in the first few invocations to Algorithm 3, $X = \emptyset$ and Y is shrinking while λ is increasing; note that, the time complexity of Algorithm 3 is generally proportional to the size of $Y \backslash X$. Thus, when k is small for computing top-k LDSes, the first few invocations to Algorithm 3 may become the bottleneck. This is also confirmed by our empirical studies (see Section VI) which show that LDS-DC takes longer time than the state-of-the-art algorithm LDS to get the first few LDSes. In this subsection, we propose techniques to improve the performance, especially for reporting the first few LDSes.

The general idea of our improvement is based on the fact that $F(\lambda)$ is a subgraph of the $\lceil \lambda \rceil$-core of G [11]. Here, the l-core of G is the maximal subgraph of G whose minimum degree is at least l; the larger the value of l, the smaller the size of the l-core. Moreover, once we have obtained $F(\lambda)$ which will be a locally dense subgraph B_i for some i, the graph that we need to work on for computing $F(\lambda')$ for $\lambda' > \lambda$ will be a subgraph of B_i which could be small, and the graph that we need to work on for computing $F(\lambda'')$ for $\lambda'' < \lambda$ can all contract/remove B_i from it. Thus, it will be beneficial to efficiently compute a locally dense subgraph initially, i.e., the first λ should be large enough while ensuring that $F(\lambda) \neq \emptyset$. Motivated by this, we propose an optimized algorithm LDS-Opt in Algorithm 4, which first splits the chain of locally dense subgraphs into multiple sub-chains and then solves each sub-chain by invoking LDS-DC (Line 9).

Algorithm 4: LDS-Opt(G, k)

Input: A graph $G = (V, E)$ and an integer k
Output: Top-k LDSes $\mathcal{R}$

1 Compute a core decomposition of G;
2 Let λ be the density of a 2-approximate solution to the densest subgraph of G;
3 $\mathcal{R} \leftarrow \emptyset$;
4 $c \leftarrow \lfloor \lambda \rfloor$; $p \leftarrow n$;
5 $X \leftarrow \emptyset$; $Y \leftarrow \emptyset$;
6 **while** $c > 0$ **and** $|\mathcal{R}| < k$ **do**
7 $\quad Y \leftarrow Y \cup \{$vertices of G whose core numbers are within the range $[c, p)\}$;
8 $\quad F(c) \leftarrow \mathsf{LD}(c, X, Y)$;
9 $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \mathsf{LDS\text{-}DC}(X, F(c), k - |\mathcal{R}|)$;
10 $\quad X \leftarrow F(c)$;
11 $\quad p \leftarrow c$; $c \leftarrow \lceil \frac{c}{2} \rceil$;
12 **if** $X \neq Y$ **and** $|\mathcal{R}| < k$ **then**
13 $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \mathsf{LDS\text{-}DC}(X, Y, k - |\mathcal{R}|)$;
14 **return** $\mathcal{R}$;

LDS-Opt works as follows. We first compute a core decomposition of G [14], [15] (Line 1), which iteratively removes the minimum-degree vertex from the graph and assigns a core number for each vertex, denoted $\mathrm{core}(v)$. Note that, from the core numbers, we can efficiently obtain the l-core for any l, which simply consists of all vertices with core number at least l (Line 7). From the core decomposition, we can also obtain

a 2-approximate solution to the densest subgraph of G [16], [17], which is the one with the highest density among the n subgraphs obtained during the process. Let λ be the density of the 2-approximate solution (Line 2). We first compute $F(c)$ for $c = \lfloor \lambda \rfloor$ (Line 8) on the c-core which is obtained at Line 7, and then obtain all locally dense subgraphs (as well as LDSes contained therein) between X and $F(c)$ by invoking LDS-DC (Line 9). After solving the sub-chain between X and $F(c)$, we then solve the next sub-chain which is between $F(c)$ and $F(\lceil \frac{c}{2} \rceil)$ (Line 10–11). Note that, to solve the sub-chain between $F(c)$ and $F(\lceil \frac{c}{2} \rceil)$, all vertices of $F(c)$ are removed from the graph with self-loops being added to their neighbors. It is easy to see that the time complexity of LDS-Opt remains to be $\mathcal{O}(n^2m)$.

VI. Experiments

In this section, we evaluate the efficiency of our algorithms for computing top-k LDSes on real-world graphs. As the effectiveness of the LDS model has already been demonstrated in [11], we do not conduct effectiveness testing in this paper.

Algorithms. We compare the following algorithms.

1) LDS: the state-of-the-art algorithm proposed in [11].
2) LDS-DC: our divide-and-conquer algorithm proposed in Section V-A.
3) LDS-Opt: our optimized algorithm proposed in Section V-B.

The source code of LDS is obtained from the authors of [11]. All the three algorithms are implemented in C++ and run in main memory.[2] All experiments are conducted on a machine with an Intel(R) 3.2GHz CPU and 64GB main memory running Ubuntu 18.04.5.

TABLE II: Statistics of datasets

Dataset	$\|V\|$	$\|E\|$	d_{ave}	$\|\mathcal{R}\|$	r	Type
DBLP	317,080	1,049,866	6.62	146	1,087	Coauthor
Stanford	281,903	1,992,636	14.13	600	2,193	Web
Amazon	403,394	2,443,408	12.11	102	1,866	Trade
Google	875,713	4,322,051	9.87	3,118	3,876	Web
wiki-Talk	2,394,385	4,659,565	3.89	2,555	655	Editing
BerkStan	685,230	6,649,470	19.4	1,137	4,139	Web
as-Skitter	1,696,415	11,095,298	13.08	761	3,501	Internet
Patent	3,774,768	16,518,947	8.75	3,670	6,956	Citation
LiveJournal	4,846,609	42,851,237	17.68	1,102	9,024	Social
UK-2002	18,459,128	261,556,721	28.34	9,584	40,809	Web

Datasets. We evaluate the algorithms on 10 large real graphs from different domains, which are downloaded from the Stanford Network Analysis Platform[3]. For each graph, we removed edge directions, self-loops, and parallel edges. Statistics of these real graphs are shown in Table II, where the graphs are listed in increasing order regarding their numbers of edges; d_{ave} in the fourth column shows the average degree of the graphs, $|\mathcal{R}|$ in the fifth column shows the number of LDSes in the graphs and r in the sixth column shows the number of locally dense subgraphs in the graphs.

[2]The source code of our algorithms is open sourced at https://lijunchang.github.io/Locally_Densest_Subgraph

[3]http://snap.stanford.edu/

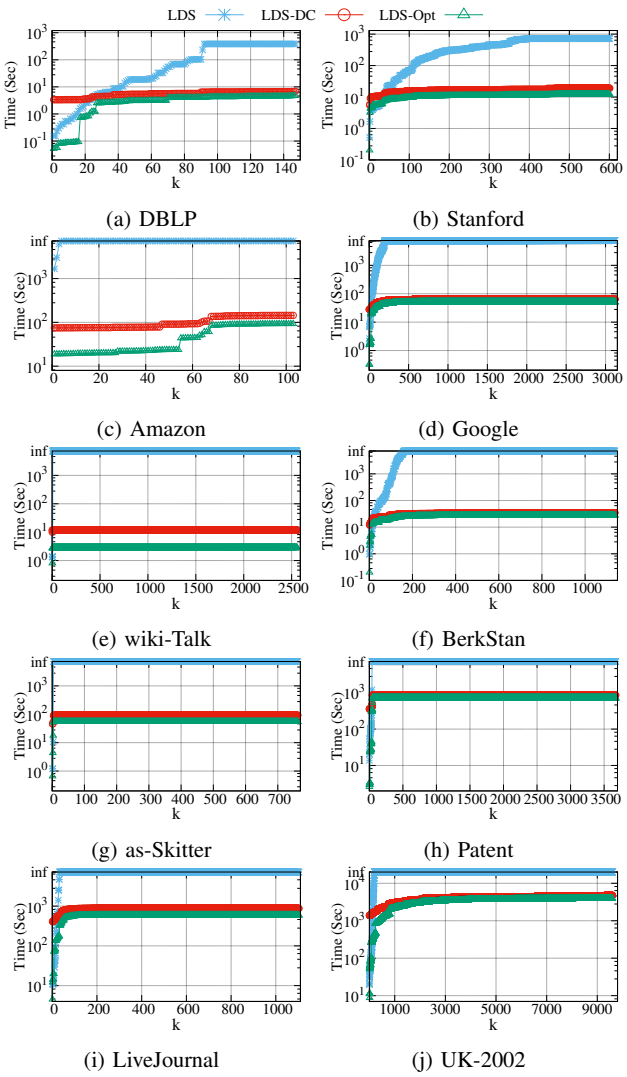


Fig. 5: Efficiency on detecting all LDSes, i.e., $k = \infty$ (best viewed in color)

Evaluation Metrics. We report the processing time, which is the total running time excluding only the I/O time of loading a graph from disk to main memory. We set a timeout of 2 hours for running an algorithm on a graph, with an exception of 5 hours for processing the largest graph UK-2002.

A. Experimental Results

Efficiency on Detecting All LDSes (i.e., $k = \infty$). In this experiment, we evaluate the efficiency of the algorithms on detecting all LDSes, by setting k to be ∞. The results are shown in Figure 5, which reports the processing time for each $k \in [1, |\mathcal{R}| + 1]$ where $|\mathcal{R}|$ is the total number of LDSes and is listed in Table II; note that, the processing time for any

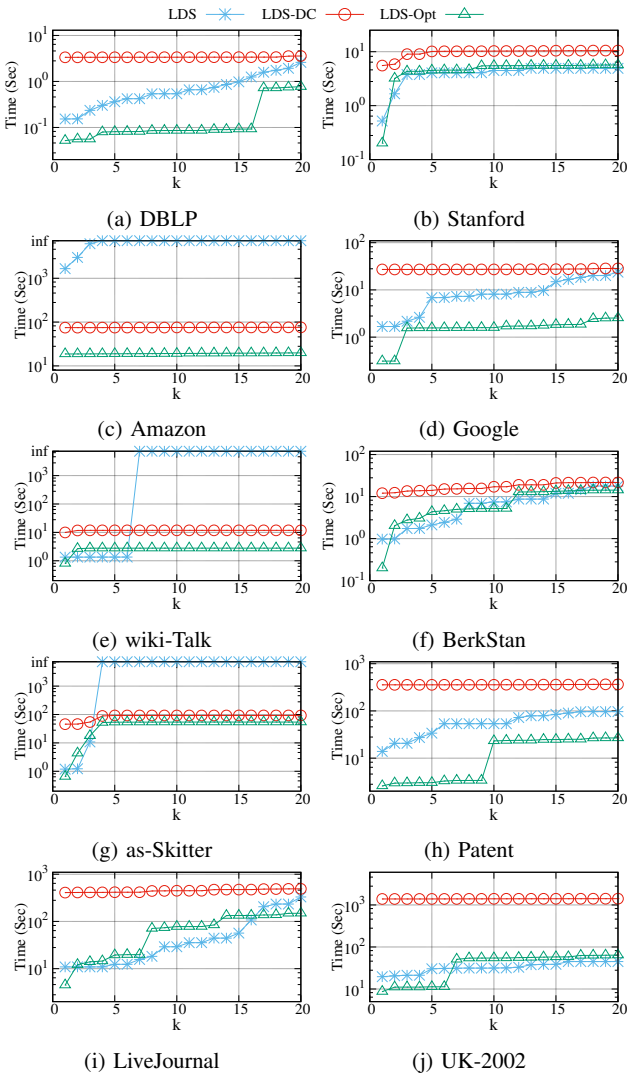


Fig. 6: Efficiency evaluation for small k ($k \leq 20$)

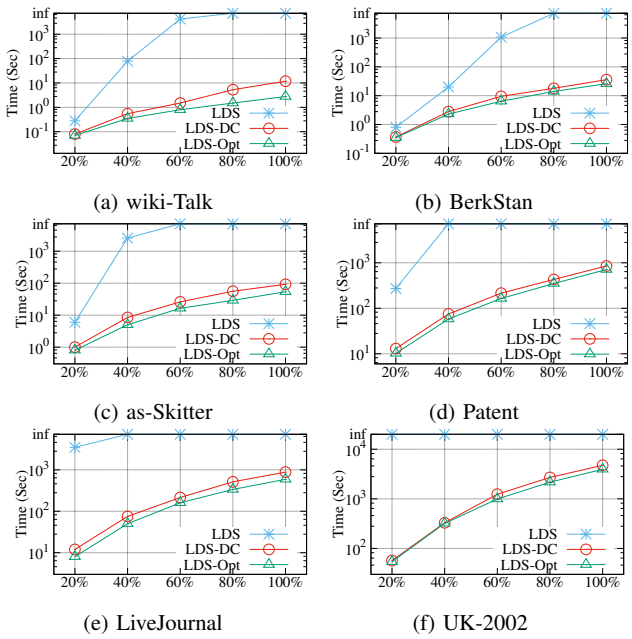


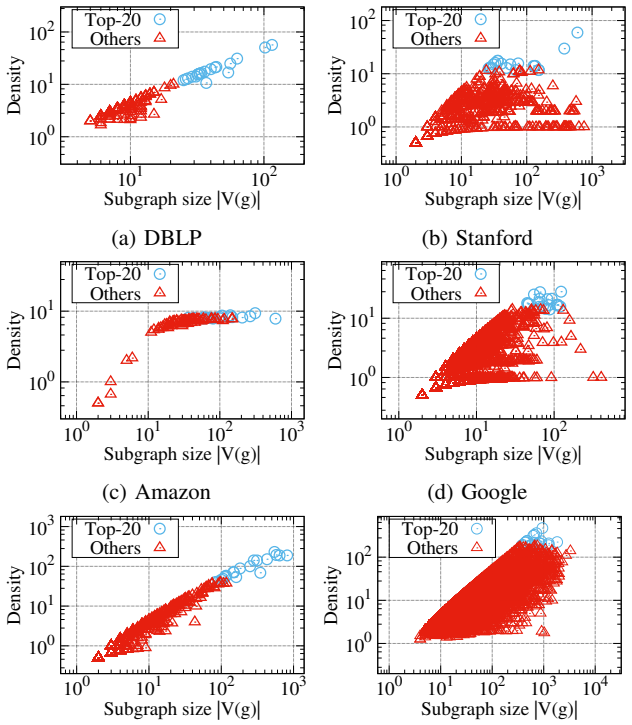
Fig. 7: Scalability evaluation ($k = \infty$, vary graph size)

$k > |\mathcal{R}| + 1$ would be the same as that for $k = |\mathcal{R}| + 1$. The state-of-the-art algorithm LDS runs the slowest and runs out-of-time for all the datasets except the two smallest ones, DBLP and Stanford, when k is ∞. This is because the lower-ranked LDSes have lower densities, which makes verification (i.e., computing $F(\lambda)$ on G) time consuming. On the other hand, both of our algorithms, LDS-DC and LDS-Opt, successfully report all the LDSes within the time limit. The improvement of our algorithms over LDS is at least two orders of magnitude on Amazon, Google, wiki-Talk, BerkStan, and as-Skitter, for $k = \infty$. Besides, our optimized algorithm LDS-Opt consistently runs faster than LDS-DC on all datasets. For example, on wiki-Talk, LDS-Opt is 4 times faster than LDS-DC on detecting all LDSes (i.e., 2.8 seconds v.s. 12 seconds). This demonstrates the efficiency of our optimization techniques.

Efficiency Evaluation for Small k. Considering that it is not easy to distinguish the algorithms in Figure 5 when k is small, we also report the processing time of the algorithms for small k values in Figure 6; specifically, $k \leq 20$. Although LDS-DC is significantly faster than LDS when k is large (see Figure 5), LDS-DC is generally slower in reporting the first few results (see Figure 6) which motivates us to design the optimized algorithm LDS-Opt. From Figure 6, we can also see that LDS-Opt runs faster than LDS for most of the cases, and the improvement can also be up to several orders of magnitude (e.g., on Amazon, wiki-Talk, and as-Skitter). Moreover, LDS-Opt consistently runs faster than LDS in reporting the top-1 result.

Scalability Evaluation. In this experiment, we evaluate the scalability of the algorithms on six of the large graphs. For each graph, we randomly sample 20%, 40%, 60%, 80% vertices and then construct the corresponding vertex-induced subgraph. The results for $k = \infty$ are reported in Figure 7. We can see that as expected, all algorithms run slower when the graph size increases. Nevertheless, our algorithms LDS-DC and LDS-Opt scale much better than the state-of-the-art algorithm LDS. Also, LDS-Opt consistently runs faster than LDS-DC. Thus, LDS-Opt has a good scalability.

Distribution of LDS Densities and Sizes. In this experiment, we report the distribution of LDSes in terms of their densities and sizes. The results are illustrated in Figure 8. We organize all detected LDSes into two groups, i.e., top-20 and non top-20, and use different colors to differentiate them. We can see that besides the top-20 LDSes, there are also many other LDSes with large size and high density. For example, on UK-2002, there are 323 LDSes with density higher than 50 and 1438 LDSes with density higher than 20. Detecting these high-



(a) DBLP (b) Stanford

(c) Amazon (d) Google

(e) LiveJournal (f) UK-2002

Fig. 8: Distribution of LDS densities and sizes

density LDSes may reveal meaningful subgraph patterns and thus be beneficial to applications. As a result, it is meaningful and necessary to identify top-k LDSes for large k values.

VII. Related Work

The problem of finding dense subgraphs from a large graph has been widely studied [1], [2], [3]. The subgraph with the largest average degree, referred to as the *densest subgraph* in the literature, can be computed exactly in $\mathcal{O}(nm \log \frac{n^2}{m})$ time by parametric network flow [9], [10]. A 2-approximation to the densest subgraph can be found in linear time by iteratively removing the vertex with the smallest degree [16], [17]. Approximately computing the densest subgraph in MapReduce, streaming and dynamic environments has also been studied [18], [19], [20]. Besides, the problem of enumerating all densest subgraphs in a large graph has been recently studied in [21] and the problem of anchored densest subgraph is studied in [22]. However, all these works only aim to find the globally densest subgraphs, while subgraphs that are locally dense in specific regions are ignored.

Recently, the notion of locally dense/densest subgraphs have been formulated to identify subgraphs that are locally dense [23], [11], [12]. Specifically, locally densest subgraph (LDS) is formulated in [11], which is also the notion we use in this paper. On the other hand, locally dense subgraph is formulated and studied in [12], [23]. LDSes are disjoint, while locally dense subgraphs form a nested structure. Both notions include the globally densest subgraph as one of their solutions, e.g., connected components of the globally densest subgraph are the top LDSes while the globally densest subgraph itself is the inner-most one in the chain of locally dense subgraphs. We in this paper are the first to demonstrate the relationship between LDSes and locally dense subgraphs, i.e., LDSes are connected components of locally dense subgraphs that do not contain parts of any other denser locally dense subgraphs. Based on this connection, we in this paper propose verification-free approaches for finding top-k LDSes that have the highest densities. Our algorithms not only have lower time complexities, but also empirically perform better than the state-of-the-art algorithm proposed in [11].

Higher-order variants of the densest subgraph problem has also been recently studied in the literature. That is, instead of maximizing the average number of edges (per vertex) in a subgraph, it maximizes the average number of l-cliques (per vertex) in a subgraph [24], [25], [26]. Note that, the higher-order variant captures classic densest subgraph (i.e., average degree-based) as a special case, since an edge is a 2-clique. Inspired by [24], higher-order version of locally densest subgraph, termed locally triangle-densest subgraph, is formulated and studied in [27]. The techniques we propose in this paper can also be used to speed up the computation of locally triangle-densest subgraphs.

Other density measures have also been used in the literature for finding dense subgraphs [3], e.g., minimum-degree based k-core computation [28], [29], triangle based k-truss computation [30], [31], [32], edge-connectivity based k-edge connected component computation [33], [34], [35], clique [36] and its relaxations (e.g., k-plex [37], n-clique [38], n-clans [39]). However, these techniques cannot be applied to compute LDSes, due to inherent different problem natures.

VIII. Conclusion

In this paper, we proposed efficient algorithms for computing top-k locally densest subgraphs (LDSes) with the highest densities. Our algorithms are designed based on a thorough investigation on the properties of maximal λ-compact subgraphs, which connect LDSes with locally dense subgraphs. As a result, our algorithms are verification free, and can be applied to efficiently obtain top-k LDSes for any k value. Our algorithms not only have lower time complexities but also empirically perform better than the state-of-the-art algorithm, and the improvement can be up to several orders of magnitude. One possible direction of future work is to incorporate convex programming techniques that are proposed in [23] to our implementation. Another possible direction of future work is to extend our implementation to compute locally triangle-dense subgraphs that are studied in [27].

Acknowledgements. Tran Ba Trung was funded by Vingroup JSC and supported by the Master, PhD Scholarship Program of Vingroup Innovation Foundation (VINIF), Institute of Big Data, code VINIF.2020.ThS.BK.04. Lijun Chang was supported by the Australian Research Council Fundings of FT180100256 and DP220103731.

REFERENCES

[1] V. E. Lee, N. Ruan, R. Jin, and C. C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. 2010.

[2] Aristides Gionis and Charalampos E. Tsourakakis. Dense subgraph discovery: KDD 2015 tutorial. In *Proc. of KDD'15*, pages 2313–2314, 2015.

[3] Lijun Chang and Lu Qin. *Cohesive Subgraph Computation over Large Sparse Graphs*. Springer Series in the Data Sciences, 2018.

[4] Jie Chen and Yousef Saad. Dense subgraph extraction with application to community detection. *IEEE Transactions on knowledge and data engineering*, 24(7):1216–1230, 2010.

[5] Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense communities in the web. In *Proc. of WWW'07*, pages 461–470, 2007.

[6] A. Angel, N. Koudas, N. Sarkas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proc. VLDB Endow.*, 5(6):574–585, 2012.

[7] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proc. of WWW'13*, pages 119–130, 2013.

[8] Eugene Fratkin, Brian T Naughton, Douglas L Brutlag, and Serafim Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

[9] Giorgio Gallo, Michael D Grigoriadis, and Robert E Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

[10] Andrew V Goldberg. *Finding a maximum density subgraph*. University of California Berkeley, 1984.

[11] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. Locally densest subgraph discovery. In *Proc. of KDD'15*, pages 965–974, 2015.

[12] Nikolaj Tatti and Aristides Gionis. Density-friendly graph decomposition. In *Proc. of WWW'15*, pages 1089–1099, 2015.

[13] Jeff Erickson. *Algorithms*. 2019.

[14] Vladimir Batagelj and Matjaz Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[15] Stephen B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[16] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

[17] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proc. of APPROX'13*, pages 84–95, 2000.

[18] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5):454–465, 2012.

[19] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proc. of STOC'15*, pages 173–182, 2015.

[20] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In *Proc. of WWW'15*, pages 300–310, 2015.

[21] Lijun Chang and Miao Qiao. Deconstruct densest subgraphs. In *Proc. of WWW'20*, pages 2747–2753, 2020.

[22] Yizhou Dai, Miao Qiao, and Lijun Chang. Anchored densest subgraph. In *Proc. of SIGMOD'22*, pages 1200–1213, 2022.

[23] M. Danisch, T.-H. H. Chan, and M. Sozio. Large scale density-friendly graph decomposition via convex programming. In *Proc. of WWW'17*, pages 233–242, 2017.

[24] Charalampos E. Tsourakakis. The k-clique densest subgraph problem. In *Proc. of WWW'15*, pages 1122–1132, 2015.

[25] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V. S. Lakshmanan, and Xuemin Lin. Efficient algorithms for densest subgraph discovery. *Proc. VLDB Endow.*, 12(11):1719–1732, 2019.

[26] Lu Chen, Chengfei Liu, Kewen Liao, Jianxin Li, and Rui Zhou. Contextual community search over large social networks. In *Proc. of ICDE'19*, pages 88–99, 2019.

[27] Raman Samusevich, Maximilien Danisch, and Mauro Sozio. Local triangle-densest subgraphs. In *Proc. of ASONAM'16*, pages 33–40, 2016.

[28] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *Proc. of KDD'10*, pages 939–948, 2010.

[29] Kai Yao and Lijun Chang. Efficient size-bounded community search over large networks. *Proc. VLDB Endow.*, 14(8):1441–1453, 2021.

[30] J. Cohen. Trusses: Cohesive subgraphs for social network analysis, 2008.

[31] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9), 2012.

[32] A. E. Sariyüce, C. Seshadhri, A. Pinar, and Ü. V. Çatalyürek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proc. of WWW'15*, pages 927–937, 2015.

[33] T. Akiba, Y. Iwata, and Y. Yoshida. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In *Proc. of CIKM'13*, 2013.

[34] L. Chang, J. X. Yu, L. Q., X. Lin, C. Liu, and W. Liang. Efficiently computing k-edge connected components via graph decomposition. In *Proc. of SIGMOD'13*, 2013.

[35] Lijun Chang and Zhiyi Wang. A near-optimal approach to edge connectivity-based hierarchical graph decomposition. *Proc. VLDB Endow.*, 15(6):1146–1158, 2022.

[36] Lijun Chang. Efficient maximum clique computation over large sparse graphs. In *Proc. of KDD'19*, pages 529–538, 2019.

[37] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978.

[38] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *CACM*, 16(9):575–576, 1973.

[39] Robert Mokken. Cliques, clubs and clans. *Quality & Quantity*, 13:161–173, 1979.