

# An Efficient and Exact Algorithm for Locally $h$ -Clique Densest Subgraph Discovery

Anonymous Author(s)

## ABSTRACT

Detecting locally, non-overlapping, near-clique densest subgraphs is a crucial problem for community search in social networks. As a vertex may be involved in multiple overlapped local cliques, detecting locally densest sub-structures considering  $h$ -clique density, i.e., *locally  $h$ -clique densest subgraph (LhCDS)* attracts great interests. This paper investigates the LhCDS detection problem and proposes an efficient and exact algorithm to list the top- $k$  non-overlapping, locally  $h$ -clique dense, and compact subgraphs. We in particular jointly consider  $h$ -clique compact number and LhCDS and design a new “Iterative Propose-Prune-and-Verify” pipeline (IPPV) for top- $k$  LhCDS detection. (1) In the proposal part, we derive initial bounds for  $h$ -clique compact numbers; prove the validity, and extend a convex programming method to tighten the bounds for proposing LhCDS candidates without missing any. (2) Then a tentative graph decomposition method is proposed to solve the challenging case where a clique spans multiple subgraphs in graph decomposition. (3) To deal with the verification difficulty, both a basic and a fast verification method are proposed, where the fast method constructs a smaller-scale flow network to improve efficiency while preserving the verification correctness. The verified LhCDSes are **returned**, while the candidates that remained unsure reenter the IPPV pipeline. (4) We further extend the proposed methods to locally more general pattern densest subgraph detection problems. We prove the exactness and low complexity of the proposed algorithm. Extensive experiments on real datasets show the effectiveness and high efficiency of IPPV.

## 1 INTRODUCTION

Finding dense subgraphs can uncover highly connected and cohesive structures in graphs, making it an effective tool for understanding complex systems. The discovery of dense subgraphs and communities has numerous applications in diverse fields including social networks [8, 12, 38], web analysis [1, 13], graph databases [18, 42], and biology [23, 32]. In these applications, the identification of near-clique subgraphs holds significant importance, as it relaxes the requirement of complete connectivity within cliques and allows for a certain degree of sparsity or missing connections while still maintaining a high level of connectivity.

Given the importance of detecting large near-clique subgraphs [37], the  $h$ -clique densest subgraph (CDS) problem that finds near-cliques formed by overlapping cliques has attracted great research attention [10, 27, 35, 37]. This is due to the fact that a vertex is generally involved in multiple overlapping cliques, such as a person may be involved in cliques as family members, office mates, etc [28]. By finding the subgraph with the highest density of  $h$ -cliques, CDS uncovers the highly-connected component that exhibits strong internal interactions [3, 24, 34]. Whereas, in the context of the real world, the discovery of a single CDS offers limited insights. Listing the top- $k$  CDSes is desired, but due to the substantial overlap

inherent in  $h$ -cliques [39, 41], the vanilla top- $k$  CDSes may refer to the same dense region, still providing limited structural insights.

Therefore, detecting the top- $k$  non-overlapping, locally maximal, dense, and compact, i.e., *locally  $h$ -clique densest subgraphs (LhCDS)* attracts great interest. For example, Figure 1 shows the relationships between a subset of characters in “Harry Potter”. The top-1 and top-2 L3CDSes are the blue and green subgraphs, respectively. The top-1 L3CDS is a family named *Weasley*, and the top-2 L3CDS is an organization named *Death Eaters*, which indicate the potential of LhCDS discovery for mining diverse dense communities.

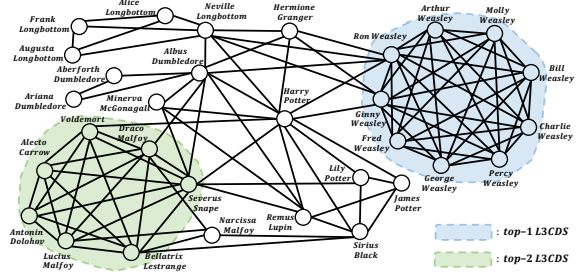


Figure 1: Part of the “Harry Potter” Network

However, no efficient and exact algorithm is known for detecting LhCDS yet. The closest work to LhCDS discovery is the locally densest subgraph (LDS) discovery [30], which is a special case of LhCDS with  $h = 2$ . But LDS only considers the density of edges, and it is challenging to generalize the techniques to arbitrary  $h$ . Firstly, the  $h$ -clique compactness is harder to evaluate, because the number of  $h$ -cliques can be several orders of magnitude more than the number of edges. Secondly, an  $h$ -clique spans over  $h$  vertices, making the subgraph division more difficult. Thirdly, verification of LhCDS is more complex than verifying LDS since the clique density and clique compactness are harder to evaluate and verify.

To address the above difficulties, we jointly consider the  $h$ -clique compact number estimation and LhCDS detection, so as to design a new *iterative propose-prune-and-verify* (IPPV) pipeline. IPPV is composed of the following iterative steps: (1) estimate bounds of  $h$ -clique compact number to propose LhCDS candidates efficiently; (2) use the bounds to prune vertices that are definitely not in any LhCDS to narrow the candidate set; (3) use verification algorithm to ensure the exactness of IPPV for extracting LhCDS; (4) remained candidates reenter the above steps until the top- $k$  LhCDSes are found. To the best of our knowledge, this paper is the first to explore the LhCDS detection. The key contributions of IPPV are as follows:

- (1) The initial  $h$ -clique compact number bounds are proposed based on the structures of graphs, and we show that a convex programming, which provides  $h$ -clique diminishingly dense decomposition, can be extended to tighten the bounds.
- (2) We propose a tentative graph decomposition method to deal with the case when a clique is spanning multiple subgraphs to generate correct decomposition proposals.

- (3) Efficient verification is the critical part since the verification is complex for verifying both the  $h$ -clique density and  $h$ -clique compactness. We propose a novel fast verification algorithm by carefully constructing a size-reduced flow network and using the maximum flow algorithm. We prove the correctness and efficiency of the proposed fast verification algorithm.
- (4) At last, we further extend the *iterative propose-prune-and-verify* pipeline to detect locally general pattern densest subgraphs. More than six patterns are investigated, showing the potential of detecting locally more general pattern densest subgraphs.

We theoretically verify the exactness and efficiency of the proposed algorithm and conduct extensive experiments with different quality measures on large real datasets to evaluate the algorithm.

## 2 RELATED WORK

### 2.1 Densest Subgraph

The **algorithms** to the densest subgraph (DS) problem can be classified into two categories: exact algorithms and approximation algorithms. The DS problem can be solved in polynomial time by exact methods based on maximum flow, linear programming, or convex optimization. Picard et al. [29] and Goldberg [14] **first** introduced the maximum-flow-based exact algorithm for the densest subgraph problem. Charikar [6] proposed an LP-based exact algorithm for the DS problem. The convex-optimization-based exact algorithm is proposed by Danisch et al. [9] and can handle graphs containing tens of billions of edges. Fang et al. [10] improved the efficiency of the flow-based exact algorithm by locating the densest subgraph in a specific  $k$ -core. Exact algorithms cannot scale well to large graphs, so a large number of **works** on faster approximation algorithms for the DS problem are also presented.

Charikar [6] proposed a 2-approximation algorithm for the DS problem, which is known as the greedy peeling algorithm. Proving that the  $k_{max}$ -core is a 2-approximation solution to the DS problem, Fang et al. [10] improved the greedy peeling algorithm based on  $k_{max}$ -core. Inspired by the multiplicative weights update method, Boob et al. [4] designed an iterative version of the greedy peeling algorithm. Based on the MapReduce model, Bahmani et al. [2] proposed an  $2(1 + \epsilon)$ -approximation algorithm, where  $\epsilon > 0$ . Based on the dual of Charikar’s LP relaxation, Harb et al. [16] presented a new iterative algorithm for the DS problem. Chekuri et al. [7] proposed a flow-based approximation algorithm for the DS problem.

The DS problem has various variants focusing on different aspects and different types of graphs. Two recent surveys [20, 25] detail different variations of the DS problem and their applications to different types of graphs, such as directed graphs [6], labeled graphs [11], and uncertain graphs [43].

### 2.2 $h$ -clique Densest Subgraph

Tsourakakis [37] defined the notion of  $h$ -clique density and introduced the  $h$ -clique densest subgraph (CDS) problem. Mitzenmacher et al. [27] presented a sampling scheme called the densest subgraph sparsifier, yielding a randomized algorithm that produces a well-approximate solution to the CDS problem. Fang et al. [10] proposed more efficient exact and approximation algorithms for the CDS problem. Sun et al. [35] aimed at developing near-optimal and exact algorithms for the CDS problem on large real-world graphs. They

modified the Frank-Wolfe algorithm for CDS to their algorithm **kClist++** and proved the effectiveness of the proposed algorithm.

### 2.3 Locally Densest Subgraph

The locally densest subgraph (LDS) problem is a variant of the densest subgraph (DS) problem. Qin et al. [30] proposed a method to discover the top- $k$  representative locally densest subgraphs of a graph. The method involves defining a parameter-free definition of an LDS, showing that the set of LDSes in a graph can be computed in polynomial time, and proposing three novel pruning strategies to reduce the search space of the algorithm. Trung et al. [36] observed the hierarchical structure of maximal  $\rho$ -compact subgraphs and presented verification-free approaches to improve the efficiency of finding top- $k$  LDSes. Ma et al. [26] proposed a convex-programming-based algorithm called LDScvx to the LDS problem by introducing the concept of the compact number and using the relations of compactness to the LDS problem and a specific convex program. Capitalizing on previous results [30], Samusevich et al. [33] studied the local triangle densest subgraph (LTDS) problem, which extended the LDS model to triangle-based density. It’s worth noting that, in essence, LDS is a special instance of  $Lh$ CDS when  $h = 2$ ; LTDS is a special instance of  $Lh$ CDS when  $h = 3$ .

## 3 PRELIMINARIES

Given an undirected graph  $G = (V, E)$ , we use  $\psi_h(V_{\psi_h}, E_{\psi_h})$  to denote an  $h$ -clique with  $|V_{\psi_h}|$  vertices and  $|E_{\psi_h}|$  edges.  $\Psi_h(G)$  is the collection of  $h$ -cliques of  $G$ .  $d_{\psi_h}(G)$  denotes the  $h$ -clique density of  $G$ ,  $d_{\psi_h}(G) = \frac{|\Psi_h(G)|}{|V|}$ , and  $deg_G(v, \psi_h)$  is the  $h$ -clique degree of  $v$ , i.e., the number of  $h$ -cliques containing  $v$ . Given a subset  $S \subseteq V$ ,  $G[S] = (S, E(S))$  is the subgraph induced by  $S$ , and  $E(S) = E(G) \cap (S \times S)$ . Table 1 summarizes the main notations used in this paper.

Table 1: MAIN NOTATIONS

Notation	Definition
$G = (V, E)$	a graph with vertex set $V$ and edge set $E$
$n, m$	$n =  V , m =  E $
$G[S]$	the subgraph induced by $S$
$\Psi_h(G)$	the collection of <b>all</b> $h$ -cliques of $G$
$\psi_h(V_{\psi_h}, E_{\psi_h})$	an $h$ -clique ( $V_{\psi_h}$ is vertex set, $E_{\psi_h}$ is edge set)
$d_{\psi_h}(G)$	the $h$ -clique density of $G$ , $d_{\psi_h}(G) = \frac{ \Psi_h(G) }{ V }$
$\phi_h(u)$	the $h$ -clique compact number of vertex $u$
$deg_G(v, \psi_h)$	the $h$ -clique degree of vertex $v$ in $G$
$\bar{\phi}_h(u)$	an upper bound of $\phi_h(u)$ in $G$
$\underline{\phi}_h(u)$	a lower bound of $\phi_h(u)$ in $G$
$CP^l(G, h)$	the convex programming of $G$ for $h$ -clique densest <b>subgraph</b>
$\alpha$	the weights distributed from $h$ -cliques to vertices
$r$	the weights received by each vertex

A densest subgraph in a local region not only means that such a subgraph is not included in any other denser subgraph, but also requires the inner density to be compact and evenly distributed. Qin et al. [30] proposed the concept of  $\rho$ -compact, which gives a reasonable definition of locally densest subgraph. A graph  $G$  is  $\rho$ -compact when removing any subset  $S$  from  $G$  removes at least  $\rho \times |S|$  edges. Considering the  $h$ -clique density in a graph, we define an  $h$ -clique  $\rho$ -compact graph as:

**Definition 1 ( $h$ -clique  $\rho$ -compact).** A graph  $G = (V, E)$  is  $h$ -clique  $\rho$ -compact if  $G$  is connected, and removing any subset of

vertices  $S \subseteq V$  will result in the removal of at least  $\rho \times |S|$   $h$ -cliques in  $G$ , where  $\rho$  is a non-negative real number.

If  $G$  is  $h$ -clique  $\rho$ -compact, then the  $h$ -clique degree of each vertex in  $G$  is at least  $\lceil \rho \rceil$ , because removing any vertex will remove at least  $\rho$   $h$ -cliques. Besides, the  $h$ -clique density of an  $h$ -clique  $\rho$ -compact graph is at least  $\rho$ . For any  $\hat{\rho} > \rho$ , an  $h$ -clique  $\hat{\rho}$ -compact graph is also an  $h$ -clique  $\rho$ -compact graph, so we define the  $h$ -clique compactness of a graph  $G$  as the largest  $\rho$  such that  $G$  is  $h$ -clique  $\rho$ -compact. A subgraph  $G[S]$  of  $G$  is a maximal  $h$ -clique  $\rho$ -compact subgraph if none of the supergraphs of  $G[S]$  is  $h$ -clique  $\rho$ -compact.

**Proposition 1.** *If a graph  $G$  has  $h$ -clique density  $d_{\psi_h}(G)$ , then the  $h$ -clique compactness of the graph is at most  $d_{\psi_h}(G)$ , i.e.,  $\rho \leq d_{\psi_h}(G)$ .*

**PROOF.** Suppose the compactness of  $G$  is higher than  $d_{\psi_h}(G)$ , then removing all vertices in  $G$  will result in the removal of more than  $d_{\psi_h}(G) \times |V|$   $h$ -cliques, which means that the  $h$ -clique density of  $G$  must be higher than  $d_{\psi_h}(G)$ , and **contradicts** the fact that the  $h$ -clique density of  $G$  is  $d_{\psi_h}(G)$ .  $\square$

Proposition 1 clarifies that the  $h$ -clique compactness of a graph cannot be greater than  $d_{\psi_h}(G)$ . We are then interested in finding the locally  $h$ -clique dense and compact subgraph  $G[S]$  in  $G$ . We formally define a locally  $h$ -clique densest subgraph as follows.

**Definition 2 (Locally  $h$ -clique densest subgraph (LhCDS)).**

A subgraph  $G[S]$  of  $G$  is a locally  $h$ -clique densest subgraph of  $G$  if the following two conditions hold:

1.  $G[S]$  is an  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraph;
2. There does not exist a supergraph  $G[S']$  of  $G[S]$  ( $S' \supsetneq S$ ), such that  $G[S']$  is also  $h$ -clique  $d_{\psi_h}(G[S'])$ -compact.

**Proposition 2 (Disjoint property).** Suppose  $G[S]$  and  $G[S']$  are two LhCDSes in  $G$ , we have  $S \cap S' = \emptyset$ .

**PROOF.** Without loss of generality, we assume  $d_{\psi_h}(G[S]) \geq d_{\psi_h}(G[S'])$ . We prove the proposition by contradiction. Suppose  $S \cap S' \neq \emptyset$ . According to the definition of LhCDS,  $G[S] \not\subseteq G[S']$ . Since  $G[S]$  and  $G[S']$  are two LhCDSes, the graph induced by  $S \cup S'$  is a connected  $h$ -clique  $d_{\psi_h}(G[S'])$ -compact graph which is larger than  $S'$ . This contradicts the fact that  $G[S']$  is an LhCDS.  $\square$

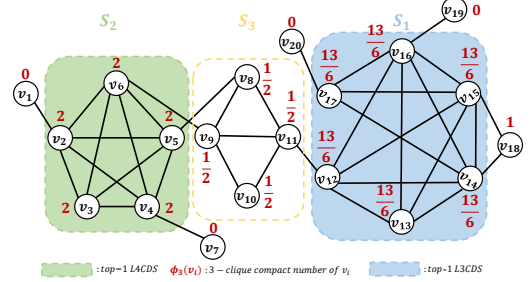
Proposition 2 proves that all LhCDSes in a graph  $G$  are pairwise disjoint. Therefore, the number of LhCDSes of  $G$  is bounded by  $|V|$ , and the LhCDSes can be used to identify all the non-overlapping  $h$ -clique dense regions of a graph.

Most applications in the real world usually require finding the top- $k$  dense regions of a graph [30], so we focus on finding the top- $k$  LhCDSes with the largest densities. When  $k$  is large enough, all LhCDSes can be found. We formulate the problem as follows.

**Definition 3 (Locally  $h$ -clique densest subgraph Problem (LhCDS Problem)).** Given a graph  $G$ , an integer  $h$ , and an integer  $k$ , the locally  $h$ -clique densest subgraph problem is to compute the top- $k$  LhCDSes ranked by the  $h$ -clique density in  $G$ .

Figure 2 shows an example of the LhCDS. We use  $S_1, S_2$ , and  $S_3$  to represent  $\{v_{12}, \dots, v_{17}\}$ ,  $\{v_2, \dots, v_6\}$ , and  $\{v_8, \dots, v_{11}\}$ . When  $h = 3$ , the top-1 L3CDS is  $G[S_1]$ , which has a 3-clique density of  $\frac{13}{6}$ , since

there are thirteen 3-cliques in it. The top-1 and top-2 L4CDSes are  $G[S_2]$  and  $G[S_1]$ . They both have a 4-clique density of 1.



**Figure 2: An example of the locally  $h$ -clique densest subgraph**

Note that an edge in a graph  $G$  is a 2-clique; therefore, the intensively studied LDS problem [26, 30] can be seen as an instance of the LhCDS problem when  $h = 2$ . Similarly, the LTDS problem [33] is exactly the L3CDS problem. Therefore, the LhCDS problem studied in this paper provides a more general framework, and we boldly infer that our method can be generalized from  $h$ -clique to general patterns, which means that we can give an algorithmic framework to solve a wider range of locally pattern densest problems.

## 4 LhCDS DISCOVERY

In this section, we focus on the design of an LhCDS discovery algorithm. According to the concept of  $h$ -clique compactness, each subgraph of a graph  $G$  has its own compactness. However, a vertex may be contained in various subgraphs with different compactness. Therefore, we introduce the concept of  $h$ -clique compact number for each vertex in a graph.

**Definition 4 ( $h$ -clique compact number).** Given a graph  $G = (V, E)$ , for each vertex  $u \in V$ , the  $h$ -clique compact number of  $u$  is the largest  $\rho$  such that  $u$  is contained in an  $h$ -clique  $\rho$ -compact subgraph of  $G$ , denoted by  $\phi_h(u)$ .

In the following theorem, we prove the relationship between the LhCDS and the  $h$ -clique compact numbers of vertices within it.

**Theorem 1.** Given an LhCDS  $G[S]$  in  $G$ , for each vertex  $u \in S$ , the  $h$ -clique compact number of  $u$  is equal to the  $h$ -clique density of  $G[S]$ , i.e.,  $\phi_h(u) = d_{\psi_h}(G[S])$ .

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

Based on Theorem 1, once we get the  $h$ -clique compact number of each vertex in  $G$ , we can **arrange the vertices in descending order based on the  $h$ -clique compact number and then check whether**



the vertices with the same  $h$ -clique compact number satisfy the definition of LhCDS to obtain top- $k$  LhCDSes. For example, in Figure 2, we list the 3-clique compact numbers of all vertices of  $G$ . It is obvious that  $G[S_1]$  and  $G[S_2]$  are both L3CDSes.

However, computing the  $h$ -clique compact numbers directly is difficult. So we jointly consider  $h$ -clique compact number and LhCDS to design a new “iterative propose-prune-and-verify” pipeline for top- $k$  LhCDS detection, which is called IPPV. In the proposal part, the true LhCDSes are allowed to be encapsulated in the proposed candidates, but without missing true LhCDSes. Proper graph decomposition methods should be designed, since a clique may span multiple subgraphs to be decomposed. In the verification part, each correct LhCDS should be outputted, and LhCDS candidates that can be further pruned should be indicated.

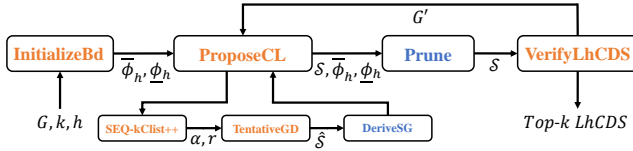


Figure 3: Flow diagram of IPPV

Figure 3 gives the flow diagram of IPPV. It has four main parts: 1) calculate the initial bounds of the  $h$ -clique compact numbers of vertices; 2) iteratively propose all LhCDS candidates (generating approximate  $h$ -clique compact numbers; decomposing the graph tentatively; grouping vertices and tightening bounds); 3) prune invalid vertices; 4) verify the locally densest property of all candidates to find top- $k$  LhCDSes, and we, in particular, propose a basic algorithm and a fast algorithm for verification. As a general algorithm framework, all blue parts are extensions of existing methods, while all orange parts are our proof and innovation for this problem.

#### 4.1 Initial $h$ -clique Compact Number Bounds

In order to derive LhCDS candidates, we first give initial upper and lower bounds of  $h$ -clique compact number  $\phi_h(u)$ . Specifically, we denote  $\bar{\phi}_h(u)$  and  $\underline{\phi}_h(u)$  as the upper and lower bound of  $\phi_h(u)$  in  $G$ . We use  $(k, \psi_h)$ -core[10], which is a cohesive subgraph model, to compute the initial bounds.

**Definition 5** ( $(k, \psi_h)$ -core). Given a graph  $G$ , the  $(k, \psi_h)$ -core is the largest subgraph of  $G$ , in which the  $h$ -clique degree of each vertex is at least  $k$ . The  $h$ -clique-core number of a vertex  $u \in V$ , denoted by  $core_G(u, \psi_h)$ , is the highest  $k$  of  $(k, \psi_h)$ -core containing  $u$ .

**Proposition 3.**  $h$ -clique compact number  $\phi_h(u)$  has the following relations to the  $h$ -clique-core number  $core_G(u, \psi_h)$ .

- (1) A  $(k, \psi_h)$ -core subgraph is  $h$ -clique  $\frac{k}{h}$ -compact. Thus, for any  $u \in V$ ,  $\underline{\phi}_h(u)$  can be assigned as  $\frac{core_G(u, \psi_h)}{h}$ ;
- (2) If  $G[S]$  is an LhCDS of  $G$ , for all  $u \in S$ ,  $core_G(u, \psi_h) \geq d_{\psi_h}(G[S])$ . Thus, for any  $u \in V$ ,  $\bar{\phi}_h(u)$  can be assigned as  $core_G(u, \psi_h)$ .

**PROOF.** Any vertex in a  $(k, \psi_h)$ -core subgraph is contained in at least  $k$   $h$ -cliques. By removal of any subset  $S$  from the  $(k, \psi_h)$ -core, at least  $\frac{k}{h} \times |S|$   $h$ -cliques would be removed. For any  $u \in V$ , there is an  $h$ -clique  $\frac{core_G(u, \psi_h)}{h}$ -compact subgraph of  $G$  that contains  $u$ ,

then  $\frac{core_G(u, \psi_h)}{h}$  is a lower bound of  $\phi_h(u)$ . The second relation can be obtained from the fact that an LhCDS  $G[S]$  in a graph  $G$  is a  $(\lceil d_{\psi_h}(G[S]) \rceil, \psi_h)$ -core subgraph of  $G$ . For any  $u \in V$ , if an LhCDS contains  $u$ , then  $core_G(u, \psi_h)$  is an upper bound of  $\phi_h(u)$ .  $\square$

---

#### Algorithm 1: Bound initialization: InitializeBd

---

**Input:**  $G = (V, E), h$

**Output:**  $\bar{\phi}_h, \underline{\phi}_h$

- 1 **foreach**  $u \in V$  **do** compute  $core_G(u, \psi_h)$ ;
  - 2 **foreach**  $u \in V$  **do**
  - 3    $\bar{\phi}_h(u) \leftarrow core_G(u, \psi_h); \underline{\phi}_h(u) \leftarrow \frac{core_G(u, \psi_h)}{h};$
  - 4 **return**  $\bar{\phi}_h, \underline{\phi}_h$ ;
- 

According to Proposition 3, we can get the initial bounds of  $h$ -clique compact number  $\phi_h(u)$  of  $G$  (Lines 2-3) by Algorithm 1.

#### 4.2 Candidate LhCDS Proposal

The initial upper and lower bounds for  $h$ -clique compact numbers from  $h$ -clique-core numbers are relatively loose. In this section, we focus on how to tighten the bounds and propose LhCDS candidates.

**4.2.1 Overall Algorithm for Candidate LhCDS Proposal.** The overall candidate LhCDS proposal algorithm is given in Algorithm 2. Approximate  $h$ -clique compact number is calculated via SEQ-kClist++ (Line 1); the preliminary partition of  $G$  and recalculated values are obtained via TentativeGD (Line 2); tighter upper and lower bounds for  $h$ -clique compact numbers and the further partition of  $G$  (stable  $h$ -clique group) are calculated via DeriveSG (Line 3). The sub-procedures introduce each of the above functions (Lines 5-33).

**4.2.2 Generate Approximate  $h$ -clique Compact Number.** Inspired by a classical convex programming [9, 26], we propose a convex programming for finding the diminishingly- $h$ -clique-dense decomposition, and prove that the optimal solution of our convex programming is exactly the  $h$ -clique compact number of a graph  $G$ .

Intuitively, the aim of  $CP(G, h)$  is that each  $h$ -clique  $\psi_h \in \Psi_h(G)$  tries to distribute its unit weight among its  $h$  vertices such that the sum of the weight received by the vertices are as even as possible. We use  $\alpha_{u, \psi_h}$  to represent the weight assigned to  $u$  from  $h$ -clique  $\psi_h$  and  $r(u)$  to denote the sum of the weights assigned to  $u$  from  $h$ -cliques that contain  $u$ . This intuition suggests that we can consider the objective function:  $Q_{G, h}(\alpha) := \sum_{u \in V} r(u)^2$ , in which  $r(u) = \sum_{\psi_h \in \Psi_h(G): u \in \psi_h} \alpha_{u, \psi_h}$ , for all  $u \in V$ . The convex programming is:

$$CP(G, h) := \min \{Q_{G, h}(\alpha) : \alpha \in \mathcal{D}(G, h)\},$$

where the domain is:

$$\mathcal{D}(G, h) := \left\{ \alpha \in \prod_{\psi_h \in \Psi_h(G)} \mathbb{R}_+^{\psi_h} : \forall \psi_h \in \Psi_h(G), \sum_{u \in \psi_h} \alpha_{u, \psi_h} = 1 \right\}.$$

Here, we demonstrate that the  $h$ -clique compact numbers can be derived from the optimal solution of  $CP(G, h)$ .

**Theorem 2.** Let  $(\alpha^*, r^*)$  be an optimal solution to  $CP(G, h)$ . Then,  $\forall u \in V$ ,  $\phi_h(u) = r^*(u)$ , i.e., each  $r^*(u)$  in  $r^*$  is exactly the  $h$ -clique compact number of  $u$ .

**Algorithm 2:** Candidate LhCDS proposal: ProposeCL

---

**Input:**  $G = (V, E)$ , number of iterations  $T$ ,  $\bar{\phi}_h, \underline{\phi}_h$

**Output:**  $S, \bar{\phi}_h, \underline{\phi}_h$

```

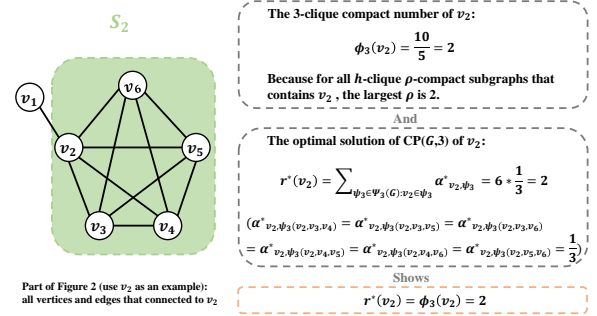
1  $(\alpha, r) \leftarrow \text{SEQ-kClist++}(G, T)$ ;
2  $\hat{S}, \alpha, r \leftarrow \text{TentativeGD}(G, \alpha, r)$ ;
3  $S, \bar{\phi}_h, \underline{\phi}_h \leftarrow \text{DeriveSG}(\hat{S}, \alpha, r, \bar{\phi}_h, \underline{\phi}_h)$ ;
4 return  $S, \bar{\phi}_h, \underline{\phi}_h$ ;
5 Procedure SEQ-kClist++( $G, T$ )
6   foreach  $h$ -clique  $\psi_h$  in  $G$  do  $\alpha_{u, \psi_h} \leftarrow \frac{1}{h}, \forall u \in V_{\psi_h}$ ;
7   foreach  $u \in V$  do  $r(u) \leftarrow \sum_{\psi_h \in \Psi_h(G): u \in \psi_h} \alpha_{u, \psi_h}$ ;
8   foreach iteration  $t = 1, \dots, T$  do
9      $\gamma_t \leftarrow \frac{1}{t+1}; \alpha \leftarrow (1 - \gamma_t) * \alpha; r \leftarrow (1 - \gamma_t) * r$ ;
10    foreach  $h$ -clique  $\psi_h$  do
11       $v_{\min} \leftarrow \arg\min_{v \in \psi_h} r(v)$ ;
12       $\alpha_{v_{\min}, \psi_h} \leftarrow \alpha_{v_{\min}, \psi_h} + \gamma_t; r(v_{\min}) \leftarrow r(v_{\min}) + \gamma_t$ ;
13  return  $(\alpha, r)$ ;
14 Procedure TentativeGD( $G, \alpha, r$ )
15  sort vertices in  $V$  in descending order according to  $r$ ;
16   $P \leftarrow \{p | p = \arg\max_{p \leq q \leq n} d_{\psi_h}(G[V[1:q]])\}$ ;
17   $\hat{S} \leftarrow$  partition  $V$  according to  $P$ ;
18  foreach  $\psi_h \in \Psi_h(G)$  do
19     $p \leftarrow \max\{1 \leq i \leq l : \psi_h \cap \hat{S}_i \neq \emptyset\}$ ;
20     $s \leftarrow \sum_{u \in \psi_h \setminus \hat{S}_p} \alpha_{u, \psi_h}$ ;
21     $\forall u \in \psi_h \setminus \hat{S}_p, \alpha_{u, \psi_h} \leftarrow 0$ ;
22     $\forall u \in \psi_h \cap \hat{S}_p, \alpha_{u, \psi_h} \leftarrow \alpha_{u, \psi_h} + \frac{s}{|\psi_h \cap \hat{S}_p|}$ ;
23  foreach  $u \in V$  do  $r(u) \leftarrow \sum_{\psi_h \in \Psi_h(G): u \in \psi_h} \alpha_{u, \psi_h}$ ;
24  return  $\hat{S}, \alpha, r$ ;
25 Procedure DeriveSG( $\hat{S}, \alpha, r, \bar{\phi}_h, \underline{\phi}_h$ )
26  while  $\hat{S}$  is not empty do
27     $S' \leftarrow$  pop out the first candidate from  $\hat{S}; S \leftarrow S \cup S'$ ;
28    if  $S$  is a stable  $h$ -clique group then put  $S$  into  $S$ ;  $S \leftarrow \emptyset$ ;
29  foreach  $S \in S$  do
30    foreach  $u \in S$  do
31       $\bar{\phi}_h(u) \leftarrow \min\{\bar{\phi}_h(u), \max_{v \in S'} r(v)\}$ ;
32       $\underline{\phi}_h(u) \leftarrow \max\{\underline{\phi}_h(u), \min_{v \in S'} r(v)\}$ ;
33  return  $S, \bar{\phi}_h, \underline{\phi}_h$ ;

```

---

**PROOF.** For any vertex  $u \in V$ , let  $S^+ = \{v \in V | r^*(v) > r^*(u)\}$ ,  $S^- = \{v \in V | r^*(v) = r^*(u)\}$ ,  $S^- = \{v \in V | r^*(v) < r^*(u)\}$ ,  $u \in S^-$ .  $S^{+*}$  denotes the vertices that are contained by  $h$ -cliques that including vertices both in  $S^+$  and  $S^-$ . We prove  $G[S^+ \cup S^-]$  is an  $h$ -clique  $r^*(u)$ -compact subgraph. First, removing  $S^-$  from  $G[S^+ \cup S^-]$  will result in the removal of  $r^*(u) \times |S^-|$  cliques in  $G[S^+ \cup S^-]$ . We know that for all  $(v, w) \in E \cap (S^+ \times S^-)$ ,  $r^*(v) > r^*(w)$  and  $\alpha_{v, \psi_h}(v, w \in \psi_h) = 0$ . Otherwise, if there exists  $(v, w) \in E \cap (S^+ \times S^-)$  such that  $\alpha_{v, \psi_h}(v, w \in \psi_h) > 0$ , there exists  $r^*(v) - r^*(w) > \epsilon > 0$ . We can reduce  $\alpha_{v, \psi_h}(v, w \in \psi_h)$  by  $\epsilon$  and increase  $\alpha_{w, \psi_h}(v, w \in \psi_h)$  by  $\epsilon$ , and the objective function be decreased by  $2\epsilon(r^*(v) - r^*(w) - \epsilon)$ , which contradicts the optimality of  $r^*$ . Similarly, we can prove that for all  $(v, w) \in E \cap (S^- \times S^-)$ ,  $r^*(v) > r^*(w)$  and  $\alpha_{v, \psi_h}(v, w \in \psi_h) = 0$ . Therefore,  $r^*(u) \times |S^-| = \sum_{\psi_h \in \Psi_h(G): v \in S^-, v \in \psi_h} \alpha_{v, \psi_h} = |\Psi_h(G[S^+])| \cup$

$\Psi_h(G[S^+])$ .  $r^*(u) \times |S^-|$  is exactly the number of  $h$ -cliques to be removed when removing  $S^-$  from  $G[S^+ \cup S^-]$ . Meanwhile, for any  $S' \subseteq S^+ \cup S^-$ , we have that  $r^*(u) \times |S'| \leq \sum_{\psi_h \in \Psi_h(G): v \in S', v \in \psi_h} \alpha_{v, \psi_h} \leq \sum_{\psi_h \in \Psi_h(G[S^+ \cup S^-]): v \in S', v \in \psi_h} 1$ , which means removing any  $S' \subseteq S^+ \cup S^-$  from  $G[S^+ \cup S^-]$  will result in the removal of at least  $r^*(u) \times |S'|$   $h$ -cliques. Therefore,  $G[S^+ \cup S^-]$  is an  $h$ -clique  $r^*(u)$ -compact subgraph. Analogously, we can prove that for any other subset  $S''$  containing  $u$ ,  $G[S'']$  is an  $h$ -clique  $\rho$ -compact subgraph, where  $\rho \leq r^*(u)$ , by contradiction. Therefore,  $r^*(u)$  is the largest  $\rho$  such that  $u$  is contained in an  $h$ -clique  $\rho$ -compact subgraph of  $G$ , which is exactly the  $h$ -clique compact number of  $u$ .  $\square$



**Figure 4:** An example of the relationship between  $r^*(u)$  and  $\phi_h(u)$  of a vertex  $u$  in  $G$

Consider the convex programming  $\text{CP}(G, 3)$  for  $G$  in Figure 2, we use  $v_2$  as an example, shown in Figure 4. The 3-clique compact number of  $v_2$  is 2, and the optimal solution  $r^*(v_2)$  value is also 2. It is clear that for each  $u \in V$ ,  $r^*(u)$  is exactly  $\phi_h(u)$ .

**Theorem 3.** The locally  $h$ -clique densest subgraph problem can be solved in polynomial time for any given positive integer  $h$ .

**PROOF.** According to Theorem 2, the LhCDS problem is equivalent to  $\text{CP}(G, h)$ .  $\text{CP}(G, h)$  is a convex quadratic programming problem, which has polynomial time solutions [5, 15]. Therefore, the LhCDS problem is polynomial-time solvable.  $\square$

However,  $\text{CP}(G, h)$  is a problem with  $|\Psi_h|$  equality constraints and  $n \cdot |\Psi_h|$  variables. The time complexity of the classical polynomial-time solution of  $\text{CP}(G, h)$  is  $O((n \cdot |\Psi_h|)^3 \cdot L)$  [15], which requires long running time. Therefore, exactly attaining the  $(\alpha^*, r^*)$  is difficult, so we use the approximate solution  $(\alpha, r)$  of  $\text{CP}(G, h)$  to tighten the  $h$ -clique compact bounds. Frank-Wolfe-based (FW-based) algorithm is efficient for finding approximate solutions to  $\text{CP}(G)$  [26]. However, FW-based algorithm for  $h$ -clique densest requires a large amount of memory. SEQ-kClist++[35] is better for approximately calculating  $\alpha_{u, \psi_h}$  for each  $h$ -clique  $\psi_h$ ,  $u \in \psi_h$ , as well as  $r(u)$  for each vertex  $u$ . All  $\alpha_{u, \psi_h}$  are initialized to  $\frac{1}{h}$  (Line 6).  $r(u)$  stores the sum over all  $\alpha_{u, \psi_h}$ 's such that  $\psi_h$  contains  $u$  (Line 7). At each iteration,  $\alpha$  and  $r$  are modified simultaneously as follows. For each  $h$ -clique  $\psi_h$ , we find the minimum  $r(v_{\min})$  among  $\psi_h$ , and new values for the  $\alpha_{v_{\min}, \psi_h}$  and the  $r(v_{\min})$  are computed as convex combinations (Lines 8-12).

**4.2.3 Tentative Graph Decomposition.** After getting approximate  $(\alpha, r)$ , we can derive a graph decomposition from the given  $(\alpha, r)$ .

**Proposition 4.** Given an LhCDS  $G[S]$  in  $G$ ,  $\forall (u, v) \in E$ , if  $u \in S$  and  $v \in V \setminus S$ , we have  $\phi_h(u) > \phi_h(v)$ .

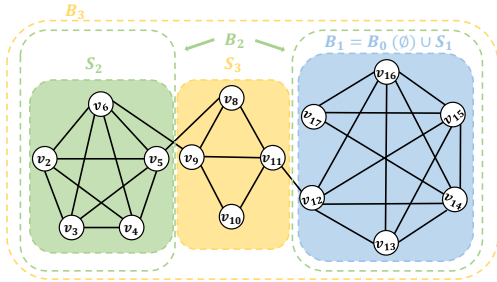
Considering vertices adjacent to  $G[S_1]$  but not in  $G[S_1]$  in Figure 2, such as  $v_{11}$  and  $v_{18}$ , their 3-clique compact numbers fulfill Proposition 4:  $\phi_3(v_{11}) = \frac{1}{2} < \frac{13}{6}$ ,  $\phi_3(v_{18}) = 1 < \frac{13}{6}$ . Proposition 4 is helpful for choosing LhCDSes from all subgraphs.

We then propose TentativeGD to generate tentative graph decomposition for proposing LhCDS. The vertices in  $V$  are sorted based on  $r$  values descendingly (Line 15). The initial partition  $\hat{S}$  of the graph is extracted based on the descending order (Lines 16-17). For each  $\psi_h \in \Psi_h(G)$ , if the clique  $\psi_h$  is contained in multiple vertex sets, the vertex set with the largest set index will be recorded as  $p$ , and the  $\alpha$  value of  $\psi_h$  will be redistributed to vertices in  $\hat{S}_p$  (Lines 18-22). In other words, for the convenience of partition, the  $\alpha$  value of  $\psi_h$  straddling multiple vertex sets is redistributed to a vertex set with the lowest  $r$  value. Finally, the  $r$  values of all vertices in  $V$  are recalculated (Line 23).

**4.2.4 Stable  $h$ -clique Group Derivation.** After getting the initial bounds of  $h$ -clique compact numbers in InitializeBd and a preliminary partition of the graph in TentativeGD, we consider obtaining tighter bounds of  $h$ -clique compact numbers and a further partition of the graph, to calculate LhCDS candidates. Inspired by two concepts, stable subset [9] and stable group [26], for solving the  $h$ -clique densest subgraph problem, we propose the definition of the stable  $h$ -clique group.

**Definition 6 (stable  $h$ -clique group).** Given a feasible solution  $(\alpha, r)$  to  $CP(G, h)$ , a stable  $h$ -clique group with respect to  $(\alpha, r)$  is a non-empty vertex group  $S \in V$  satisfying the following conditions:

- (1) For any  $v \in V \setminus S$ ,  $r(v) > \max_{u \in S} r(u)$  or  $r(v) < \min_{u \in S} r(u)$ ;
- (2) For any  $v \in V$ , if  $r(v) > \max_{u \in S} r(u)$ ,  $\forall \psi_h(u, v \in \psi_h)$ ,  $\alpha_{v, \psi_h} = 0$ ;
- (3) For any  $v \in V$ , if  $r(v) < \min_{u \in S} r(u)$ ,  $\forall \psi_h(u, v \in \psi_h)$ ,  $\alpha_{u, \psi_h} = 0$ .



**Figure 5: The relationship between stable 3-clique subset  $\mathcal{B}$  and stable 3-clique group  $\mathcal{S}$**

The concept of stable  $h$ -clique subset  $\mathcal{B}$  is related to stable  $h$ -clique group  $\mathcal{S}$ , and the relationship between stable  $h$ -clique subset and stable  $h$ -clique group can be shown in Figure 5 with  $h = 3$ . All stable  $h$ -clique groups are disjoint, and a stable  $h$ -clique subset is the union of the previous stable  $h$ -clique subset and the first stable  $h$ -clique group outside this previous stable  $h$ -clique subset. Either  $\mathcal{B}$  or  $\mathcal{S}$  can form a consecutive subsequence of the whole sequence, and we only use the stable  $h$ -clique group in our algorithm.

**Theorem 4.** Given a feasible solution  $(\alpha, r)$  to  $CP(G, h)$  and a stable  $h$ -clique group  $S$  with respect to  $(\alpha, r)$ , for all  $v \in S$ , we have that  $\min_{u \in S} r(u) \leq \phi_h(v) \leq \max_{u \in S} r(u)$ .

**PROOF.** According to Theorem 2, for all  $u \in V$ ,  $r^*(u) = \phi_h(u)$ . Suppose there exists a vertex  $v \in S$  such that  $r^*(v) = \phi_h(v) < \min_{u \in S} r(u) \leq r(v)$ . Since  $\sum_{u \in V} r(u) = \sum_{u \in V} r^*(u)$ , correspondingly, there must exist another vertex  $w \in V$ ,  $r^*(w) = \phi_h(w) > r(w)$ . The difference between  $r(w)$  and  $r^*(w)$  means that there exists  $\psi_h$  containing both  $v$  and  $w$ ,  $\alpha_{v, \psi_h} > 0$ . Since  $S$  is a stable  $h$ -clique group, according to the third condition in Definition 6,  $r(w) > \min_{u \in S} r(u)$ . There exists  $\epsilon > 0$ , we can increase  $r^*(v)$  by  $\epsilon$  and decrease  $r^*(w)$  by  $\epsilon$  to decrease the value of the objective function. This contradicts that  $r^*$  is the optimal solution to  $CP(G, h)$ . By the same token, for all  $u \in V$ ,  $r^*(u) = \phi_h(u) \leq \max_{u \in S} r(u)$ .  $\square$

Based on Theorem 4, the stable  $h$ -clique groups can give tighter bounds of  $h$ -clique compact numbers, so we propose DeriveSG algorithm to derive the stable  $h$ -clique groups, which are our LhCDS candidates. In DeriveSG, the subsets in  $\hat{S}$  are checked one by one; if the subset is a stable  $h$ -clique group, it will be pushed into the set of stable  $h$ -clique groups  $\mathcal{S}$ ; otherwise, in the next iteration, the current subset  $S$  will be merged with the next subset  $S'$  (Lines 26-28). Then, the upper and lower bounds of  $h$ -clique compact numbers are updated based on Theorem 4 (Lines 29-32).

### 4.3 Pruning for Candidate LhCDS Derivation

We prove that the following proposition can help to prune invalid vertices that are certainly not contained by any LhCDS.

**Proposition 5.** For any  $v \in V$ ,  $v$  is not contained by any LhCDS in  $G$  if either of the following two conditions is satisfied.

- (1) If there exists  $(u, v) \in E$ , such that  $\phi_h(u) > \bar{\phi}_h(v)$ ,  $v$  is invalid;
- (2) Let  $G'$  denote the graph after pruning all invalid vertices in condition (1).  $\bar{\phi}_h^G(u)$  is the upper bound of  $\phi_h^G(u)$  in  $G$ . For any  $u$  in  $G'$ , if  $\bar{\phi}_h^{G'}(v) < \phi_h(u)$ ,  $v$  is invalid.

**PROOF.** First, we prove condition (1). For any  $u, v \in V$ ,  $(u, v) \in E$ , if  $\phi_h(u) > \bar{\phi}_h(v)$ , then  $\phi_h(u) > \phi_h(v)$ . According to Proposition 4,  $v$  is not contained in LhCDS, i.e.,  $v$  is invalid.

For condition (2),  $\bar{\phi}_h^{G'}(u) < \phi_h(u)$  means that to form an  $h$ -clique  $\phi_h(u)$ -compact subgraph containing  $u$ , some already pruned vertices are needed. So using the vertices in  $G'$  only cannot form an  $h$ -clique  $\phi_h(u)$ -compact subgraph containing  $u$ . Therefore,  $u$  cannot be contained by any LhCDS in  $G$ , i.e.,  $v$  is invalid.  $\square$

According to Proposition 5, we design Pruning Rule to prune invalid vertices by condition (1) and condition (2). An example can be seen in Figure 2 with  $h = 3$ .  $v_9$  and  $v_{11}$  can be pruned, because for edge  $(v_6, v_9)$ ,  $\phi_3(v_6) = 2 > \bar{\phi}_3(v_9) = \frac{1}{2}$ ; for edge  $(v_{11}, v_{12})$ ,  $\phi_3(v_{12}) = \frac{13}{6} > \bar{\phi}_3(v_{11}) = \frac{1}{2}$ . Analogously, the vertices  $v_1$ ,  $v_7$ ,  $v_{18}$ ,  $v_{19}$ , and  $v_{20}$  are also pruned by condition (1).

We denote the graph after pruning by  $G'$ . Some vertices in  $G'$  become invalid vertices, because any LhCDS in  $G$  containing these vertices needs to include some already pruned vertices, which can not form an LhCDS in  $G'$ . Therefore, we utilize condition (2). Based on Proposition 3, for any vertex  $u \in V(G')$ ,  $\text{core}_G(u, \psi_h)$  provides an upper bound of  $\phi_h^G(u)$ . For example, after  $v_9$  and  $v_{11}$  are pruned, the upper bounds of 3-clique compact numbers of  $v_8$  and  $v_{10}$  in

graph  $G'$  are  $\bar{\phi}_3^{G'}(v_8) = \bar{\phi}_3^{G'}(v_{10}) = 0 < \underline{\phi}_3(v_8) = \underline{\phi}_3(v_{10}) = \frac{1}{2}$ . So  $v_8$  and  $v_{10}$  are pruned using condition (2).

We propose Prune algorithm shown in Algorithm 3.  $G$  is replicated to  $G'$  for pruning (Line 1). Condition (1) is used to remove invalid vertices in  $G'$  (Lines 2-3); after computing the  $h$ -clique core numbers for all vertices in  $G'$  (Line 4), condition (2) is applied to further remove invalid vertices in  $G'$  (Lines 5-7). Finally, the LhCDS candidates are updated from the intersection of  $h$ -clique stable groups and the unpruned vertex sets (Line 8).

---

**Algorithm 3:** Pruning algorithm: Prune

---

**Input:**  $G = (V, E)$ ,  $S$ ,  $\bar{\phi}_h$ ,  $\underline{\phi}_h$

**Output:**  $S$

```

1  $G' = (V(G'), E(G')) \leftarrow G$ ;
2 foreach  $(u, v) \in E$  do
3   if  $\bar{\phi}_h(v) < \underline{\phi}_h(u)$  then remove  $v$  from  $G'$ ;
4 foreach  $u \in V(G')$  do compute  $core_{G'}(u, \psi_h)$ ;
5 while there exists  $u \in V(G')$ ,  $core_{G'}(u, \psi_h) < \underline{\phi}_h(u)$  do
6   remove  $u$  from  $G'$ ;
7   update  $h$ -clique-core numbers of vertices adjacent to  $u$ ;
8 foreach LhCDS candidate  $S \in S$  do  $S \leftarrow S \cap V(G')$ ;
9 return  $S$ ;
```

---

#### 4.4 LhCDS Verification

Since candidate LhCDSes are obtained approximately, we need to confirm whether the candidates are LhCDSes.

**Proposition 6.** *An LhCDS must satisfy the following properties:*

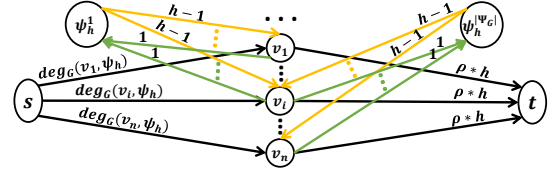
- (1) Any subgraph of an LhCDS cannot be denser than itself;
- (2) An LhCDS itself is compact, and any supergraph of an LhCDS cannot be more compact than itself.

**PROOF.** (2) of Proposition 6 can be directly obtained from the definition of LhCDS. We prove (1) by contradiction. Suppose that there is a subgraph  $G[S']$  in an LhCDS  $G[S]$ ,  $S' \subset S$ , such that  $d_{\psi_h}(G[S']) > d_{\psi_h}(G[S])$ . By removal of the set  $U = S \setminus S'$  from  $G[S]$ , we remove  $|\Psi_h(G[S])| - |\Psi_h(G[S'])|$   $h$ -cliques. Note that  $|\Psi_h(G[S])| - |\Psi_h(G[S'])| = d_{\psi_h}(G[S])|S| - d_{\psi_h}(G[S'])|S'| < d_{\psi_h}(G[S])(|S| - |S'|) = d_{\psi_h}(G[S])|U|$ , which contradicts the fact that  $G[S]$  is an LhCDS, i.e.  $h$ -clique  $d_{\psi_h}(G[S])$ -compact.  $\square$

We need to verify: **1)** whether a candidate LhCDS  $G[S]$  is self-densest and **2)** whether  $G[S]$  is a maximal  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraph in  $G$ . We use IsDensest [35] algorithm to check whether a candidate LhCDS  $G[S]$  is self-densest. In this section, we focus on the verification of the second property, to verify whether  $G[S]$  is a connected component of maximal  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraphs in  $G$ . We design a basic verification algorithm, and we further propose a fast algorithm by reducing the scale of the flow network. The correctness of both algorithms is proved.

**4.4.1 Basic Verification Algorithm.** Given an LhCDS candidate  $G[S]$ , we propose an innovative flow network to derive maximal  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraph  $G'$  in  $G$ . If  $G[S]$  is a connected

component of  $G'$ ,  $G[S]$  is indeed maximal  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraph and an LhCDS in  $G$ ; otherwise,  $G[S]$  is not an LhCDS. The flow network  $\mathcal{F}(V_{\mathcal{F}}, E_{\mathcal{F}})$  is shown in Figure 6. The vertex set of  $\mathcal{F}$  is  $\{s\} \cup V \cup \Psi_h \cup \{t\}$ . The arc set of  $\mathcal{F}$  is given as follows. For each  $h$ -clique  $\psi_h^j$ , we add  $h$  incoming arcs of capacity 1 from the vertices which form  $\psi_h^j$ , and  $h$  outgoing arcs of capacity of  $h - 1$  to the same set of vertices. For each vertex  $v_i \in V$ , we add an incoming arc of capacity  $deg_G(v_i, \psi_h)$  from the source vertex  $s$ , and an outgoing arc of capacity  $\rho * h$  to the sink vertex  $t$ . Given a parameter  $\rho$ , we prove that the flow network in DeriveCompact can be used to derive maximal  $h$ -clique  $\rho$ -compact subgraphs in  $G$  according to Theorem 5.



**Figure 6:** The flow network of DeriveCompact( $G, \rho, \theta$ )

**Theorem 5.** *If  $G$  contains maximal  $h$ -clique  $\rho$ -compact subgraphs, then the result returned by DeriveCompact( $G, \rho - \frac{1}{|V|^2}, \theta$ ) is the set of all maximal  $h$ -clique  $\rho$ -compact subgraphs in  $G$ .*

**PROOF.** Based on Proposition 2, two LhCDSes are disjoint. We use  $G[S_1]$  to represent the union of all maximal  $h$ -clique  $\rho$ -compact subgraphs in  $G$ .  $G[S_2]$  denotes the subgraph returned by DeriveCompact( $G, \rho - \frac{1}{|V|^2}, \theta$ ), which is the largest subgraph in  $G$  with maximum  $|\Psi_h(G[S_2])| - \rho \times |S_2|$  [14][10]. We prove that  $G[S_1]$  and  $G[S_2]$  are the same. First, we prove that  $G[S_2]$  is a subgraph of  $G[S_1]$  by contradiction. Suppose a connected component  $G[S]$  of  $G[S_2]$  is not  $h$ -clique  $\rho$ -compact, then there exists a subset  $S' \subseteq S$  such that removing  $S'$  from  $S$  will result in removing less  $h$ -cliques than  $\rho \times |S'|$ , then  $|\Psi_h(G[S])| - |\Psi_h(G[S'])| < \rho \times |S'| = \rho \times (|S| - |S \setminus S'|)$ . We have  $|\Psi_h(G[S])| - \rho \times |S| < |\Psi_h(G[S \setminus S'])| - \rho \times |S \setminus S'|$ . Therefore, replacing  $G[S]$  by its subgraph  $G[S \setminus S']$  in  $G[S_2]$  will enlarge the value of  $|\Psi_h(G[S_2])| - \rho \times |S_2|$ , which contradicts the condition that  $G[S_2]$  has the maximum  $|\Psi_h(G[S_2])| - \rho \times |S_2|$ . Second, we prove that  $G[S_1]$  is a subgraph of  $G[S_2]$  by contradiction. Suppose  $G[S_1]$  is not a subgraph of  $G[S_2]$ , according to the result before, we have  $S_2 \subset S_1$ . There exists a subset  $S \neq \emptyset$  and  $S = S_1 \setminus S_2$ . Removing  $S$  from  $G[S_1]$  will result in removing at least  $\rho \times |S|$   $h$ -cliques, then  $|\Psi_h(G[S_1])| - |\Psi_h(G[S_2])| \geq \rho \times |S| = \rho \times (|S_1| - |S_2|)$ . We have  $|\Psi_h(G[S_1])| - \rho \times |S_1| \geq |\Psi_h(G[S_2])| - \rho \times |S_2|$ , so enlarging  $G[S_2]$  to  $G[S_1]$  will not decrease the value of  $|\Psi_h(G[S_2])| - \rho \times |S_2|$ , which contradicts the condition that  $G[S_2]$  has the maximum  $|\Psi_h(G[S_2])| - \rho \times |S_2|$ . Therefore, the theorem is proved.  $\square$

In Algorithm 4, we first derive all connected components of the  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraph  $G'$  in  $G$  by DeriveCompact (Line 2). If  $G[S]$  is a connected component of  $G'$ , the algorithm returns True (Line 3). In DeriveCompact, all the instances of  $h$ -clique is collected (Line 5). To build a flow network  $\mathcal{F}(V_{\mathcal{F}}, E_{\mathcal{F}})$ , a vertex set  $V_{\mathcal{F}}$  is created, and vertices in  $V_{\mathcal{F}}$  are linked by directed edges with different capacities (Lines 6-19). Then, the minimum  $s - t$  cut  $(S, \mathcal{T})$  is computed (Line 20).



**Algorithm 4:** Basic LhCDS verification algorithm

---

**Input:**  $G(V, E), S$   
**Output:** VerifyLhCDS

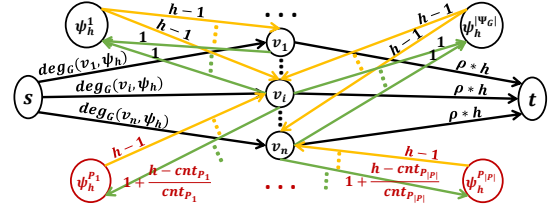
```

1  $\rho \leftarrow d_{\psi_h}(G[S]), \text{VerifyLhCDS} \leftarrow \text{True};$ 
2  $G' \leftarrow \text{DeriveCompact}(G, \rho - \frac{1}{|V|^2}, \emptyset);$ 
3 return  $G[S]$  is a connected component in  $G'$ ;
4 Procedure DeriveCompact( $G, \rho, P$ )
5    $\text{cnt} \leftarrow 0; \Psi_h \leftarrow$  all the instances of  $h$ -clique  $\psi_h$  in  $G$ ;
6    $V_{\mathcal{F}} \leftarrow \{s\} \cup V \cup \Psi_h \cup P \cup \{t\};$ 
7   foreach  $\psi_h \in \Psi_h$  do
8     foreach  $v \in \psi_h$  do
9       add an edge  $\psi_h \rightarrow v$  with capacity  $h - 1$ ;
10      add an edge  $v \rightarrow \psi_h$  with capacity 1;
11   foreach  $\psi_h \in P$  do
12      $\text{cnt} \leftarrow \text{cnt of } \psi_h$ ;
13     foreach  $v \in \psi_h$  and  $v \in G$  do
14       add an edge  $\psi_h \rightarrow v$  with capacity  $h - 1$ ;
15       add an edge  $v \rightarrow \psi_h$  with capacity  $1 + \frac{h - \text{cnt}}{\text{cnt}}$ ;
16        $\text{deg}_G(v, \psi_h) \leftarrow \text{deg}_G(v, \psi_h) + 1 + \frac{h - \text{cnt}}{\text{cnt}}$ ;
17   foreach  $v \in V$  do
18     add an edge  $v \rightarrow t$  with capacity  $\rho * h$ ;
19     add an edge  $s \rightarrow v$  with capacity  $\text{deg}_G(v, \psi_h)$ ;
20   Compute the minimum  $s - t$  cut  $(S, T)$  from the flow network
     $\mathcal{F}(V_{\mathcal{F}}, E_{\mathcal{F}})$ ;
21   return  $G[S \setminus s]$ ;
```

---

**4.4.2 Fast Verification Algorithm.** Although the basic verification algorithm can successfully verify whether a given subset is LhCDS, the scale of the flow network in algorithm 4 is large, and the running time is long in large-scale graphs. We prove that the verification can be done by verifying only the subgraph  $G[S]$  and the vertices around the subgraph  $G[S]$ , which is denoted by  $G[T]$ . Since  $G[T]$  is much smaller than  $G$ , checking the minimum cut in  $G[T]$  is much more efficient. Considering the complexity of the overlap of cliques, we propose a fast verification algorithm by constructing a smaller flow network based on  $G[T]$ . Based on the fact that only the  $h$ -cliques at the boundary of  $G[T]$  affect  $h$ -clique compact numbers in  $G[T]$  compared to the  $h$ -clique compact numbers in  $G$ , we use a set  $P$  to record these  $h$ -cliques. For each  $\psi_h^{P_r} \in P$ , the number of vertices that are contained in both  $\psi_h^{P_r}$  and  $G[T]$  is  $\text{cnt}_{P_r}$ . The flow network  $\mathcal{F}(V_{\mathcal{F}}, E_{\mathcal{F}})$  is shown in Figure 7. The vertex set of  $\mathcal{F}$  is  $\{s\} \cup V \cup \Psi_h \cup P \cup \{t\}$ . We add the boundary  $h$ -clique set  $P$  into  $V_{\mathcal{F}}$  to ensure that the results of solving the flow network of  $G[T]$  are precisely consistent with that of  $G$ . The arc set of  $\mathcal{F}$  is given as follows. The arcs for vertices and  $h$ -cliques in  $G[T]$  are the same as the former flow network. For each  $\psi_h^{P_r} \in P$ , we add  $\text{cnt}_{P_r}$  incoming arcs of capacity  $1 + \frac{h - \text{cnt}_{P_r}}{\text{cnt}_{P_r}}$  from the vertices that both in  $\psi_h^{P_r}$  and  $G[T]$ , and  $\text{cnt}_{P_r}$  outgoing arcs of capacity of  $h - 1$  to the same set of vertices.

In Algorithm 5, the  $h$ -clique density of  $G[S]$  is assigned to  $\rho$  (Line 1). Then, a breadth-first search is performed.  $U$  is used to store the vertices to be traversed (Line 4). The first vertex  $v$  from  $U$  is popped out (Line 6), and all  $h$ -cliques containing  $v$  are iterated (Lines 7-25). For each  $\psi_h$  that is not in  $W$ , if any vertex  $w$  in  $\psi_h$  satisfying

Figure 7: The flow network of DeriveCompact( $G, \rho, P$ )**Algorithm 5:** Fast LhCDS verification algorithm

---

**Input:**  $G = (V, E), S, \Psi_h(G), \bar{\phi}_h, \underline{\phi}_h$   
**Output:** VerifyLhCDS

```

1  $\rho \leftarrow d_{\psi_h}(G[S]), \text{VerifyLhCDS} \leftarrow \text{True}, \text{Valid} \leftarrow \text{True};$ 
2  $U \leftarrow$  an empty queue,  $P \leftarrow \emptyset, T \leftarrow \emptyset, W \leftarrow \emptyset, \text{cnt} \leftarrow 0$ ;
3 foreach  $u \in S$  do
4   if  $u \notin T$  then push  $u$  to  $U$ , insert  $u$  into  $T$ ;
5   while  $U$  is not empty do
6      $v \leftarrow$  pop out the front vertex in  $U$ ;
7     foreach  $\psi_h \in \Psi_h(G)$  where  $v \in \psi_h$  do
8       Valid  $\leftarrow \text{True}$ ;
9       if  $\psi_h \notin W$  then
10         foreach  $w \in \psi_h$  do
11           if  $\bar{\phi}_h(w) < \rho$  then Valid  $\leftarrow \text{False}$ ;
12           insert  $\psi_h$  into  $W$ ;
13         else Valid  $\leftarrow \text{False}$ ;
14         if Valid then
15            $\text{cnt} \leftarrow 1$ ;
16           foreach  $w \in \psi_h$  and  $w \neq v$  do
17             if  $w \notin T$  and  $w$  is in any LhCDS then
18               VerifyLhCDS  $\leftarrow \text{False}$ ;
19             if  $\underline{\phi}_h(w) \leq \rho$  then
20               if  $w \notin T$  then
21                 push  $w$  to  $U$ , insert  $w$  into  $T$ ;
22                $\text{cnt} \leftarrow \text{cnt} + 1$ ;
23             if  $\text{cnt} \neq h$  and  $\psi_h \notin P$  then
24               insert  $\psi_h$  and  $\text{cnt}$  into  $P$ ;
25               VerifyLhCDS  $\leftarrow \text{False}$ ;
26         foreach  $(v, w) \in E$  do
27           if  $w \notin T$  and  $\bar{\phi}_h(w) > \rho$  then
28             VerifyLhCDS  $\leftarrow \text{False}$ ;
29           else if  $w \notin T$  and  $\bar{\phi}_h(w) > \rho$  then
30             push  $w$  to  $U$ , add  $w$  into  $T$ ;
31 if VerifyLhCDS then return True;
32  $G' \leftarrow \text{DeriveCompact}(G[T], \rho - \frac{1}{|V(G[T])|^2}, P);$ 
33 return  $G[S]$  is a connected component in  $G'$ ;
```

---

$\bar{\phi}_h(w) < \rho$ ,  $\psi_h$  will not affect the  $h$ -clique compact number of  $w$  (Lines 9-13); if any vertex  $w$  is in any outputted LhCDS, False is assigned to VerifyLhCDS (Lines 17-18); the number of vertices in  $\psi_h$  satisfying  $\bar{\phi}_h(w) \leq \rho$  is recorded and  $\psi_h$  is added into  $P$  (Lines 19-25). All neighbors of  $v$  are iterated (Lines 26-30). For each neighbor  $w$  that is not in  $T$ , if  $\bar{\phi}_h(w) > \rho$ , False is assigned to VerifyLhCDS (Line 28). If  $\underline{\phi}_h(w) \leq \rho < \bar{\phi}_h(w)$ ,  $w$  will be added into  $U$  and  $T$



(Line 30). If `VerifyLhCDS` is False, a subgraph  $G[T]$  induced by  $T$  and peripheral  $h$ -cliques in  $P$  are used to compute all  $h$ -clique  $\rho$ -compact subgraphs in  $G[T]$  via min-cut (Line 32). Finally, True is returned if  $G[S]$  is maximal  $h$ -clique  $\rho$ -compact; otherwise, the algorithm returns False (Line 33). The flow network here is much smaller.

**Theorem 6.** *Given a graph  $G$  and a self-densest subgraph  $G[S]$ ,  $G[S]$  is an LhCDS of  $G$  if and only if the fast LhCDS verification algorithm returns True.*

**PROOF.** On the one hand, if  $G[S]$  is an LhCDS of  $G$ ,  $G[S]$  is still an LhCDS in  $G[T]$ , because only the  $h$ -cliques in  $P$  might increase the  $h$ -clique compact numbers in  $G[T]$  compared to the  $h$ -clique compact numbers in  $G$ . Otherwise, there exists a vertex  $v$  with  $h$ -cliques in  $P$  contained in the maximal  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraph containing  $G[S]$ , and we can construct a larger  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraph in  $G$  by adding vertices with  $\phi_h(w) > d_{\psi_h}(G[S])$  connected to  $v$ , which contradicts that  $G[S]$  is an LhCDS. On the other hand, if  $G[S]$  is not an LhCDS of  $G$ , we will find a larger  $h$ -clique  $d_{\psi_h}(G[S])$ -compact subgraph containing  $G[S]$  in  $G[T]$ . Therefore,  $G[S]$  is not an LhCDS in  $G[T]$ , and the algorithm returns False. Therefore, the algorithm returns True only when  $G[S]$  is an LhCDS of  $G$ .  $\square$

#### 4.5 The LhCDS Discovery Algorithm (IPPV)

Combining all the algorithms above, we derive the LhCDS discovery algorithm, called the IPPV algorithm shown in Algorithm 6. An empty stack  $st$  is initialized, and  $G'$  is assigned to  $G$  (Line 1). The bounds of  $h$ -clique compact numbers are initialized via `InitializeBd` (Line 2). LhCDS candidates are derived via `ProposeCL` and `Prune` (Line 4-5). Next, the LhCDS candidates in  $\mathcal{S}$  are reversely pushed into  $st$ , and the first LhCDS candidate in  $st$ , the one with the highest  $\phi_h$  value, is popped out (Lines 6-7). The LhCDS candidate is verified by `IsDensest` (Line 8) and `VerifyLhCDS` (line 9). If  $G[S]$  is an LhCDS, it will be outputted, and  $k$  is decreased by 1 (Line 10). If  $G[S]$  is not an LhCDS but is self-densest,  $S$  is updated as the top LhCDS candidate from  $st$  (Line 12). Then,  $G[S]$  is assigned to  $G'$  for the next iteration (Line 13). The above process is repeated until top- $k$  LhCDSes are found (line 3) or the stack is empty (Line 11). Our algorithms can also be extended to find all LhCDSes.

**Theorem 7.** *The LhCDS discovery algorithm IPPV is an exact algorithm, i.e., it can output all LhCDSes correctly.*

**PROOF.** According to Theorem 4, the set of LhCDS candidates proposed by IPPV is a superset of all LhCDSes with tight bounds of  $h$ -clique compact number. Based on Proposition 5, the pruning part only prunes vertices that are not in any LhCDS and guarantees that the vertices in any LhCDS will not be pruned. Theorem 5 and Theorem 6 state that the verification part verifies whether a candidate is exactly an LhCDS without misjudgment. And all candidates are repeated through the above bound updating, pruning, and verification operations until  $k$  LhCDSes are outputted. Since the set of LhCDS candidates are arranged in descending order, the outputted  $k$  LhCDSes must be the top- $k$  LhCDSes. Therefore, the proposed algorithm is an exact algorithm to output the top- $k$  LhCDSes.  $\square$

---

#### Algorithm 6: Iterative propose-prune-and-verify algorithm based on convex programming (IPPV)

---

**Input:**  $G = (V, E)$ , number of iterations  $T$ , an integer  $k$

**Output:** top- $k$  LhCDSes

```

1  $st \leftarrow$  an empty stack;  $G' \leftarrow G$ ;
2  $\bar{\phi}_h, \underline{\phi}_h \leftarrow$  InitializeBd( $G', h$ );
3 while  $k > 0$  do
4    $S, \bar{\phi}_h, \underline{\phi}_h \leftarrow$  ProposeCL( $G', T, \bar{\phi}_h, \underline{\phi}_h$ );
5    $S \leftarrow$  Prune( $G', S, \bar{\phi}_h, \underline{\phi}_h$ );
6   foreach  $S \in \mathcal{S}$  reversely do push  $S$  into  $st$ ;
7    $S \leftarrow$  pop out the top stable group from  $st$ ;
8   if IsDensest( $G[S]$ ) then
9     if VerifyLhCDS( $G, S, \bar{\phi}_h, \underline{\phi}_h$ ) then
10      output  $G[S]$ ;  $k \leftarrow k - 1$ ;
11     if  $st$  is empty then break;
12      $S \leftarrow$  pop out the top stable group from  $st$ ;
13    $G' \leftarrow G[S]$ ;

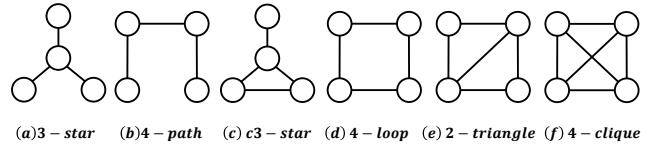
```

---

**Complexity Analysis.** We use  $T$  to denote the number of iterations that SEQ-kClist++ needs. Each iteration of SEQ-kClist++ costs  $O(n + |\Psi_h|)$ . We use  $N_{CL}$  to represent the total number of LhCDS candidates,  $N_{CL} \ll n$ . Each iteration of verifying an LhCDS candidate costs  $O(n + |\Psi_h|)$ .  $N_{Flow}$  is the number of times `IsDensest` and `VerifyLhCDS` are called. The time complexity of max-flow computation, which is  $O((n + |\Psi_h|)^2 \cdot (n + |\Psi_h| \cdot h))$  for `IsDensest` and `VerifyLhCDS` when Dinic Algorithm is Applied. The time complexity of IPPV is  $O((T + N_{CL}) \cdot (n + |\Psi_h|) + N_{Flow} \cdot (n + |\Psi_h|)^2 \cdot (n + |\Psi_h| \cdot h))$ . As  $(T + N_{CL}) \cdot (n + |\Psi_h|) \ll N_{Flow} \cdot (n + |\Psi_h|)^2 \cdot (n + |\Psi_h| \cdot h)$ , the time complexity of IPPV is  $O(N_{Flow} \cdot (n + |\Psi_h|)^2 \cdot (n + |\Psi_h| \cdot h))$ . The memory complexity is  $(n + |\Psi_h|)$ . **Note that the time complexity of the classical exact solution of only computing  $h$ -clique compact number is  $O((n \cdot |\Psi_h|)^3 \cdot L)$  [15], which is much greater than that of our algorithm.**

#### 5 LhCDS DISCOVERY

A pattern (also known as a motif) [17, 22, 40] is a small connected subgraph that appears frequently in a larger graph, which can be considered as a basic module. Figure 8 shows all kinds of patterns with four vertices: 4a-pattern, ..., 4f-pattern.



**Figure 8: An example of all patterns with four vertices**

We further show that the algorithm for the locally  $h$ -clique densest subgraph discovery problem can be extended to solve the locally general pattern densest subgraph discovery problem, which contributes to a deeper understanding of the organizational principles and functional modules within complex networks.

#### 5.1 Densest Supermodular Set Decomposition

In this section, we discuss the feasibility of extending the  $h$ -clique problem to a general pattern problem. The convex programming of

$h$ -clique can be further generalized to the convex programming of supermodular sets, so that the convex programming for the general pattern densest subgraph problem and the corresponding compact number can be derived. A function  $f : 2^V \rightarrow \mathbb{R}_+$  is said to be supermodular if  $\forall A, B \subseteq V, f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$ . Harb et al. [16] proposed the densest supermodular subset (DSS) problem: given a normalized, nonnegative monotone supermodular function  $f : 2^V \rightarrow \mathbb{R}_+$ , return  $S \subseteq V$  that maximizes  $\frac{f(S)}{|S|}$ . According to our observation, when  $f(S) = |E(S)|$  and  $f(S) = |\Psi_h(G[S])|$ , the DSS problem is the DS and CDS problem, respectively. When  $f(S)$  represents the number of a particular pattern in a graph, the problem is the densest problem of the proposed pattern. The convex program[16] for the densest supermodular set decomposition is  $CP(G) := \min \{\sum_{u \in V} r(u)^2\}$ , subject to:  $r \in \{x \in \mathbb{R}^V | x \geq 0, x(S) \geq f(S) \text{ for all } S \subseteq V, x(V) = f(V)\}$ .

With supermodularity, there is a property that each graph has a unique nested diminishingly decomposition for each type of density. The analysis of the generalization of CDS problem to DSS problem has triggered our thinking on the solution of locally general pattern densest problem.

## 5.2 Locally General Pattern Densest Subgraph Problem

Given an undirected graph  $G = (V, E)$ ,  $\psi_{hx}(V_{\psi_{hx}}, E_{\psi_{hx}})$  denotes a particular kind of pattern  $x$  with  $h$  vertices and  $\Psi_{hx}(G)$  is the collection of the  $hx$ -patterns of  $G$ .  $d_{\psi_{hx}}(G) = \frac{|\Psi_{hx}(G)|}{|V|}$  denotes the  $hx$ -pattern density of  $G$ .  $deg_G(v, \psi_{hx})$  is the  $hx$ -pattern degree of  $v$ , i.e., the number of  $hx$ -patterns containing  $v$ . A graph  $G = (V, E)$  is  $hx$ -pattern  $\rho$ -compact if  $G$  is connected, and removing any subset of vertices  $S \subseteq V$  will result in the removal of at least  $\rho \times |S|$   $hx$ -patterns in  $G$ . We can formally define a locally  $hx$ -pattern densest subgraph as follows.

**Definition 7 (Locally  $hx$ -pattern densest subgraph (LhxPDS)).** A subgraph  $G[S]$  of  $G$  is an LhxPDS of  $G$  if  $G[S]$  is  $hx$ -pattern  $d_{\psi_{hx}}(G)$ -compact, and there does not exist a supergraph  $G[S']$  of  $G[S]$  ( $S' \supsetneq S$ ), such that  $G[S']$  is also  $hx$ -pattern  $d_{\psi_{hx}}(G)$ -compact.

Similarly, we formulate the locally  $hx$ -pattern densest subgraph problem as follows.

**Definition 8 (Locally  $hx$ -pattern densest subgraph Problem (LhxPDS Problem)).** Given a graph  $G$ , an integer  $h$ , a pattern  $x$  and an integer  $k$ , the LhxPDS problem is to compute the top- $k$  LhxPDSes ranked by the  $hx$ -pattern density in  $G$ .

Here, we utilize our “iterative propose-prune-and-verify” pipeline to solve the LhxPDS problem. To apply the  $hx$ -pattern subgraph, there are some differences between Algorithm 6 and Algorithm 7 in the algorithmic details. In Algorithm 7, we need to **count**  $hx$ -pattern graphs for Seq-kClist++ algorithm and derive candidate LhxPDS algorithm. In pruning part, the computation of  $hx$ -pattern graph cores is different for diverse kinds of patterns. Unlike  $h$ -clique, there may be more than one  $hx$ -pattern on a graph with  $h$  vertices. In verification part, the methods for reducing the size of subgraph to compute the min-cut need small adjustments for different patterns. In general, the process of extending our algorithm to general patterns is concise and clear. **In addition, our method may also support**

### Algorithm 7: The IPPV algorithm for LhxPDS

**Input:**  $G = (V, E)$ , number of iterations  $T$ , an integer  $k$   
**Output:** top- $k$  LhxPDS

```

1  $st \leftarrow$  an empty stack;  $G' \leftarrow G$ ;
2  $\bar{\phi}_{hx}, \underline{\phi}_{hx} \leftarrow$  InitializeBd for  $hx$ -pattern  $(G', h, x)$ ;
3 while  $k > 0$  do
4    $S, \bar{\phi}_{hx}, \underline{\phi}_{hx} \leftarrow$  ProposeCL for  $hx$ -pattern  $(G', T, \bar{\phi}_{hx}, \underline{\phi}_{hx})$ ;
5    $S \leftarrow$  Prune for  $hx$ -pattern  $(G', S, \bar{\phi}_{hx}, \underline{\phi}_{hx})$ ;
6   foreach  $S \in S$  reversely do
7      $\mid$  push  $S$  into  $st$ ;
8    $S \leftarrow$  pop out the top stable group from  $st$ ;
9   if IsDensest  $(G[S])$  for  $hx$ -pattern then
10    if VerifyLhxPDS  $(G, S, \bar{\phi}_{hx}, \underline{\phi}_{hx})$  then
11       $\mid$  output  $G[S]$ ;  $k \leftarrow k - 1$ ;
12    if  $st$  is empty then
13       $\mid$  break;
14     $S \leftarrow$  pop out the top stable group from  $st$ ;
15  $G' \leftarrow G[S]$ ;
```

**expressive graph models as directed graph, attributed graph, etc. These graph models also have meaningful patterns, such as directed closed loop in directed graph. The difficulty of generalizing IPPV to different graph models is related to the properties of the models.**

## 6 EXPERIMENTS

### 6.1 Experimental Setup

The datasets we use are undirected real-world graphs [21, 31], including social networks, biological networks, web graphs, and collaboration networks. All datasets are listed in Table 2.

Table 2: Datasets used in our experiments

Name	Abbr.	$ V $	$ E $	$ \Psi_3 $	$ \Psi_5 $
soc-hamsterster	HA	2,426	16,630	53,251	298,013
CA-GrQc	GQ	5,242	14,484	48,260	2,215,500
fb-pages-politician	PP	5,908	41,706	174,632	2,002,250
fb-pages-company	PC	14,113	52,126	56,005	207,829
web-webbase-2001	WB	16,062	25,593	21,115	382,674
CA-CondMat	CM	23,133	93,439	173,361	511,088
soc-epinions	EP	26,588	100,120	159,700	521,106
Email-Enron	EN	36,692	183,831	727,044	5,809,356
loc-gowalla	GW	196,591	950,327	2,273,138	14,570,875
DBLP	DB	317,080	1,049,866	2,224,385	262,663,639
Amazon	AM	334,863	925,872	667,129	61,551
soc-youtube	YT	495,957	1,936,748	2,443,886	5,306,643
soc-lastfm	LF	1,191,805	4,519,330	3,946,207	10,404,656
soc-flixster	FX	2,523,386	7,918,801	7,897,122	96,315,278
soc-wiki-talk	WT	2,394,385	4,659,565	9,203,519	382,777,822

We compare the performances of the following algorithms:

**IPPV** : the top- $k$  LhCDS discovery algorithm proposed by us.

**LTDS [33]** : the top- $k$  LTDS discovery algorithm based on the maximum-flow, which solves the LhCDS problem with  $h = 3$ .

**LDSflow [30]** : the top- $k$  LDS discovery algorithm based on the maximum-flow, which solves the LhCDS problem with  $h = 2$ .

**Greedy** : the top- $k$  CDS discovery algorithm based on KClst++ [35] using greedy approach. It has no guarantee on the locally densest property.

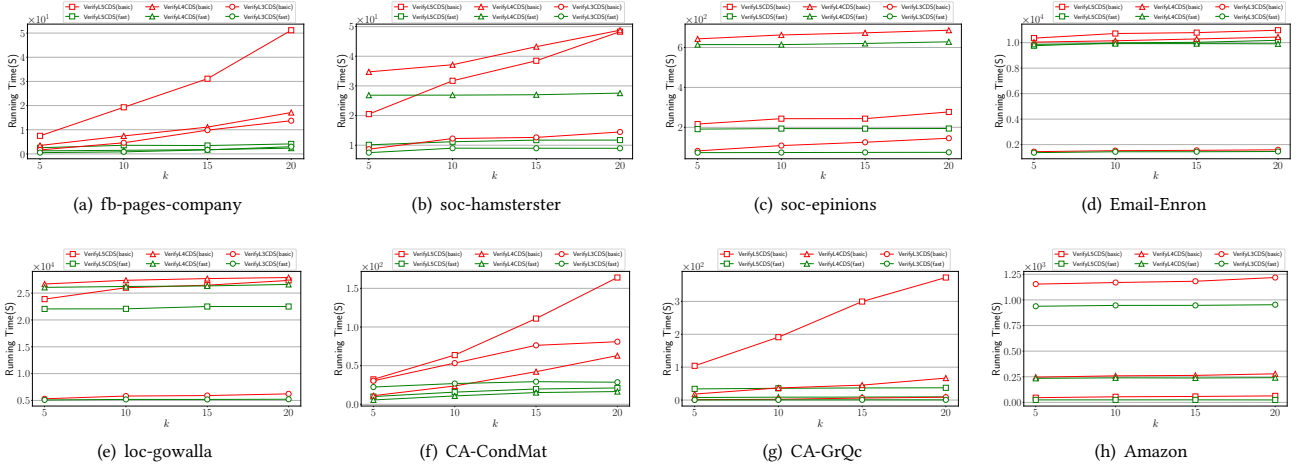


Figure 9: Running time of algorithms with different  $h$  ( $= 3, 4, 5$ ) and  $k$ . Red is VerifyLhCDS(basic), Green is VerifyLhCDS(fast)

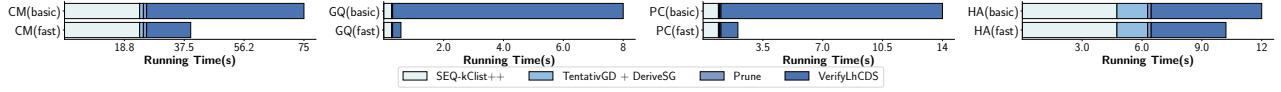


Figure 10: Running time of each part of IPPV with  $h = 3$  and  $k = 20$

All algorithms are implemented in C++ and compiled by g++ compiler at -O3 optimization level. All experiments are evaluated on a machine with Intel(R) Xeon(R) CPU 3.20GHz processor and 128GB memory, with Ubuntu operating system. Algorithms running for more than 48 hours are forcibly terminated.

## 6.2 Efficiency under Parameter Variations

In this section, we summarize the influence of different parameter changes on the running time.

**6.2.1 Efficiency improvement by fast verification algorithm.** We use VerifyLhCDS(basic) to represent the IPPV algorithm with Algorithm 4 and VerifyLhCDS(fast) to represent the IPPV algorithm with Algorithm 5. Their running times are compared in Figure 9. The fast verification algorithm with a smaller flow network is much faster than basic verification method. Especially as  $k$  increases, the efficiency gap between the two algorithms becomes more apparent. We also compare the running time of the two verification algorithms in the total running time in Figure 10. The acceleration effect of the fast algorithm is obvious. The results demonstrate the importance and benefit of optimizing the verification algorithm.

**6.2.2 Running time trends with varying  $k$ .** Parameter  $k$  has a more pronounced impact on the running time of the algorithm than  $h$ . The experiments in Figure 9 indicate a direct relationship, where an increase in  $k$  corresponds to a proportional increase in execution time. This trend is consistently observed across different datasets, which strengthens the premise that  $k$  is an important factor in computational complexity. The running time of both algorithms increases significantly for incremental values of  $k$ . The only deviation is observed in the Email-Enron dataset, where the running time remains relatively static despite changes in  $k$ , due to the fact that the total number of LhCDSes in this dataset is smaller than  $k$ .

**6.2.3 Running time trends with varying  $h$ .** We took  $h = 3, 4, 5$  to compare the impact of  $h$  on the running time. The results are shown in Figure 9. When  $h = 5$ , the running time is generally longer. The reason is that when  $h = 5$ , the number of 5-cliques is larger, as shown in Table 2. On Amazon, the running time is shorter when  $h = 5$ , because the number of 5-cliques is smaller. The running time is proportional to the number of  $h$ -cliques with different  $h$ .

**6.2.4 Influence of structural property.** We discuss how the density property ( $|E|/|V|$ ) of graph datasets influences the efficiency of IPPV. As shown in Figure 11, we select four datasets and randomly sample edges of these datasets in different proportions. According to the outputs from these synthetically generated graphs, the running time increases as the density of datasets increases. The reason is that when density is higher, the number of  $h$ -cliques increases, lengthening the total running time.

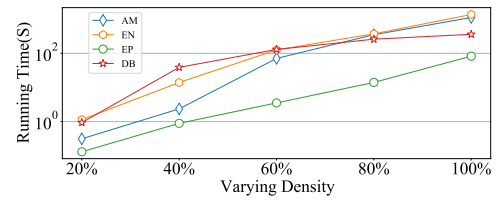


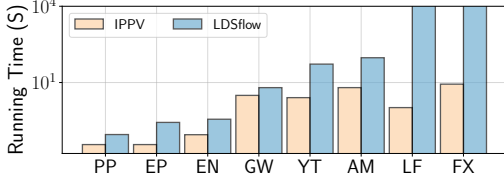
Figure 11: The running time on four datasets with varying density when  $h = 3$  and  $k = 5$

## 6.3 Efficiency v.s. Existing Algorithms

Since LDSflow [30] outputs LhCDSes with  $h = 2$ , and LTDS [33] outputs LhCDSes with  $h = 3$ , we compare IPPV with them by setting  $h = 2$  and  $h = 3$  respectively.



**6.3.1 Efficiency: IPPV v.s. LDSflow.** We set  $h = 2, k = 5$  to observe the running time of the two algorithms. The results are shown in Figure 12. We compare IPPV with LDSflow on eight datasets, and IPPV has efficiency improvements on all the datasets. The bottleneck of LDSflow is the loose upper and lower bounds.



**Figure 12: Efficiency of IPPV ( $h=2$ ) and LDSflow**

**6.3.2 Efficiency: IPPV v.s. LTDS.** We set  $k = 5$  and  $h = 3$  in experiments and the results are shown in Table 3. IPPV and LTDS are compared on all the datasets. There are significant efficiency improvements on all datasets. The main bottleneck of LTDS is the time-consuming verification part, and the reason is that the upper and lower bounds of LTDS are not as tight as that of IPPV, so there will be more failures in the verification part. The running time of our algorithm is closely related to the size of the graph and the number of  $h$ -cliques. A rise in the number of  $h$ -cliques can cause an increase in running time.

**Table 3: Efficiency of IPPV ( $h=3$ ) and LTDS**

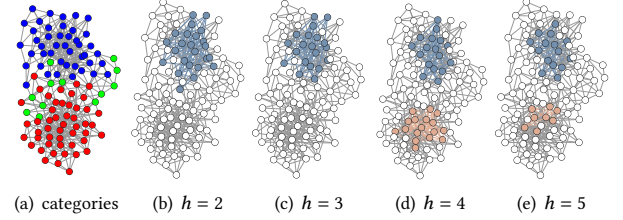
Dataset	IPPV ( $h=3$ )	LTDS	Speedup
soc-hamsterster	<b>7.50(s)</b>	46.54	6.20×
CA-GrQc	<b>0.38</b>	18.97	49.92×
fb-pages-politician	<b>32.32</b>	436.30	13.50×
fb-pages-company	<b>2.56</b>	51.48	20.11×
web-webbase-2001	<b>0.14</b>	12.20	87.14×
CA-CondMat	<b>21.63</b>	541.63	25.04×
soc-einions	<b>82.54</b>	558.91	6.77×
Email-Enron	<b>1369.84</b>	2253.14	1.64×
loc-gowalla	<b>5095.63</b>	68216.14	13.39×
DBLP	<b>360.49</b>	4888.93	13.56×
Amazon	<b>1118.08</b>	1308.53	1.17×
soc-youtube	<b>9070.89</b>	42821.99	4.72×
soc-lastfm	<b>11223.13</b>	$\geq 172,800$	$\geq 15.40 \times$
soc-flixster	<b>3018.62</b>	$\geq 172,800$	$\geq 57.24 \times$
soc-wiki-talk	<b>57382.42</b>	$\geq 172,800$	$\geq 3.011 \times$

## 6.4 Characteristics of the Detected LhCDSes

We adopt different quality measures to show the characteristics of the detected LhCDSes, and then we compare our detected LhCDSes with those detected by the Greedy algorithm.

**6.4.1 Visualization of LhCDSes with varying  $h$ .** We use a network of books about US politics which were sold by Amazon [19] to visualize the characteristics of detected LhCDSes with varying  $h$ . The vertices represent different books. Figure 13(a) shows the books fall into neutral(green), liberal(blue), and conservative(red) categories. The edges represent frequent co-purchasing of books by the same buyers, which indicate “customers who bought this book also bought the other books” on Amazon. Figure 13(b)-Figure 13(e) visualize the detected LhCDSes with varying  $h$ . The set of steel-blue vertices is the top-1 LhCDS, and the set of orange vertices if exists, is the top-2 LhCDS. The vertices in the dataset have clear attributes, which is convenient for us to measure whether IPPV has the ability to mine diverse communities. As visualized by the results, LhCDSes with larger  $h$  are closer to a clique. Besides, when  $h$

is larger, LhCDSes can find multiple dense communities in different categories. L4CDSes contain both liberal and conservative book communities, whereas LDSes (i.e., L2CDSes) only contain liberal book community. The results show the potential of IPPV to mine diverse categorical communities.



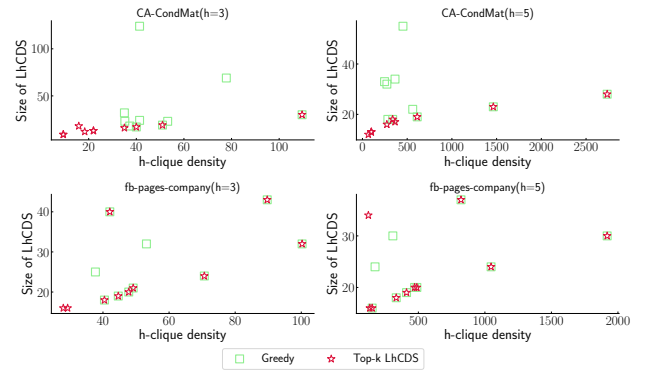
**Figure 13: LhCDS case study on real network (the top-1 LhCDS: steelblue; the top-2 LhCDS: orange vertices)**

**6.4.2 Edge density of LhCDSes with varying  $h$ .** We compare the average edge density  $((2 \times |E|) / (|V| \times (|V| - 1)))$  of top-5 LhCDSes for varying  $h$  in Table 4. When  $h$  is larger, LhCDSes generally have higher edge density. This is aligned with our general purpose.

**Table 4: Average edge density and diameter with varying  $h$**

dataset	Average Edge Density					Average Diameter				
	$h=2$	$h=3$	$h=5$	$h=7$	$h=9$	$h=2$	$h=3$	$h=5$	$h=7$	$h=9$
PC	0.752	0.858	0.891	0.894	<b>0.957</b>	2.00	2.00	2.00	2.00	2.00
HA	0.805	0.869	0.987	0.982	<b>0.995</b>	1.80	1.60	1.20	1.40	1.40
PP	<b>0.709</b>	0.685	0.696	0.729	<b>0.827</b>	2.20	2.00	2.00	2.00	2.00
CM	0.984	0.984	0.986	0.986	<b>0.986</b>	1.20	1.20	1.40	1.40	1.40
EP	0.487	0.724	<b>0.799</b>	0.770	0.755	2.60	1.80	1.80	1.67	2.00
WB	0.987	0.987	0.997	0.997	<b>0.997</b>	1.40	1.40	1.20	1.20	1.20
GQ	0.972	0.972	0.972	<b>0.975</b>	OOM	1.60	1.60	1.60	1.60	OOM

**6.4.3 Diameter of LhCDSes with varying  $h$ .** We list the average diameter (the longest distance of all pairs of nodes in a graph) of top-5 LhCDSes of different  $h$  in Table 4. It is obvious that the diameters of the subgraphs produced by LhCDS ( $h \geq 3$ ) do not exceed 2. Coupled with the edge density results, while LhCDS can find subgraphs with relatively higher edge density compared to LDS, the diameters of the subgraphs found by LhCDS are as small as those found by LDS. This suggests that LhCDSes are cohesive, and the nodes within LhCDSes are highly interconnected.



**Figure 14: Subgraph statistics of  $h$ -clique density and size**

**6.4.4 Size and  $h$ -clique density of subgraphs: IPPV vs Greedy.** Next, we compare the  $LhCDS$ es detected by IPPV and the  $h$ -clique densest subgraphs found by the Greedy algorithm. We select  $h = 3, 5$  on two datasets, and the results are shown in Figure 14. First, the results of the two algorithms overlap to a certain extent, among which the top-1  $CDS$  is the same because the first  $LhCDS$  must be the  $h$ -clique densest subgraph in the whole graph. Second, there is a certain difference between the returned subgraphs of the Greedy algorithm and IPPV. For example, in CA-CondMat ( $h=3$ ), the  $h$ -clique density of the second output subgraph is 51 for IPPV, and 78 for Greedy, but the returned subgraphs of Greedy is adjacent to the first output subgraph without the guarantee of the locally densest property. Therefore, the Greedy algorithm cannot solve the  $LhCDS$  problem well. The two algorithms totally overlap if and only if the top- $k$   $h$ -clique densest subgraph belongs to different regions occasionally.

## 6.5 Clustering Coefficient of Different $h$

Since near clique is an important criterion for evaluating dense subgraphs, we evaluate how  $LhCDS$ es of different  $h$  are close to the clique structure. In graph theory, *clustering coefficient* is a measure of the degree to which vertices in a graph tend to cluster together, which is a direct measure to the degree of near clique. For each vertex  $u \in V$ , which has  $k_u$  neighbors  $N_u$  ( $|N_u| = k_u$ ), the clustering coefficient of  $u$  is  $C_u = \frac{2|\{e_{vw}: v, w \in N_u, e_{vw} \in E\}|}{k_u(k_u-1)}$ . We compare the average  $C_u$  of all the  $LhCDS$ es of different  $h$  in Table 5.

Table 5: Average clustering coefficient of different  $h$  values

dataset	Average Clustering Coefficient				
	$h=2$	$h=3$	$h=5$	$h=7$	$h=9$
fb-pages-company	0.582	0.852	0.895	0.915	<b>0.930</b>
soc-hamsterster	0.480	0.910	0.990	0.984	<b>0.995</b>
fb-pages-politician	0.583	0.683	0.776	0.798	<b>0.835</b>
CA-CondMat	0.567	0.977	<b>0.992</b>	0.992	0.991
soc-epinions	0.231	0.722	0.701	0.705	<b>0.773</b>
web-webbase-2001	0.831	0.884	0.989	<b>0.992</b>	0.979
CA-GrQc	0.533	0.975	0.982	<b>0.985</b>	OOM

According to the results shown in Table 5, when  $h$  is larger, the average  $C_u$  is generally larger, showing the  $LhCDS$ es with larger  $h$  are closer to clique. In addition, there is a big difference between  $h = 3$  and  $h = 2$  ( $L2CDS$  is  $LDS$ ), which shows that  $LDS$  is less dense than other  $LhCDS$ . Our algorithm is important for finding near-clique subgraphs, which cannot be replaced by  $LDS$ .

## 6.6 Memory Overheads

We compare the memory utilization for the IPPV and LTDS algorithms across all datasets ( $h = 3, k = 5$ ). Figure 15 illustrates a clear correlation between memory usage and dataset size. IPPV strategically reduces the size of candidate subgraphs through a pruning mechanism prior to evaluating self-compactness. The verifying part often dominates the memory consumption.

## 6.7 The Number of Iterations

To choose the optimal number of iterations  $T$  of SEQ-kClist++, we set different  $T$  on the IPPV algorithm. We select  $T = 5, 10, 15, 20, 40, 60, 80, 100$ , as shown in Figure 16. The experiment on eight datasets

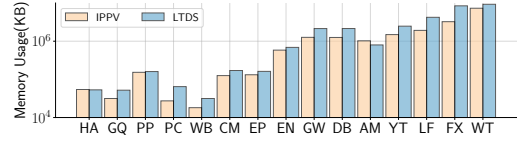


Figure 15: Memory usage of algorithms

shows that the optimal performance is between 15 and 20 iterations. In our experiments, we choose  $T = 20$ .

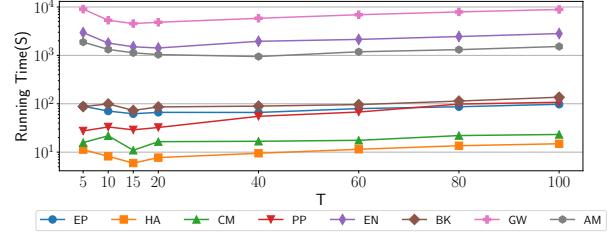


Figure 16: Running time of eight datasets with different  $T$

## 6.8 Case Study of $LhxPDS$

We utilize the same real dataset [19] to experimentally illustrate the  $LhxPDS$  problem. For each pattern depicted in Figure 8, we compute the results of  $L4xPDS$ . In Figure 17, the set of steelblue vertices is the top-1  $LhxPDS$ , and the set of orange vertices if exists, is the top-2  $LhxPDS$  of the pattern  $hx$ . It is evident that the  $L4xPDS$  corresponding to various patterns exhibit differences in terms of the number of  $L4xPDS$ , the number of vertices, and the position of vertices. To delve deeper into graph analysis, tasks such as community clustering can be extended to explore the  $LhxPDS$  subgraph.

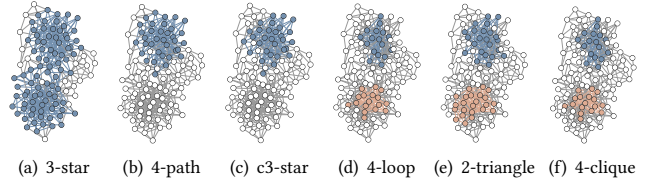


Figure 17:  $L4xPDS$  case study on real network (the top-1  $LhxPDS$ : steelblue; the top-2  $LhxPDS$ : orange vertices)

## 7 CONCLUSION

In this paper, we study how to discover locally  $h$ -clique densest subgraphs in a graph  $G$ , i.e., the  $LhCDS$  problem. We present IPPV, an iterative propose-prune-and-verify pipeline for top- $k$   $LhCDS$  detection. The  $h$ -clique compact number bounds and graph decomposition method which help to derive  $LhCDS$  candidates efficiently are proposed. A new optimized verification algorithm is designed, and its correctness is proved. The extension of our algorithm to solve the locally general pattern densest subgraph problem is feasible and promising. Extensive experiments on real datasets show the high efficiency and scalability of our proposed algorithm. When  $h$  is large in large-scale graphs, there is still room for further optimizing IPPV. We will continue to optimize the algorithm and further explore the  $LhxPDS$  problem in our future work.

## REFERENCES

- [1] Albert Angel, Nick Koudas, Nikos Sarkas, Divesh Srivastava, Michael Svendsen, and Srikanth Tirthapura. 2012. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *The VLDB Journal* 23 (2012), 175–199. <https://api.semanticscholar.org/CorpusID:2185310>
- [2] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest Subgraph in Streaming and MapReduce. *ArXiv abs/1201.6567* (2012).
- [3] Austin R. Benson, David F. Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353 (2016), 163 – 166. <https://api.semanticscholar.org/CorpusID:3635447>
- [4] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos E. Tsourakakis, Di Wang, and Junxing Wang. 2019. Flowless: Extracting Densest Subgraphs Without Flow Computations. *Proceedings of The Web Conference 2020* (2019).
- [5] Stephen P. Boyd and Lieven Vandenbergh. 2010. Convex Optimization. *IEEE Trans. Automat. Control* 51 (2010), 1859–1859. <https://api.semanticscholar.org/CorpusID:37925315>
- [6] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*.
- [7] Chandra Chekuri, Kent Quanrud, and Manuel R. Torres. 2022. Densest Subgraph: Supermodularity, Iterative Peeling, and Flow. In *ACM-SIAM Symposium on Discrete Algorithms*.
- [8] Jie Chen and Yousef Saad. 2012. Dense Subgraph Extraction with Application to Community Detection. *IEEE Transactions on Knowledge and Data Engineering* 24 (2012), 1216–1230. <https://api.semanticscholar.org/CorpusID:11360561>
- [9] Maximilien Danisch, T-H. Hubert Chan, and Mauro Sozio. 2017. Large Scale Density-friendly Graph Decomposition via Convex Programming. *Proceedings of the 26th International Conference on World Wide Web* (2017).
- [10] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V. S. Lakshmanan, and Xuemin Lin. 2019. Efficient Algorithms for Densest Subgraph Discovery. *Proc. VLDB Endow.* 12 (2019), 1719–1732.
- [11] Adriano Fazzone, Tommaso Lanciano, Riccardo Denni, Charalampos E. Tsourakakis, and Francesco Bonchi. 2022. Discovering polarization niches via dense subgraphs with attractors and repulsers. *Proceedings of the VLDB Endowment* 15, 13 (2022), 3883–3896.
- [12] David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering Large Dense Subgraphs in Massive Graphs. In *Very Large Data Bases Conference*. <https://api.semanticscholar.org/CorpusID:120822>
- [13] A. Gionis, Flavio Paiva Junqueira, Vincent Leroy, Marco Serafini, and Ingmar Weber. 2013. Piggybacking on Social Networks. *Proc. VLDB Endow.* 6 (2013), 409–420. <https://api.semanticscholar.org/CorpusID:2240241>
- [14] Andrew V. Goldberg. 1984. Finding a Maximum Density Subgraph. In *Technical report, University of California, Berkeley*.
- [15] Donald Goldfarb and Shucheng Liu. 1991. An  $O(n^3L)$  primal interior point algorithm for convex quadratic programming. *Mathematical Programming* 49 (1991), 325–340. <https://api.semanticscholar.org/CorpusID:29420601>
- [16] Elfarouk Harb, Kent Quanrud, and Chandra Chekuri. 2022. Faster and Scalable Algorithms for Densest Subgraph and Decomposition. In *Neural Information Processing Systems*.
- [17] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian Yee Hong Lam. 2019. Discovering Maximal Motif Cliques in Large Heterogeneous Information Networks. *2019 IEEE 35th International Conference on Data Engineering (ICDE)* (2019), 746–757. <https://api.semanticscholar.org/CorpusID:174819659>
- [18] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhr. 2009. 3-HOP: a high-compression indexing scheme for reachability query. *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data* (2009). <https://api.semanticscholar.org/CorpusID:7625491>
- [19] Valdis Krebs. 2004. Books about US politics. Unpublished. <http://www.orgnet.com/>
- [20] Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. 2023. A survey on the densest subgraph problem and its variants. *arXiv preprint arXiv:2303.14467* (2023).
- [21] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. <http://snap.stanford.edu/data>.
- [22] Jure Leskovec, Ajit Singh, and Jon M. Kleinberg. 2006. Patterns of Influence in a Recommendation Network. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. <https://api.semanticscholar.org/CorpusID:332896>
- [23] Ruiming Li, Jung-Yu Lee, Jinn-Moon Yang, and Tatsuya Akutsu. 2022. Densest subgraph-based methods for protein-protein interaction hot spot prediction. *BMC Bioinformatics* 23 (2022).
- [24] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. *ACM computing surveys (CSUR)* 51, 3 (2018), 1–34.
- [25] Wensheng Luo, Chenhao Ma, Yixiang Fang, and Laks VS Lakshman. 2023. A Survey of Densest Subgraph Discovery on Large Graphs. *arXiv preprint arXiv:2306.07927* (2023).
- [26] Chenhao Ma, Reynold Cheng, Laks VS Lakshmanan, and Xiaolin Han. 2022. Finding locally densest subgraphs: a convex programming approach. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2719–2732.
- [27] Michael Mitzenmacher, Jakub W. Pachocki, Richard Peng, Charalampos E. Tsourakakis, and Shen Chen Xu. 2015. Scalable Large Near-Clique Detection in Large-Scale Networks via Sampling. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015).
- [28] Gergely Palla, Imre Derényi, Illés J. Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435 (2005), 814–818. <https://api.semanticscholar.org/CorpusID:3250746>
- [29] Jean-Claude Picard and Maurice Queyranne. 1982. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks* 12 (1982), 141–159.
- [30] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally densest subgraph discovery. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 965–974.
- [31] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. <https://networkrepository.com>
- [32] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. 2010. Dense Subgraphs with Restrictions and Applications to Gene Annotation Graphs. In *Annual International Conference on Research in Computational Molecular Biology*. <https://api.semanticscholar.org/CorpusID:11280177>
- [33] Raman Samusevich, Maximilien Danisch, and Mauro Sozio. 2016. Local triangle-densest subgraphs. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 33–40.
- [34] Victor Spirin and Leonid A. Mirny. 2003. Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences of the United States of America* 100 (2003), 12123 – 12128. <https://api.semanticscholar.org/CorpusID:136093>
- [35] Binta Sun, Maximilien Danisch, TH Hubert Chan, and Mauro Sozio. 2020. Kclist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. *Proceedings of the VLDB Endowment (PVLDB)* (2020).
- [36] Tran Ba Trung, Lijun Chang, Nguyen Tien Long, Kai Yao, and Huynh Thi Thanh Binh. 2023. Verification-Free Approaches to Efficient Locally Densest Subgraph Discovery. *2023 IEEE 39th International Conference on Data Engineering (ICDE)* (2023), 1–13. <https://api.semanticscholar.org/CorpusID:260171546>
- [37] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*, 1122–1132.
- [38] Charalampos E. Tsourakakis, Francesco Bonchi, A. Gionis, Francesco Gullo, and Maria A. Tsiarli. 2013. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (2013). <https://api.semanticscholar.org/CorpusID:213308>
- [39] Jia Wang, James Cheng, and Ada Wai-Chee Fu. 2013. Redundancy-aware maximal cliques. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (2013). <https://api.semanticscholar.org/CorpusID:16494932>
- [40] Stefan Wuchty, Zoltán N. Oltvai, and A L Barabasi. 2003. Evolutionary conservation of motif constituents in the yeast protein interaction network. *Nature Genetics* 35 (2003), 176–179. <https://api.semanticscholar.org/CorpusID:1627007>
- [41] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and W. Zhang. 2015. Diversified top-k clique search. *The VLDB Journal* 25 (2015), 171–196. <https://api.semanticscholar.org/CorpusID:15668109>
- [42] Feng Zhao and Anthony Kum Hoe Tung. 2012. Large Scale Cohesive Subgraphs Discovery for Social Network Visual Analysis. *Proc. VLDB Endow.* 6 (2012), 85–96. <https://api.semanticscholar.org/CorpusID:12588941>
- [43] Zhaonian Zou. 2013. Polynomial-time algorithm for finding densest subgraphs in uncertain graphs. In *Proceedings of MLG Workshop*.