Efficient Algorithms for Densest Subgraph Discovery

Yixiang Fang[♡†⋆], Kaiqiang Yu[‡], Reynold Cheng[‡], Laks V.S. Lakshmanan[§], Xuemin Lin[†⋆]
[♡]Guangzhou University, China, [†]The University of New South Wales, Australia, [⋆]Zhejiang Lab, China,
[‡]The University of Hong Kong, China, [§]The University of British Columbia, Canada

[†]{yixiang.fang@,lxue@cse.}unsw.edu.au,[‡]{ky, ckcheng}@cs.hku.hku,[§]laks@cs.ubc.ca

ABSTRACT

Densest subgraph discovery (DSD) is a fundamental problem in graph mining. It has been studied for decades, and is widely used in various areas, including network science, biological analysis, and graph databases. Given a graph G, DSD aims to find a subgraph D of G with the highest density (e.g., the number of edges over the number of vertices in D). Because DSD is difficult to solve, we propose a new solution paradigm in this paper. Our main observation is that the densest subgraph can be accurately found through a k-core (a kind of dense subgraph of G), with theoretical guarantees. Based on this intuition, we develop efficient exact and approximation solutions for DSD. Moreover, our solutions are able to find the densest subgraphs for a wide range of graph density definitions, including clique-based- and general pattern-based density. We have performed extensive experimental evaluation on both real and synthetic datasets. Our results show that our algorithms are up to four orders of magnitude faster than existing approaches.

1. INTRODUCTION

Given a graph G with n vertices and m edges, the *densest subgraph discovery* (DSD) is the problem of discovering a "dense" subgraph from G [11, 66, 28, 14]. For example, the densest subgraph of Figure 1(a) is S_1, because its *edge-density*, or the average number of edges over the number of vertices in S_1, is the highest among all possible subgraphs of G. The DSD problem is fundamental to graph mining [31], and is widely used in network science, biological analysis, graph databases, and system optimization. In network science, for instance, the densest subgroups discovered can be used to find "cohesive groups" in social networks, for purposes of community detection [11, 66]. In biology, as another example, bioinformatics researchers have studied the use of DSD in identifying regulatory motifs in genomic DNA [28] and gene annotation graphs [55]. In graph databases, the DSD is a building block for many graph algorithms, such as creating elegant index structures for reachability and distance queries [14, 41] and supporting graph visualization [71, 72]. In system optimization, DSD has been used
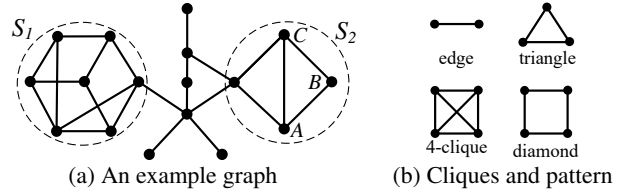
(a) An example graph   (b) Cliques and pattern

Figure 1: Illustrating the densest subgraphs.

in social piggybacking [30, 31], which can be used to improve the throughput of social networking systems (e.g., Facebook).

At present, two variants of DSD have been proposed. The first problem is to find the subgraph with the highest edge-density in G. In Figure 1(a), for example, S_1 has the highest edge-density of 11/7 among all possible subgraphs of G. Recently, researchers have studied DSD by defining density based on h-clique, which is a complete graph of h vertices, with $h \geq 2$. Figure 1(b) shows a 3-clique (or "triangle") and a 4-clique. The goal of DSD is then to find the subgraph of G that has the highest h-*clique-density* [65, 49], or the average number of h-cliques that a vertex participates in. In Figure 1(a), subgraph S_2 has the highest "3-clique-density", in terms of number of triangles. The DSD problem, based on clique-density, can be used for detecting larger near-cliques [65, 49] (which can be used for communication network analysis and automatic test pattern generation [1]). The triangle-based densest subgraphs are useful for finding research groups in the DBLP network and clusters in senators' network on US bill voting [65], and discovering compact dense subgraphs from networks [57]. Note that an edge is a 2-clique, so edge-density is the 2-clique-density.

Our main goal is to solve the DSD problem with respect to edge- and clique- densities. This problem is technically challenging [32, 65, 10, 72]. Existing DSD solutions, which often involve solving the maximum flow problem, are computationally expensive. For example, given a graph G with n vertices and m edges, a well-known algorithm based on edge-density [32] may incur a time complexity of $\mathcal{O}((mn + m^3) \log n)$, and is thus impractical for very large graphs. The h-clique-based DSD problem is even more complex [65, 49]. Moreover, our experiments show that existing DSD solutions cannot handle large graphs very well, and there is considerable room for developing faster solutions.

In this paper, our goal is to develop efficient algorithms for finding the subgraph with the highest edge- and h-clique-density. We leverage the k-core [62], or the largest subgraph of graph G, where each vertex has at least k neighbors. We show that the densest subgraph (in terms of edge-density) is located in some k-cores, which are often much smaller than the entire graph G. For example, in Figure 1(a), the subgraph S_1 is the 3-core, which is also the densest subgraph of G, w.r.t. edge-density. To solve DSD w.r.t. h-clique-density, we extend the k-core to the k-clique-core, or (k, Ψ)-core, which incorporates an h-clique Ψ into the k-core definition. Based

on the cores, we develop efficient exact and approximation algorithms for finding the subgraphs with the highest edge-density and h-clique-density. Notably, this "core-based solution" achieves the same approximation ratio as the current state-of-the-art.

It is non-trivial to use (k, Ψ)-core to solve the DSD problem. Here we give an outline of this process. We denote by k the *core number*. We first derive the lower and upper bounds on the h-clique-density for each (k, Ψ)-core. Based on these tight bounds, we can compute the upper and lower bounds of ρ_{opt}, which is the density of the densest subgraph, and further locate the densest subgraph w.r.t. an h-clique in some specific (k, Ψ)-cores. These (k, Ψ)-cores are often much smaller than the entire graph G, and thus we can directly compute the densest subgraph from these small cores, resulting in high efficiency.

Specifically, to compute the exact densest subgraph D, we first locate D in a specific (k, Ψ)-core. Then, we build a flow network on this core, and find D by solving the maximum flow problem using binary search. During the binary search, whenever we obtain a larger lower bound of ρ_{opt}, we can further locate D in another core with higher core number and build an even smaller flow network to compute D. The binary search process stops when we have found D. We further show that the $(k_{\max}, \Psi)$-core, which is a (k, Ψ)-core with k attaining the maximum value, is a good approximation to the densest subgraph, with theoretical guarantees. To find the $(k_{\max}, \Psi)$-core, a straightforward method is to perform core decomposition, which computes all the (k, Ψ)-cores in an incremental manner. This is costly and unnecessary because we only need the $(k_{\max}, \Psi)$-core, rather than all the (k, Ψ)-cores. We thus develop another efficient method that extracts the $(k_{\max}, \Psi)$-core without computing all the (k, Ψ)-cores. This solution finds the $(k_{\max}, \Psi)$-core from a set of small subgraphs induced by vertices with high degrees, and thus yields better performance.

In addition, we generalize the notion of density to allow arbitrary "pattern graphs" (e.g., the diamond pattern in Figure 1(b)), and propose *pattern-density* to measure the average number of *patterns* in which a vertex participates. We further extend k-clique-core to k-pattern-core, and show that our solutions above can be smoothly adapted to finding the densest subgraph w.r.t. pattern-density.

We have performed extensive experiments to evaluate our approaches. On both real and synthetic graph datasets ranging from a few thousand to millions of vertices and edges, our new solutions show high efficiency. For example, our core-based exact algorithm, namely CoreExact, is up to four orders of magnitude faster than the state-of-the-art exact DSD solution. Our best approximation algorithm, called CoreApp, is up to two orders of magnitude faster than the existing approximation solution. We further perform experiments to find pattern-based densest subgraphs and our results again confirm the superiority of our core-based approaches.

Contributions. In summary, our main contributions are:

- We present a new perspective on solving the DSD problem. Particularly, we propose the (k, Ψ)-core by incorporating an h-clique Ψ where $h \geq 2$ (Section 5). We further establish the lower and upper bounds of densities for (k, Ψ)-cores.

- Based on the (k, Ψ)-cores, we develop fast exact and approximation DSD algorithms w.r.t. h-clique-density (Section 6).

- We generalize h-clique-density to pattern-density and adapt our solutions to solving DSD w.r.t. pattern-density (Section 7).

- We conduct extensive experiments on ten real datasets and three synthetic datasets to evaluate our algorithms. The results reveal that our proposed DSD algorithms are several orders of magnitude faster than existing ones (Section 8).

Organization. We review the related work in Section 2. The DSD problem is stated in Section 3. In Sections 4-6 we present different DSD solutions. In Section 7, we extend our algorithms

for finding densest subgraphs for general patterns. We report experimental results in Section 8, and conclude in Section 9. *Due to space limitation, for some lemmas, we do not show the complete proof in this paper; instead, we give the proof sketch and show the complete proof in the technical report [27].*

2. RELATED WORK

The problem of dense subgraph computation has been extensively studied [31, 56, 11, 66]. In the following, we review existing works that are highly related to our DSD problem.

Edge-based Densest Subgraph (EDS). The edge-density of an undirected graph $G(V, E)$ is defined as $\frac{m}{n}$ with $n=|V|$ and $m=|E|$. The EDS problem aims to find a subgraph such that its edge-density is the highest among all subgraphs. This problem can be addressed by solving a parametric maximum-flow problem [32, 29]. A typical variant of EDS is to impose a size restriction on the returned subgraph, i.e., finding a subgraph of up to a given number of vertices whose density is the highest. This problem is NP-hard [5, 4]. Another version of EDS, called optimal quasi-clique [66], extracts a subgraph, which is more compact, with a smaller diameter than the EDS. Again, this variant is NP-hard [9]. Qin et al. developed solutions for finding the top-k locally densest subgraphs [54]. The EDS problem on evolving graphs is studied in [19]. In [64, 18], the edge-density-based graph decomposition is extensively studied. Kannan and Vinay [43] modeled the density on directed graphs, and then studied the DSD problem on directed graphs [10].

In general, exact EDS solutions work well for small graphs, but they perform poorly for large graphs. Thus, researchers have developed approximation algorithms, in order to achieve higher efficiency. In [10], Charikar et al. proposed a greedy 0.5-approximation algorithm for solving the EDS problem. Bahmani et al. [6] devised a $1/(2 + 2\varepsilon)$-approximation algorithm under the streaming model, which takes $\mathcal{O}(m \frac{\log(n)}{\varepsilon})$ time. The densest subgraph on directed graphs can also be computed by an approximation algorithm [44].

Our solution is based on computing k-cores, which can then be used to find the EDS. Based on this intuition, we have developed exact and approximation algorithms, and show using extensive experiments that they are much faster than existing EDS solutions.

h**-clique Densest Subgraph (CDS).** In [65, 49], Tsourakakis et al. modeled graph density based on h-cliques, and studied the h-clique densest subgraph (CDS) problem. It generalizes the EDS problem, which is a special case of CDS for h=2. They found that the 3-clique densest subgraphs (a 3-clique is a triangle) help identify cohesive researcher groups in a bibliographical network, as well as clusters of republicans in the network of US senators. Recently, a variant based on the 3-clique, called top-k local triangle-densest subgraphs discovery, has been investigated [57].

There are four key differences between existing works [65, 49] and our work. (1) Our algorithms, based on (k, Ψ)-cores where Ψ is an h-clique, are substantially different from existing CDS solutions. (2) Whereas [65, 49] can only handle h-cliques, our work supports any general pattern (e.g., 4-vertex subgraph [40, 68]). (3) The approximation algorithm in [49] is a randomized algorithm which has a failure probability to obtain an approximation solution, while our core-based approximation algorithms are deterministic algorithms. (4) Our empirical evaluation shows that our algorithms significantly outperform previous exact algorithms [65, 49] and deterministic approximation algorithm [65].

Other Dense Subgraphs. Recently, many other dense subgraph models [24], such as k-core [7, 47, 51, 23, 22, 20, 25, 26, 67, 12], k-truss [15, 37, 69, 39, 38], k-(r, s) nucleus [60, 58, 61, 59] (a generalization of k-core and k-truss), k-clique [16, 34], k-edge connected components [35, 36], and k-plexes [63], have also been explored. However, these dense subgraphs are different from EDS and CDS, which attain the highest edge-density and clique-density.

3. PROBLEM DEFINITION

Data model. In this paper, we consider an undirected, unweighted, and simple graph $G(V, E)$ with vertex set V and edge set E, where $n=|V|$ and $m=|E|$. The degree of a vertex v in G, denoted by $deg_G(v)$, is the number of its neighbors, and we denote the maximum degree by d. Table 1 summarizes all the notations frequently used in this paper. Next, we first introduce two prominent notions of density that were employed in the DSD literature, namely edge-density and h-clique-density.

DEFINITION 1 (EDGE-DENSITY [32, 29]). *Given a graph G (V, E), its edge-density is $\tau(G)= \frac{|E|}{|V|}$.*

DEFINITION 2 (CLIQUE INSTANCE). *Given a graph $G(V, E)$ and an integer $h \geq 2$, we say a set of h vertices, $S \in V$, is an h-clique instance, if each pair of vertices $u, v \in S$ is connected by an edge.*

DEFINITION 3 (CLIQUE-DEGREE). *Given a graph $G(V, E)$ and an h-clique Ψ, the clique-degree of a vertex v in G, or $deg_G(v, \Psi)$, is the number of clique instances containing v.*

Note that for each of these instances, we do not consider permutations of vertices. For example, let Ψ be the triangle (i.e., 3-clique). Then in Figure 1(a), the subgraph S_2 contains two clique instances of Ψ, which share an edge. The clique-degrees of vertices A, B, and C are 2, 1, and 2 respectively.

DEFINITION 4 (h-CLIQUE-DENSITY [65]). *Given a graph G (V, E) and an h-clique $\Psi(V_\Psi, E_\Psi)$ with $h \geq 2$, the h-clique-density of G w.r.t. Ψ is*

$$\rho(G, \Psi) = \frac{\mu(G, \Psi)}{|V|}, \qquad (1)$$

where $\mu(G, \Psi)$ is the number of clique instances of Ψ in G.

The densest subgraph of G w.r.t. edge-density (resp., h-clique-density), i.e., *EDS* [32] (resp., *CDS* [65, 49]), is the subgraph $D=(V_D, E_D)$ of G whose edge-density (resp., h-clique-density) is the highest. Clearly, if the h-clique is a single edge (i.e., $h=2$), the h-clique-density reduces to edge-density. For ease of exposition, in the following we simply focus on the h-clique-density with $h \geq 2$. We use the term CDS when we refer to the DSD problem using the edge-, or h-clique-based density. Where necessary, we make the distinction between EDS and CDS.

Now we formally introduce the problem studied in this paper.

PROBLEM 1 (CDS PROBLEM [65, 49]). *Given a graph $G(V ,E)$ and an h-clique $\Psi(V_\Psi, E_\Psi)$ ($h \geq 2$), return the subgraph D of $G(V, E)$, whose h-clique-density $\rho(D, \Psi)$ is the highest.*

We denote the h-clique-density of D by ρ_{opt}, i.e., $\rho_{opt}=\rho(D, \Psi)$, where D is the CDS. For the graph G of Figure 1(a), if we let Ψ be the single edge, we will return S_1 as the densest subgraph; if we let Ψ be the 3-clique (i.e., triangle), then, S_2 is the subgraph with the highest 3-clique-density.

4. EXISTING APPROACHES

In this section, we review existing algorithms for the EDS and CDS problems, and then discuss their limitations.

4.1 The Exact Method

Generally, the algorithms for finding exact EDS and CDS [32, 65, 49] follow the same framework by solving a maximum flow problem using binary search. A flow network [33] is a directed graph $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$, where there is a source node[1] s, a sink node

[1]We use "node" to mean "flow network node" in this paper.

Table 1: Notations and meanings.

Notation	Meaning
$G(V, E)$	a graph with vertex set V and edge set E
n, m	$n=|V|$, $m=|E|$
$deg_G(v)$	(classical edge-based) degree of vertex v in G
d	the maximum (classical edge-based) degree of G
$G[T]$	a subgraph of G induced by vertex set T
$\Psi(V_\Psi, E_\Psi)$	an h-clique (vertex set: V_Ψ, edge set E_Ψ)
$deg_G(v, \Psi)$	clique-degree of vertex v in G w.r.t. Ψ
$\mu(S, \Psi)$	number of clique instances of Ψ in the graph S
$\rho(G, \Psi)$	h-clique-density of graph G w.r.t. an h-clique Ψ
$D(V_D, E_D)$	the CDS whose h-clique-density is ρ_{opt}
$\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$	a flow network with node set $V_\mathcal{F}$ and edge set $E_\mathcal{F}$

t, and some intermediate nodes; each edge has a capacity and the amount of flow on an edge cannot exceed the capacity of the edge. The maximum flow of a flow network equals the capacity of its minimum st-cut, $(\mathcal{S}, \mathcal{T})$, which partitions the node set $V_\mathcal{F}$ into two disjoint sets, $\mathcal{S}$ and $\mathcal{T}$, such that $s \in \mathcal{S}$ and $t \in \mathcal{T}$.

Algorithm 1: The algorithm: `Exact`.

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$;
Output: The CDS $D(V_D, E_D)$;

1 initialize $l \leftarrow 0$, $u \leftarrow \max\limits_{v \in V} deg_G(v, \Psi)$;

2 initialize $\Lambda \leftarrow$ all the instances of $(h–1)$-clique in G, $D \leftarrow \emptyset$;

3 **while** $u - l \geq \frac{1}{n(n-1)}$ **do**

4      $\alpha \leftarrow \frac{l+u}{2}$;

5      $V_\mathcal{F} \leftarrow \{s\} \cup V \cup \Lambda \cup \{t\}$;   // build a flow network

6   **for** *each vertex $v \in V$* **do**

7   add an edge $s \rightarrow v$ with capacity $deg_G(v, \Psi)$;

8   add an edge $v \rightarrow t$ with capacity $\alpha|V_\Psi|$;

9   **for** *each $(h–1)$-clique $\psi \in \Lambda$* **do**

10   **for** *each vertex $v \in \psi$* **do**

11   add an edge $\psi \rightarrow v$ with capacity $+\infty$;

12   **for** *each $(h–1)$-clique $\psi \in \Lambda$* **do**

13   **for** *each vertex $v \in V$* **do**

14   **if** *ψ and v form an h-clique* **then**

15   add an edge $v \rightarrow \psi$ with capacity 1;

16   find minimum st-cut $(\mathcal{S}, \mathcal{T})$ from the flow network $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$;

17   **if** $\mathcal{S}=\{s\}$ **then** $u \leftarrow \alpha$;

18   **else** $l \leftarrow \alpha$, $D \leftarrow$ the subgraph induced by $\mathcal{S}\backslash\{s\}$;

19 **return** D;

We present the state-of-the-art algorithm from [49] in Algorithm 1, where the input is a graph G and an h-clique Ψ. First, it initializes lower and upper bounds of ρ_{opt} and collects all the instances of $(h–1)$-clique (lines 1-2). Then, it finds D by using binary search (lines 3-18). Specifically, in each binary search (lines 4-18), it tries to find a subgraph with density larger than a guessed value α, by computing the minimum st-cut using Gusfield's algorithm [2] in a flow network $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$. To build $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$, it first creates a node set $V_\mathcal{F}$ (line 5), and then links its nodes by directed edges with different capacities (lines 6-15). The binary search stops when the gap between the upper and lower bounds of α is less than $\frac{1}{n(n-1)}$. We denote this algorithm by `Exact`.

Note that if Ψ is the single edge, the flow network $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$ can be simplified such that [32]: $V_\mathcal{F}=\{s\} \cup V \cup \{t\}$, and for each vertex $v \in G$, there is a directed edge from s to v with capacity m and a directed edge from v to t with capacity $m+2\alpha-deg_G(v)$; for each edge $(v, u) \in G$, there is a directed edge from u to v with capacity 1 and a directed edge from v to u with capacity 1.
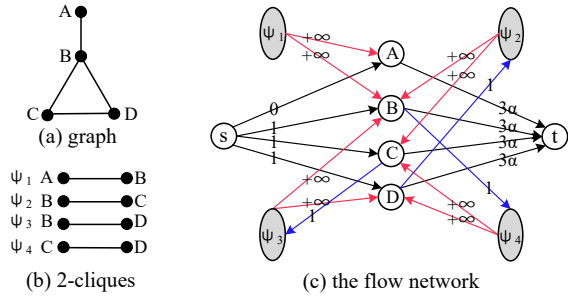
Figure 2: Illustrating the flow network (Ψ is a triangle).

EXAMPLE 1. Let Ψ be the triangle and G be the graph in Figure 2(a). The graph contains 4 edges (see Figure 2(b)). By Algorithm 1, we construct the flow network, as depicted in Figure 2(c), where the value on each edge denotes its capacity. □

LEMMA 1. *Given a graph $G(V,E)$ and an h-clique $\Psi(V_\Psi, E_\Psi)$, Exact takes $\mathcal{O}\left(n \cdot \binom{d-1}{h-1} + (n|\Lambda| + \min(n, |\Lambda|)^3) \log n\right)$ time and $\mathcal{O}(n + |\Lambda|)$ space, where Λ is set of $(h-1)$-clique instances in G [65].*

PROOF SKETCH: In the worst case, we will consider $n \cdot \binom{d-1}{h-1}$ h-clique instances, and each binary search of Exact takes $\mathcal{O}(n \cdot |\Lambda| + \min(n, |\Lambda|)^3)$ time. These are the main cost of Exact. □

In practice, h is often small and the number of clique instances, $|\Lambda|$, is often much larger than the number n of vertices, so the second summand dominates the overall computational cost.

4.2 The Approximation Method

The approximation method of computing the EDS [10] and CDS [65] follows the *peeling paradigm* and achieves an approximation ratio of $\frac{1}{|V_\Psi|}$. Here, the approximation ratio is the ratio of the h-clique-density of subgraph returned, over ρ_{opt}, which is at most 1.0. Specifically, given a graph G of n vertices, it works in n rounds. In each round, it removes the vertex that participates in the minimum number of h-cliques, and recomputes the density of the residual graph. Finally, the subgraph of the largest h-clique-density is returned. Algorithm 2 outlines the steps. Because the algorithm removes vertices one by one, we call it PeelApp.

Algorithm 2: The algorithm: PeelApp.

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$;
Output: A subgraph S^*;
1 initialize $S \leftarrow G$, $S^* \leftarrow \emptyset$;
2 compute the clique-degree for each vertex of G;
3 **while** $S \neq \emptyset$ **do**
4     $v \leftarrow$ the vertex with the minimum clique-degree in S;
5     $S \leftarrow$ remove the vertex v from S;
6   **if** $\rho(S, \Psi) > \rho(S^*, \Psi)$ **then** $S^* \leftarrow S$;
7 **return** S^*;

LEMMA 2. *Given a graph G and an h-clique $\Psi(V_\Psi, E_\Psi)$, then PeelApp takes $\mathcal{O}\left(n \cdot \binom{d-1}{h-1}\right)$ time and $\mathcal{O}(m)$ space [65].*

PROOF SKETCH: The main time cost comes from enumerating clique instances, whose number is $n \cdot \binom{d-1}{h-1}$ in the worst case. □

4.3 Limitations of Existing Algorithms

From the above lemmas, we see that while PeelApp is faster than Exact, it also sacrifices some accuracy. For example, when Ψ is an edge, Exact finds the exact EDS in $\mathcal{O}((mn + m^3) \log n)$ time, while PeelApp returns a subgraph with 0.5-approximation ratio in linear time, i.e., $\mathcal{O}(m)$. Both solutions can be inefficient on

larger graphs with more complex cliques. We found that Exact suffers from several problems: (1) the initial lower and upper bounds of α are not very tight; (2) the size of the flow network can be large when the graph is large and there are many clique instances of Ψ; and (3) the flow network $\mathcal{F}$ is always built on the entire graph G in each iteration, while the CDS is often in a small subgraph of G. The PeelApp algorithm also involves a lot of unnecessary computation: for the first few iterations, the graph contains many vertices with lower clique-degrees, which are unlikely to be in the CDS, but PeelApp still computes the h-clique-density. As shown in our experiments later, on a moderate-size graph ($n \approx 26K$ and $m \approx 100K$), Exact takes more than 5 days to find the densest subgraphs for 6-clique; on a million-scale graph ($n \approx 19M$ and $m \approx 298M$), PeelApp takes more than 2 days to find the CDS for 6-clique. Thus, there is room for improving their efficiency.

We next propose a core-based approach for locating a CDS, by quickly converging on smaller dense subgraphs that contain the CDS. To make our approach applicable for processing all the h-clique-density definitions ($h \geq 2$), we lift the notion of k-cores to k-clique-cores and study how to exploit them in the DSD solution.

5. THE CLIQUE-BASED CORES

We now study the k-clique-core, or (k, Ψ)-core, which is a generalization of the classical k-core [62, 7] for an h-clique Ψ (Section 5.1). As we will show, (k, Ψ)-cores are useful in locating the CDS in both exact and approximation algorithms. We then establish upper and lower bounds on the clique-density of (k, Ψ)-cores (Section 5.2), present efficient algorithms for decomposing (k, Ψ)-cores (Section 5.3), and give some discussions (Section 5.4).

5.1 k-core and (k, Ψ)-core

We first review the definition of k-core.

DEFINITION 5 (k-CORE [62, 7]). *Given a graph G and an integer k ($k \geq 0$), the k-core, denoted by $\mathcal{H}_k$, is the largest subgraph of G, such that $\forall v \in \mathcal{H}_k$, $deg_{\mathcal{H}_k}(v) \geq k$.*

We say that $\mathcal{H}_k$ has order k. The *core number* of a vertex $v \in V$ is defined as the highest order of a k-core that contains v. In other words, a k-core is the largest subgraph induced by vertices whose core numbers are at least k. A k-core has some interesting properties [7]: (1) k-cores are "nested": given two nonnegative integers i and j, if $i < j$, then $\mathcal{H}_j \subseteq \mathcal{H}_i$; (2) a k-core may not be connected; and (3) computing core numbers of all the vertices in a graph, known as k-core decomposition, can be done in linear time.
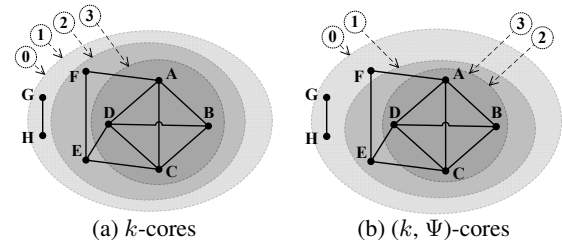

(a) k-cores   (b) (k, Ψ)-cores

Figure 3: k-core, and (k, Ψ)-core (Ψ is a triangle).

EXAMPLE 2. Figure 3(a) depicts a graph of 8 vertices and its k-cores. The number k in each ellipse indicates the k-core contained in that ellipse. For instance, the subgraph induced by $\{A, B, C, D\}$ is the 3-core, and the entire graph is both the 0-core and 1-core, which consist of two connected components. □

DEFINITION 6 ((k, Ψ)-CORE). *Given a graph G, an integer k ($k \geq 0$), and an h-clique Ψ, the (k, Ψ)-core, denoted by $\mathcal{R}_k$, is the largest subgraph of G such that $\forall v \in \mathcal{R}_k$, $deg_{\mathcal{R}_k}(v, \Psi) \geq k$.*

Similar to k-cores, we say that $\mathcal{R}_k$ has order k. The *clique-core number* of a vertex $v \in V$, $core_G(v, \Psi)$, is then the highest order of a (k, Ψ)-core containing v. We denote the maximum clique-core number by $k_{\max}$, where the underlying clique Ψ is understood from the context. Given a clique Ψ, a (k, Ψ)-core also has the following properties: (1) (k, Ψ)-cores are "nested": given two nonnegative integers i and j, if $i < j$, then $\mathcal{R}_j \subseteq \mathcal{R}_i$; (2) a (k, Ψ)-core may not be connected; and (3) $core_G(v, \Psi) \leq deg_G(v, \Psi)$.

EXAMPLE 3. Let Ψ be the triangle. Figure 3(b) shows all (k, Ψ)-cores of the graph. The number k in each circle indicates the (k, Ψ)-core contained in that ellipse. For instance, the subgraph of $\{A, B, C, D\}$ is the $(3, \Psi)$-core as the 4-clique contains 4 triangle instances, and each vertex participates in 3 of them. Observe that k-cores and (k, Ψ)-cores are different between Figures 3(a) and 3(b), for k=1, 2. Also, the entire graph is a $(0, \Psi)$-core. □

5.2 Density Bounds of (k, Ψ)-core

The main result of this section is on the lower and upper bounds on the density of a (k, Ψ)-core.

THEOREM 1. *Given a graph G and an h-clique $\Psi(V_\Psi, E_\Psi)$, let $\mathcal{R}_k$ be a (k, Ψ)-core of G. Then, the h-clique-density of $\mathcal{R}_k$ satisfies*

$$\frac{k}{|V_\Psi|} \leq \rho(\mathcal{R}_k, \Psi) \leq k_{\max}. \quad (2)$$

To prove this theorem, we develop the following lemmas.

LEMMA 3. *Given a graph G and an h-clique Ψ, the connected components of CDS D have the same clique-density.*

PROOF SKETCH. The lemma can be proved by contradiction. □

LEMMA 4. *Given a graph $G(V, E)$, an h-clique $\Psi(V_\Psi, E_\Psi)$, and the CDS $D(V_D, E_D)$, for any subset U of V_D, removing U from D will result in the removal of at least $\rho_{opt} \times |U|$ clique instances from D.*

PROOF. We prove the lemma by contradiction. Assume that D is the CDS and the removal of U results in removing less than $\rho_{opt} \times |U|$ clique instances. Then, after removing U from V_D, the clique-density of the residual graph (denoted by $D \backslash U$) becomes:

$$\rho(D \backslash U, \Psi) = \frac{\mu(D \backslash U, \Psi)}{|V_D| - |U|} > \frac{\rho_{opt}|V_D| - \rho_{opt}|U|}{|V_D| - |U|} = \rho_{opt}. \quad (3)$$

However, this contradicts the assumption that D is the CDS. Hence, the lemma holds. □

Based on the lemma above, we show an upper bound of ρ_{opt}.

LEMMA 5. *Given a graph G, an h-clique $\Psi(V_\Psi, E_\Psi)$, and its maximum clique-core number $k_{\max}$, we have:*

$$\rho_{opt} \leq k_{\max}. \quad (4)$$

PROOF. We prove the lemma by contradiction. Suppose that we have $\rho_{opt} > k_{\max}$. From Lemma 4, we know that removing any vertex of D will result in the removal of at least ρ_{opt} clique instances, or more than $k_{\max}$ clique instances from D. In other words, each vertex of D participates in at least $k_{\max}+1$ clique instances. This contradicts the fact that $k_{\max}$ is the maximum clique-core number. Hence, the value of ρ_{opt} is at most $k_{\max}$. □

Proof of THEOREM 1: The upper bound follows by Lemma 5. Let us focus on the lower bound. Let r_k be the number of vertices in $\mathcal{R}_k$. By Definition 6, since $\mathcal{R}_k$ is a (k, Ψ)-core, each vertex v of $\mathcal{R}_k$ participates in at least k clique instances. Meanwhile, each clique instance involves $|V_\Psi|$ vertices. As a result, there are at least $\frac{k \times r_k}{|V_\Psi|}$ clique instances in $\mathcal{R}_k$. Thus, we have $\rho(\mathcal{R}_k, \Psi) \geq \frac{k}{|V_\Psi|}$. □
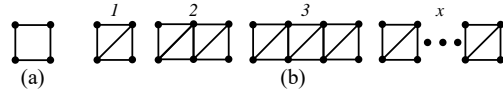
To further illustrate Theorem 1, we give Example 4.

Figure 4: Illustrating the lower and upper bounds.

EXAMPLE 4. Let Ψ be an edge and consider the $k_{\max}$-core with $k_{\max}$=2. By Theorem 1, the lower and upper bounds of the density of $k_{\max}$-core are 1 and 2 respectively. These bounds are attained by graphs in Figures 4(a) and 4(b) respectively. In Figure 4(a), the density of the $k_{\max}$-core is 4/4=1. In Figure 4(b), there is a list of graphs with $k_{\max}$=2, and the density values of $k_{\max}$-cores in the 1st, 2nd, $\cdots$, x-th graphs are $\frac{1+4}{2+2}$, $\frac{1+8}{2+4}$, $\cdots$, $\frac{1+4x}{2+2x}$, respectively. Clearly, when $x \to \infty$, the density converges to 2. □

5.3 (k, Ψ)-Core Decomposition

Inspired by the k-core decomposition algorithm [7], we develop an efficient (k, Ψ)-core decomposition algorithm for computing the clique-core number of each vertex. The algorithm exploits a key observation that, if we recursively remove vertices whose clique-degrees are less than a non-negative integer k, then the remaining graph, if non-empty, must be the (k, Ψ)-core.

Specifically, we first compute the clique-degree of each vertex, and sort vertices in increasing order of their clique-degrees. Then, we iteratively remove the vertex v whose clique-degree is the smallest in each iteration, until the graph is empty. In each iteration, after removing v, we need to decrease the clique-degrees of vertices, which share clique instances with v, and re-sort the vertices. Notice that by using the bin-sort technique [7], sorting all the vertices takes linear time and re-sorting can also be done efficiently.

Algorithm 3: (k, Ψ)-core decomposition.

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$;
Output: The clique-core number of each vertex;
1   initialize $core[\ ] \leftarrow$ an array with n entries;
2   **for** *each vertex $v \in V$* **do** compute its clique-degree $deg_G(v, \Psi)$;
3   sort vertices of V in increasing order of their clique-degrees;
4   **while** V *is not empty* **do**
5     $core[v] \leftarrow deg_G(v, \Psi)$ where v has the minimum clique-degree;
6   **for** *each clique instance ψ containing v* **do**
7   **for** *each vertex u in ψ* **do**
8   **if** $deg_G(u, \Psi) > deg_G(v, \Psi)$ **then**
9   decrease u's clique-degree;
10   update G by removing v and its incident edges;
11   resort the vertices in V;
12 **return** the array $core[\ ]$;

Algorithm 3 presents the core decomposition algorithm. First, we initialize an array $core[\ ]$ and compute the clique-degree of each vertex (lines 1-2). Then, we sort all the vertices in increasing order (line 3). Next, we recursively remove the vertex v whose clique-degree is the smallest (lines 4-11). In each iteration, we record v's clique-core number (line 5), decrease the clique-degrees of vertices in v's clique instances as removing v causes the deletion of some clique instances (lines 6-9), update G, and resort vertices (lines 10-11). Finally, we return $core[\ ]$ (line 12).

To compute the clique-degrees of all the vertices, we can first run an h-clique enumeration algorithm, and then compute the clique-degree of each vertex by listing all the h-cliques. During the core decomposition process, after removing a vertex v, we can first locate the subgraph induced by v and its neighbors, then enumerate all the h-cliques in this subgraph, and finally decrease the clique-degrees of the vertices involved. In this paper, we use the state-of-the-art h-clique enumeration algorithm [17].

LEMMA 6. *Given a graph G and an h-clique $\Psi(V_\Psi, E_\Psi)$, the core decomposition algorithm above completes in $\mathcal{O}\left(n \cdot \binom{d-1}{h-1}\right)$ time and $\mathcal{O}(m)$ space.*

PROOF. For each vertex v, we need to compute the number of clique instances it involves, i.e., $deg_G(v, \Psi)$. In the worst case, any $h-1$ neighbors of v can form an h-clique with v, so $deg_G(v, \Psi)$ is up to $\binom{d-1}{h-1}$. By using the bin-sort technique in [7], we can sort vertices of V in linear time cost, and resorting after removing a vertex takes linear time cost to the clique-degree. In addition, computing $deg_G(v, \Psi)$ takes $\mathcal{O}(m)$ space as we can compute the clique instances sequentially. Hence, Lemma 6 holds. □

5.4 Extension and Discussion

The k-clique-core can be extended to k-pattern-core by incorporating a general pattern (e.g., star, loop, etc.). Let Ψ be a pattern. Then, the (k, Ψ)-core is the largest subgraph of G, in which each vertex participates in at least k instances of Ψ. The properties of k-clique-cores also hold for k-pattern-core. Besides, for any two patterns Ψ and Ψ', if $|V_\Psi|=|V_{\Psi'}|$ and $\Psi \subseteq \Psi'$, i.e., Ψ is a subpattern of Ψ', then the (k, Ψ')-core is a subgraph of the (k, Ψ)-core. Algorithm 3 can also be extended for decomposing k-pattern-cores. We skip the details due to the space limitation.

Recently, Sariyüce et al. studied the k-(r, s) nucleus [60, 58, 59], which is the maximal connected subgraph of the r-cliques where each r-clique is contained in at least k s-cliques ($r<s$). When Ψ is an h-clique, our (k, Ψ)-core can be considered as a special case of k-(r, s) nucleus, i.e., k-$(1, h)$ nucleus, in terms of clique-degree (or $\mathcal{S}$-degree in [59]). However, when Ψ is a non-clique, (k, Ψ)-core is different with the k-(r, s) nucleus, because in our (k, Ψ)-core, Ψ can be an arbitrary pattern, such as clique, star, loop, etc., while k-(r, s) nucleus is defined purely based on cliques. In other words, our (k, Ψ)-core can capture pattern-based dense subgraphs. A second difference is that a k-(r, s) nucleus requires that any two r-cliques R and R' are $\mathcal{S}$-connected: i.e., there exists a sequence of r-cliques $R=R_1, R_2, \cdots, R_l=R'$, such that R_i, R_{i+1} are both contained by a specific s-clique ($i \in [1, l-1]$). In addition, when Ψ is an h-clique, the nucleus decomposition algorithm [59] can be applied to decomposing (k, Ψ)-cores. We will experimentally compare this method with ours in Section 8.1.

6. CORE-BASED APPROACHES

Based on (k, Ψ)-cores, we develop efficient exact and approximation DSD algorithms. While our exact algorithm, `CoreExact`, is significantly faster than the state-of-the-art algorithm (`Exact`), we can speed it up further by trading accuracy: we develop an efficient approximation algorithm, namely `CoreApp`, which has an approximation ratio of $\frac{1}{|V_\Psi|}$.

6.1 The Core-Based Exact Method

As shown in Lemma 1, the major limitation of Algorithm `Exact` is its high computational cost. To address this, in this section we exploit the k-clique-cores and propose the following three optimization techniques for boosting the efficiency.

1 Tighter bounds on α. In `Exact`, the value of α is within the range $[0, \max_{v \in V} deg_G(v, \Psi)]$. As discussed in Section 5.2, by using the (k, Ψ)-cores, we can derive a tighter bound on α. Specifically, consider a $(k_{\max}, \Psi)$-core $\mathcal{R}_{k_{\max}}$. By Theorem 1, we can see that $\rho(\mathcal{R}_{k_{\max}}, \Psi) \geq \frac{k_{\max}}{|V_\Psi|}$, which implies that $\rho_{opt} \geq \frac{k_{\max}}{|V_\Psi|}$, so the lower bound of α is $\frac{k_{\max}}{|V_\Psi|}$. On the other hand, by Lemma 5, we have $\rho_{opt} \leq k_{\max}$ and thus the upper bound of α is $k_{\max}$. In practice, since $\frac{k_{\max}}{|V_\Psi|}$ is larger than 0 and $k_{\max}$ is smaller than the maximum clique-degree, the number of binary searches can be greatly reduced by using the tighter bounds.

2 Locating the CDS in a core. Recall that in each binary search of the algorithm `Exact`, the flow network is reconstructed based on the entire graph G. This, however, is unnecessary, since the CDS is often in some (k, Ψ)-cores which could be much smaller than G.

LEMMA 7. *Given a graph G and an h-clique Ψ, the CDS is contained in the (k, Ψ)-core, where $k = \lceil \rho_{opt} \rceil$.*

PROOF. By Lemma 4, deleting any single vertex from CDS will result in the removal of $\lceil \rho_{opt} \rceil$ clique instances in CDS. In other words, each vertex of the CDS has participated in $\lceil \rho_{opt} \rceil$ clique instances. By the definition of (k, Ψ)-core, we conclude that the CDS is in the (k, Ψ)-core, where $k=\lceil \rho_{opt} \rceil$. □

As the value of ρ_{opt} may not be known in advance, we can only locate it in the cores using the lower bounds of ρ_{opt}, by exploiting the nested property of cores. For example, by Theorem 1, we have $\rho_{opt} \geq \frac{k_{\max}}{|V_\Psi|}$, which implies that the CDS must be in the (k, Ψ)-core, where $k=\left\lceil \frac{k_{\max}}{|V_\Psi|} \right\rceil$. Recall that in the core decomposition process, we delete vertices iteratively and obtain a residual subgraph after removing a vertex. In order to get a tighter lower bound on ρ_{opt}, we can compute the densities of these residual subgraphs.

• *Pruning1:* The CDS is in the (k', Ψ)-core, where $k'=\lceil \rho' \rceil$ and ρ' is the highest h-clique-density of all residual graphs. The correctness directly follows Lemma 7, since $\rho' \leq \rho_{opt}$.

Since the (k', Ψ)-core may be disconnected and some connected components may be denser than others, we can further locate the CDS in a core with a larger core number, using *Pruning2*.

• *Pruning2:* For each connected component of the (k', Ψ)-core, we compute its h-clique-density. Let ρ'' be the maximum h-clique-density of these connected components. If $\lceil \rho'' \rceil >k'$, we increase k' to $k''=\lceil \rho'' \rceil$ and the CDS is in the (k'', Ψ)-core. The correctness holds by Lemma 7, since $\rho' \leq \rho'' \leq \rho_{opt}$.

• *Pruning3:* After locating the CDS in a connected component $C(V_C, E_C)$, we can change the stopping criterion of binary search to "$u-l<\frac{1}{|V_C|(|V_C|-1)}$". Since $C(V_C, E_C)$ contains the CDS and the flow network is built using $C(V_C, E_C)$, the pruning is correct by following Algorithm 1.

3 The flow network gradually becomes smaller. During the binary search, since the lower bound l of α is gradually enlarged, we can locate the CDS in cores with larger clique-core numbers. As clique-core numbers increase, the sizes of cores become smaller, so the flow networks constructed become smaller gradually, and the cost of computing the minimum st-cut is greatly reduced.

Combining the three optimization techniques above, we develop an advanced exact algorithm, called `CoreExact`, as presented in Algorithm 4. We first perform core decomposition and locate the CDS in the (k'', Ψ)-core (lines 1-2). Then, we initialize some variables including the lower and upper bounds of α and put the connected components of (k'', Ψ)-core into a set $\mathcal{C}$ (lines 3-4). Next, in the loop (lines 5-20), we consider the connected components one by one. Note that the lower bound l is never decreased during the iterations. If the current lower bound $l>k''$, we replace C by the core which has higher clique-core number and is contained by C (line 6). Intuitively, if l is too large, then C may not contain a subgraph with density l and thus we skip it. Consequently, we build a flow network (line 7), and check whether l is a feasible lower bound (line 8), i.e., whether there exists a subgraph with density at least l.

If C cannot be skipped, we use binary search to find the CDS (lines 10-19). Each time we set a guess value of ρ_{opt}, namely α, and check whether there is a subgraph with density of α or more. Once we get a larger lower bound (line 16), we locate the CDS in the core with a larger clique-core number, so the network based on C is even smaller. In other words, during the binary search, as the value of α approaches the true value of ρ_{opt}, the flow networks

Algorithm 4: The algorithm: `CoreExact`.

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$;
Output: The CDS $D(V_D, E_D)$;
1   perform core decomposition using Algorithm 3;
2   locate the (k'', Ψ)-core using pruning criteria;
3   initialize $\mathcal{C} \leftarrow \emptyset, D \leftarrow \emptyset, U \leftarrow \emptyset, l \leftarrow \rho'', u \leftarrow k_{\max}$;
4   put all the connected components of (k'', Ψ)-core into $\mathcal{C}$;
5   **for** *each connected component* $C(V_C, E_C) \in \mathcal{C}$ **do**
6   **if** $l > k''$ **then** $C(V_C, E_C) \leftarrow C \cap (\lceil l \rceil, \Psi)$-core;
7   build a flow network $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$ by lines 5-15 of Algorithm 1;
8   find minimum st-cut $(\mathcal{S}, \mathcal{T})$ from $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$;
9   **if** $\mathcal{S} = \emptyset$ **then** continue;
10   **while** $u - l \geq \frac{1}{|V_C|(|V_C| - 1)}$ **do**
11        $\alpha \leftarrow \frac{l + u}{2}$;
12   build $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$ by lines 5-15 of Algorithm 1;
13   find minimum st-cut $(\mathcal{S}, \mathcal{T})$ from $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$;
14   **if** $\mathcal{S} = \{s\}$ **then**
15          $u \leftarrow \alpha$;
16   **else**
17   **if** $\alpha > \lceil l \rceil$ **then** remove some vertices from C;
18          $l \leftarrow \alpha$;
19          $U \leftarrow \mathcal{S} \setminus \{s\}$;
20   **if** $\rho(G[U], \Psi) > \rho(D, \Psi)$ **then** $D \leftarrow G[U]$;
21   **return** D;

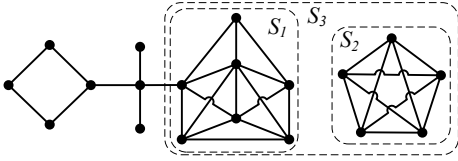constructed become smaller. Finally, we get the CDS (line 21). We illustrate `CoreExact` by Example 5.

Figure 5: Illustrating the core-based algorithms.

EXAMPLE 5. Let Ψ be a single edge and consider the graph in Figure 5, where $k_{\max} = 4$. During core decomposition, we track densities of residual graphs and obtain $\rho' = 25/12 \approx 2.08$ (i.e., density of subgraph S_3). Thus, we get $\lceil \rho' \rceil = 3$ and locate the EDS in the 3-core (i.e., subgraph S_3). The EDS (i.e., subgraph S_1 with density $15/7 \approx 2.14$) can be computed by conducting binary search using the flow networks built on the two connected components S_1 and S_2 of S_3, rather than the entire graph, respectively. □

6.2   The Core-Based Approximation Methods

Recall that Theorem 1 gives the lower and upper bounds of density of a (k, Ψ)-core. Moreover, for a specific clique Ψ, the larger the value of k, the higher the lower bound on the density of the corresponding (k, Ψ)-core. Using Theorem 1, we can show:

LEMMA 8. *Given a graph G and an h-clique $\Psi(V_\Psi, E_\Psi)$, the $(k_{\max}, \Psi)$-core is a $\frac{1}{|V_\Psi|}$-approximation solution to CDS problem.*

PROOF. By Theorem 1, we have $\frac{k_{\max}}{|V_\Psi|} \leq \rho(\mathcal{R}_{k_{\max}}, \Psi) \leq k_{\max}$. Using the fact that $\rho_{opt} \leq k_{\max}$, we have

$$\frac{\rho(\mathcal{R}_{k_{\max}}, \Psi)}{\rho_{opt}} \geq \frac{k_{\max} / |V_\Psi|}{k_{\max}} = \frac{1}{|V_\Psi|}. \tag{5}$$

The lemma follows. □

To compute the $(k_{\max}, \Psi)$-core, we can use the core decomposition method discussed in Section 5.3, which computes all the cores in an incremental manner. We denote this approximation algorithm by `IncApp` (see Algorithm 5). Clearly, it has the same time complexity as the core decomposition algorithm.

Algorithm 5: The algorithm: `IncApp`.

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$;
Output: The $(k_{\max}, \Psi)$-core;
1   run (k, Ψ)-core decomposition algorithm (Section 5.3);
2   **return** the $(k_{\max}, \Psi)$-core;

A subtle point is that although the $(k_{\max}, \Psi)$-core is dense and provides an approximation solution, the CDS may not be in the $(k_{\max}, \Psi)$-core or even share some vertices with it. For example, in Figure 5, let Ψ be a single edge. Then, the subgraph S_2 is the $k_{\max}$-core ($k_{\max} = 4$), but the EDS is the subgraph S_1.

To further improve efficiency, we propose another method, called `CoreApp`. Unlike `IncApp` which computes all the cores, it focuses on computing the $(k_{\max}, \Psi)$-core directly. It relies on a key observation that the $(k_{\max}, \Psi)$-core often tends to be a subgraph of vertices with higher clique-degrees. We thus propose to discover the CDS from a sequence of subgraphs induced by vertices, whose clique-degrees are the largest. Moreover, once we find a core with higher clique-core number, we can prune some subgraphs, whose vertices' clique-degrees are too small. Thus, the CDS can be discovered efficiently. We remark that for h-cliques where $h \geq 3$, computing the clique-degree $deg_G(v, \Psi)$ may be costly. Instead, we replace it by an upper bound $\gamma(v, \Psi)$, which can be computed more efficiently. Specifically, we run the k-core decomposition algorithm [7], and for each vertex v in an x-core, we set $\gamma(v, \Psi) = \binom{x}{h-1}$.

Algorithm 6: The algorithm: `CoreApp`.

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$;
Output: The $(k_{\max}, \Psi)$-core;
1   **for** $\forall v \in V$ **do** compute $\gamma(v, \Psi)$ of $deg_G(v, \Psi)$;
2   sort vertices of V in decreasing order of their $\gamma(v, \Psi)$ values;
3   initialize $W, k_{\max} \leftarrow 0, S^* \leftarrow \emptyset$;
4   **while** $\max\limits_{v \in V \setminus W} \gamma(v, \Psi) \geq k_{\max}$ **do**
5   **for** $\forall v \in W$ **do** compute $deg_{G[W]}(v, \Psi)$;
6      $k_l \leftarrow \min\limits_{v \in W} deg_{G[W]}(v, \Psi), k_u \leftarrow \max\limits_{v \in W} deg_{G[W]}(v, \Psi)$;
7      $k \leftarrow \max\{k_l, k_{\max} + 1\}$;
8   **while** $k \leq k_u$ *and* $|W| > 0$ **do**
9   **while** $(\exists v \in W, deg_{G[W]} < k)$ **do**
10   delete v from W and decrease clique-degrees;
11   **if** $|W| > 0$ **then**
12   **if** $k > k_{\max}$ **then**
13            $k_{\max} \leftarrow k, S^* \leftarrow G[W]$;
14        $k \leftarrow k + 1$;
15      $W \leftarrow$ top-$(2 \times |W|)$ vertices in V;
16   **return** S^*;

Algorithm 6 presents `CoreApp`. First, we compute $\gamma(v)$ for each vertex, and sort vertices based on their $\gamma(v)$ values (lines 1-2). Then, we initialize three variables W, $k_{\max}$, and S^*, where W keeps a set of vertices whose clique-degrees are the largest, and S^* is used to track the $(k_{\max}, \Psi)$-core (line 3), which is computed from the vertex-induced subgraph $G[W]$. Next, we compute the $(k_{\max}, \Psi)$-core in G, by using a while loop (lines 4-15). Specifically, we first compute the exact clique-degree for each vertex in $G[W]$, and record the minimum and maximum clique-degrees (lines 6-7). Then, we perform core decomposition for $G[W]$ with clique-core numbers in $[k_l, k_u]$ (lines 8-15), during which the maximum clique-core number $k_{\max}$ and $(k_{\max}, \Psi)$-core are kept (lines 13-14). After that, we double the size of W for the next iteration (line 15). The loop can be stopped safely by using the stopping criterion (line 4). Finally, we get $(k_{\max}, \Psi)$-core (line 16). Note that $k_{\max}$ tracks the maximum clique-core number during the iterations, and for each subgraph $G[W]$, we focus on finding cores with core numbers larger than the previous $k_{\max}$ (line 7).

Correctness. Essentially, `CoreApp` finds the $(k_{\max}, \Psi)$-core from a sequence of subgraphs induced by vertices in W which have the largest clique-degrees. For each small subgraph $G[W]$, it computes the core with the highest core number by running core decomposition steps (lines 7-14). The stopping criterion (line 4) ensures that the $(k_{\max}, \Psi)$-core is correctly computed, i.e., since the maximum clique-degree of all the remaining vertices (in the set $V \backslash W$) is less than $k_{\max}$, their clique-core numbers must be less than $k_{\max}$.

LEMMA 9. *The time and space complexities of* `CoreApp` *are* $\mathcal{O}\left(n \cdot \binom{d-1}{h-1}\right)$ *and* $\mathcal{O}(m)$ *respectively.*

PROOF. Let the number of iterations be t. Since we adopt the exponential growth strategy, the numbers of vertices involved in these iterations are at most $\left(\frac{1}{2}\right)^{t-1} \cdot n, \left(\frac{1}{2}\right)^{t-2} \cdot n, \cdots, n$ respectively, which form a geometric sequence. In the i-th iteration $i \in [1, t]$, it takes $\mathcal{O}\left(\left(\frac{1}{2}\right)^{t-i} \cdot n \cdot \binom{d-1}{h-1}\right)$ time and $\mathcal{O}(m)$ space, as it performs core decomposition. By summarizing the time cost of all iterations, we obtain $\mathcal{O}\left(2 \cdot n \cdot \binom{d-1}{h-1}\right) = \mathcal{O}\left(n \cdot \binom{d-1}{h-1}\right)$. The space cost is $\mathcal{O}(m)$, since the iterations are sequentially executed. □

Although, `CoreApp` has almost the same *worst-case cost* as `PeelApp` and `IncApp`, it performs much faster in practice, because the CDS is often much smaller than G and thus only a few subgraphs are examined in the iterations. As shown by our experiments next, `CoreApp` is up to two orders of magnitude faster than `PeelApp` and `IncApp`. Moreover, the approximation algorithms generate high-quality solutions – their actual approximation ratios are often much higher than their theoretical approximation ratios.
Remark. In [13], Cheng et al. present an external-memory core decomposition algorithm, called `EMcore`, which also works in a top-down manner. However, there are four differences between `CoreApp` and `EMcore`: (1) `CoreApp` can handle any h-clique- and pattern-cores, while `EMcore` is developed for processing the classical (edge-based) k-cores. (2) `CoreApp` focuses on computing the $(k_{\max}, \Psi)$-core while `EMcore` decomposes all k-cores. (3) The methods of estimating upper bounds of core numbers are different. (4) In the worst case, for classical k-cores, `CoreApp` takes $\mathcal{O}(n + m)$ time while `EMcore` takes $\mathcal{O}(k_{\max}(n + m))$ time, since both of them conduct core decomposition for a sequence of subgraphs, but the strategies of considering subgraphs are different. Our later experiments show that for computing the $k_{\max}$-core, `CoreApp` is faster than `EMcore`.

6.3 Discussions

Below, we discuss the parallelizability of our algorithms and show that our algorithms can solve a variant of the CDS problem.
Parallelizability. The existing parallel k-core decomposition algorithms [50, 48, 59] can be easily extended for decomposing (k, Ψ)-cores, so our approximation solutions, which rely on the $(k_{\max}, \Psi)$-core, can be computed in parallel. Moreover, for the exact solution `CoreExact`, the main overhead comes from the step of computing the minimum st-cut. The parallel algorithms of computing the minimum st-cut have been studied extensively [42, 52], so our exact algorithm can also be easily parallelized.
A variant of CDS problem. In [65], Tsourakakis et al. studied a variant of the densest k subgraph problem [8, 3], which aims to find a subgraph that contains a given set Q of k query vertices ($|Q|=k$) with the highest density, and its exact solution follows the framework of the exact solution of CDS problem by solving a maximum flow problem. To solve this problem with edge-density, we can first decompose k-cores and get the minimum core number x of these k vertices. Then, then lower bound of the edge-density of x-core is $\frac{x}{2}$ by Theorem 1. Since x-core contains Q, we get a lower bound

of ρ_{opt} which is $\frac{x}{2}$. As a result, we can locate the densest subgraph in $\frac{x}{2}$-core, so we can build a flow network on $\frac{x}{2}$-core, rather than the entire graph, resulting in higher efficiency.

7. THE PDS PROBLEM AND SOLUTIONS

A *pattern* (a.k.a. *motif* or higher-order structure) is a small graph containing a few vertices (e.g., a diamond in Figure 1(b)). These patterns can be considered as building blocks of knowledge graphs or biological databases [70, 34, 21]. Compared to graph edges, they can better capture the intricate relationship among vertices, as well as the underlying rich semantics. For example, in a protein interaction network, proteins are often organized in cohesive patterns of interactions, each of which represents some particular functions [70]. We now study the discovery of *pattern-aware densest subgraphs*, i.e., subgraphs that are "dense" in terms of the number of patterns. We term this pattern densest subgraph (PDS) problem and show how our previous CDS solutions can be adapted.

7.1 The PDS problem

We generalize the h-clique to a general pattern, which is a connected simple graph $\Psi(V_\Psi, E_\Psi)$. We formally introduce definitions of pattern instance and pattern-density below.

DEFINITION 7 (SUBGRAPH ISOMORPHISM). *A graph $G(V, E)$ is subgraph isomorphic to a pattern $\Psi(V_\Psi, E_\Psi)$ if there exists an injection $\phi : V_\Psi \to V$, such that for all $v, v' \in V_\Psi$, if $(v, v') \in E_\Psi$, then $(\phi(v), \phi(v')) \in E$.*

DEFINITION 8 (PATTERN INSTANCE). *Given a graph $G(V, E)$ and a pattern $\Psi(V_\Psi, E_\Psi)$, a subgraph $S(V_S, E_S) \subseteq G$ is a pattern instance of Ψ, if S is isomorphic to Ψ.*

DEFINITION 9 (PATTERN-DEGREE). *Given a graph $G(V, E)$ and a pattern Ψ, the pattern-degree of a vertex v, or $deg_G(v, \Psi)$, is the number of pattern instances of Ψ containing v.*

Clearly, G is subgraph isomorphic to Ψ iff it has a subgraph $S(V_S, E_S)$ that is isomorphic to Ψ. Note that S may not be a vertex-induced subgraph, although we note that our algorithms can be easily adapted for the vertex-induced case. Due to symmetry, for a single subgraph S of G, there may be multiple mappings witnessing that Ψ is isomorphic to S, which are automorphisms, but in this case we do not distinguish between different automorphisms of S and instead count instances based on the edge set.

DEFINITION 10 (PATTERN-DENSITY). *Given a graph $G(V, E)$ and a pattern $\Psi(V_\Psi, E_\Psi)$, the pattern-density of G w.r.t. Ψ is $\rho(G, \Psi) = \frac{\mu(G, \Psi)}{|V|}$, where $\mu(G, \Psi)$ is the number of pattern instances of Ψ in G.*

PROBLEM 2 (PDS PROBLEM). *Given a graph $G(V, E)$ and a pattern $\Psi(V_\Psi, E_\Psi)$, return the subgraph D of $G(V, E)$, whose pattern-density $\rho(D, \Psi)$ is the highest.*

For example, consider the graph in Figure 6(a) and let Ψ be the diamond pattern (Figure 1(b)). Then, the subgraph of $\{A, D, E, F\}$ is the densest subgraph, which contains three pattern instances (Figure 6(c)) and has the highest pattern-density.

7.2 Algorithms for PDS Problem

Approximation algorithms. To compute the approximate PDS's, we can directly adapt algorithm `PeelApp` by replacing the steps of computing clique instances (clique-degrees) by pattern instances (pattern-degrees). The correctness is guaranteed by Lemma 10. Similarly, `IncApp` and `CoreApp` can be adapted.

LEMMA 10. *Given a graph G and a pattern $\Psi(V_\Psi, E_\Psi)$, the subgraph S^* returned by* PeelApp *is a $\frac{1}{|V_\Psi|}$-approximation solution to the PDS problem w.r.t. pattern-density for pattern Ψ.*

PROOF SKETCH. We can prove the lemma by generalizing Lemma 8 and Theorem 1 for supporting an arbitrary pattern. $\square$

Exact algorithms. The algorithm Exact in Section 4.1 cannot be trivially extended for computing the exact PDS's since it relies on $(h-1)$-cliques. Nevertheless, we can adapt the exact CDS algorithm in [65], which follows the framework of Exact but introduces a different flow network construction method, for computing the exact PDS's by replacing the steps of computing clique instances (clique-degrees) by pattern instances (pattern-degrees). We denote this algorithm by PExact, and its pseudocodes are presented in the technical report [27]. Theorem 2 shows its correctness.

THEOREM 2. *Given a graph G and a pattern $\Psi(V_\Psi, E_\Psi)$, the algorithm* PExact *correctly finds the PDS of G w.r.t. pattern-density of Ψ.*

PROOF. Please refer to the technical report [27]. $\square$

Our core-based techniques can be used for improving PExact. Specifically, we adopt the k-pattern-core in Section 5.4 and use the three optimization techniques in Section 6.1. In addition, we propose a new optimization strategy, which relies on the following key observation: for a general pattern Ψ, different pattern instances may share the same set of vertices, but PExact creates a node for each of them when building the flow network. For example, consider the graph in Figure 6(a). If the pattern is a diamond, then the three pattern instances in Figure 6(c) share the same set of vertices.

Algorithm 7: construct+(G, Ψ, α).

Input: $G(V, E)$, $\Psi(V_\Psi, E_\Psi)$, α;
Output: The flow network $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$;
1 $\Lambda \leftarrow$ all the pattern instances of Ψ in G;
2 $\Lambda' = \{g_1, g_2, \cdots, g_{|\Lambda'|}\} \leftarrow$ group the pattern instances in Λ;
3 $V_\mathcal{F} \leftarrow \{s\} \cup V \cup \Lambda' \cup \{t\}$;
4 $\forall v \in V$, add an edge $s \rightarrow v$ with capacity $deg_G(v, \Psi)$;
5 $\forall v \in V$, add an edge $v \rightarrow t$ with capacity $\alpha|V_\Psi|$;
6 $\forall v \in V$, if it appears in a group $g \in \Lambda'$, add an edge $v \rightarrow g$ with capacity $|g|$;
7 $\forall g \in \Lambda'$, if it contains a vertex v, add an edge $g \rightarrow v$ with capacity $|g|(|V_\Psi| - 1)$;
8 **return** $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$;

Based on the observation above, we propose a new flow network construction method construct+, by grouping nodes of pattern instances having same set of vertices. Algorithm 7 shows construct+. First, a set $\Lambda' = \{g_1, g_2, \cdots, g_{|\Lambda'|}\}$ is collected, where each g_i denotes a group of pattern instances sharing the same set of vertices. Second, for each vertex $v \in V$, we set the capacities of edges (s, v) and (v, t) similarly with that in PExact. Third, for each vertex $v \in V$, if it appears in a group $g \in \Lambda'$, the capacity of edge (v, g) is set to $|g|$; for each group $g \in \Lambda'$, the capacity of edge (g, v) is set to $|g|(|V_\Psi|-1)$. Here, we define the capacties based on the intuition that the densest subgraph D is obtained by computing the minimum st-cut $(\mathcal{S}, \mathcal{T})$, and vertices of D must be in one partition $\mathcal{S}$. This implies that nodes of all the pattern instances in D should be in $\mathcal{S}$, and thus we can accumulate their capacities by using the term $|g|$ when computing the maximum flow from $\mathcal{S}$ to $\mathcal{T}$. Note that if Ψ is a clique, then $|g|=1$. The correctness is stated by Lemma 11. We illustrate construct+ by Example 6. We denote the above core-based exact PDS algorithm by CorePExact.

LEMMA 11. *Given a graph G, a pattern $\Psi(V_\Psi, E_\Psi)$, the flow networks built by* PExact *(lines 5-12) and* construct+ *have the same capacity for their minimum st-cut.*
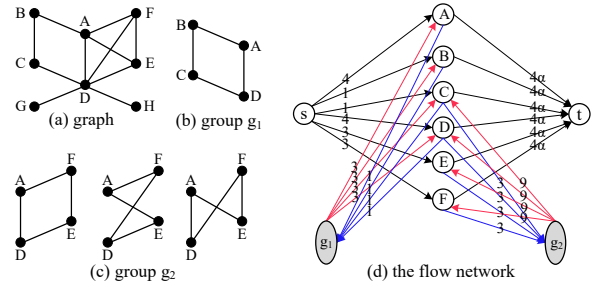

Figure 6: Illustrating the flow network in CorePExact.

PROOF. Please refer to the technical report [27]. $\square$

EXAMPLE 6. Let Ψ be the diamond pattern. The graph in Figure 6(a) has 4 pattern instances, which are grouped into 2 groups as shown in Figures 6(b) and 6(c). Clearly, we can locate the PDS in $(1, \Psi)$-core, in which the vertex set is $\{A, B, \cdots, F\}$ and $\Lambda' = \{g_1, g_2\}$. To build $\mathcal{F}(V_\mathcal{F}, E_\mathcal{F})$, we first collect the set $V_\mathcal{F}$, then create 10 nodes, and finally add edges. For example, for group g_2, we link it to all its vertices with capacities $|g_2|(|V_\Psi|-1)=9$ and their reversed edges are with capacities 3. Figure 6(d) shows $\mathcal{F}$. $\square$

Remark. CorePExact relies on the core decomposition. For some special patterns such as stars and loops, the core decomposition algorithm in Algorithm 3 can be performed faster by optimizing the steps of computing pattern-degrees and decreasing the vertices' pattern-degrees. For details, please refer to the technical report [27]. For general patterns, we use the state-of-the-art pattern enumeration algorithm [53] for computing the pattern-degrees.

8. EXPERIMENTS

We have performed experiments on ten real graphs [2] (see Table 2). These graphs cover various domains, such as biological networks (e.g., Yeast), collaboration networks (e.g., Ca-HepTh), autonomous system graphs (e.g., As-Caida), bibliographical graphs (e.g., DBLP), web graphs (e.g., UK-2002), citation networks (e.g., Cit-Patents), social networks (e.g., Friendster), etc.

Table 2: Datasets used in our experiments.

Graph	Name	Vertices	Edges
Real small graphs (all algo.)	Yeast	1,116	2,148
	Netscience	1,589	2,742
	As-733	1,486	3,172
	Ca-HepTh	9,877	25,998
	As-Caida	26,475	106,762
Real large graphs (approx. algo.)	DBLP	425,957	1,049,866
	Cit-Patents	3,774,768	16,518,948
	Friendster	20,145,325	106,570,765
	Enwiki-2017	5,409,498	122,008,994
	UK-2002	18,520,486	298,113,762
Synthetic random graphs	SSCA	100,000	3,405,676
	ER	100,000	4,837,534
	R-MAT	100,000	2,571,986

Besides, as shown in Table 2, we have used three synthetic random graphs (SSCA, ER, and R-MAT) generated by GTgraph [3]. These three graphs follow three representative distributions: SSCA

is made by random-sized cliques, ER follows the random distribution, and R-MAT follows the power-law distribution. Note that for SSCA and R-MAT, we set default parameters of their generators; for ER, we set the probability of an edge between any pair of vertices to 0.0005, which is also the chance that an edge exists in the real graph Cap-HepTh. A more detailed analysis of the characteristics of these datasets is in the technical report [27], which we omit here due to lack of space.

We considered two groups of patterns: (1) h-cliques (with $h \in [2, 6]$); and (2) seven other patterns (Figure 7) studied in [70, 46, 45], each of which is associated with an ID (e.g., 4 = diamond).
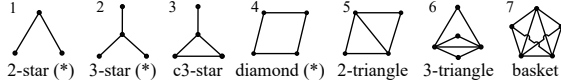
Figure 7: Patterns used in evaluation of PDS.

For CDS problem, we tested 2 exact algorithms (`Exact` and `CoreExact`) and 5 approximation algorithms (`Nucleus` [59], `EMcore` [13], `PeelApp`, `IncApp`, and `CoreApp`). `Nucleus` is applied for decomposing the (k, Ψ)-core where Ψ is an h-clique. For fair comparison, we implement the the faster nucleus decomposition algorithm AND [59] on a single core. We also adapt `EMcore` such that it works in main memory and stops when the $k_{\max}$-core is computed. For PDS problem, we tested both exact algorithms (`PExact`, `CorePExact`) and approximation algorithms (`PeelApp`, `IncApp`, `CoreApp`). For special patterns marked $*$ in Figure 7, we have implemented optimizations discussed in the technical report [27], for all algorithms. Note that `CoreExact`, `IncApp`, `CoreApp`, and `CorePExact` are our core-based approaches. All these solutions are implemented in Java, and executed on a machine having an Intel(R) Xeon(R) 3.40GHz processor, 16 cores, and 125GB of memory, with Ubuntu installed.

8.1 DSD for Edge- and h-Clique-Densities

1 Exact algorithms. Figures 8(a)-(e) show the performance of exact algorithms on five small datasets. (As these solutions cannot finish in a reasonable time on larger datasets, we do not report their results here.) We see that the time costs of all the algorithms increase with the h-clique size. Moreover, `CoreExact` is at least $4.5\times$ and up to four orders of magnitude faster than the existing algorithm `Exact` [4]. This is because `CoreExact` employs the k-clique-cores, or (k, Ψ)-cores, which not only effectively locate the CDS in some smaller subgraphs, but also significantly reduce the flow network sizes in the binary search process. In contrast, the flow network of `Exact` is built on the entire graph in each iteration, and the sizes of the flow networks remain unchanged in all the iterations. Hence, `CoreExact` is faster than `Exact`.

We now investigate how the flow network size (number of nodes) changes in the first six iterations of `CoreExact`, on Ca-HepTh and As-Caida (Figure 9). In the x-axis, "−1" denotes that the flow network is constructed for the entire graph G, instead of a subgraph located by the (k, Ψ)-cores (Section 4.1); "0" means that the flow network is built on the subgraph located by the clique-cores. The (k, Ψ)-cores are indeed effective for locating the CDS, as it greatly prunes vertices and clique instances. The flow networks shrink, as the number of iterations increases. After an iteration of the binary search is completed, a tighter lower bound of ρ_{opt} is obtained, which can then be used to locate the CDS in a smaller subgraph with a larger core number, resulting in a smaller flow network. For example, for the triangle on the Ca-HepTh dataset, over 95% of the nodes in the flow network is pruned after six iterations. As the flow network is smaller, the minimum st-cut can be computed faster,
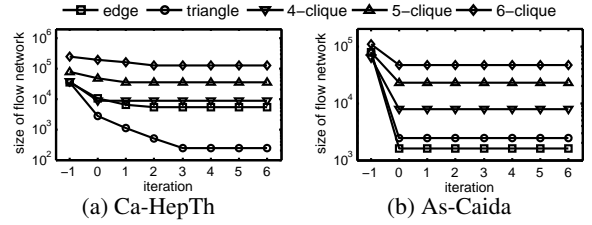

Figure 9: Flow network sizes in `CoreExact`.

thus yielding a better performance. As the clique size (i.e., h) increases, the proportion of cliques instances in the densest subgraph becomes larger, so the degree of pruning gets smaller.
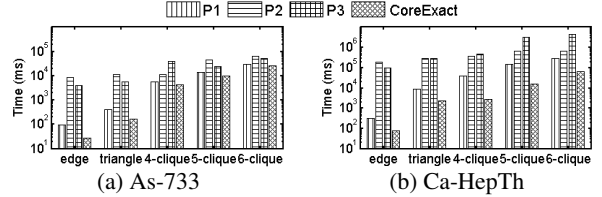

Figure 10: The effect of pruning criteria in `CoreExact`.

Next, we evaluate the individual effect of the three pruning criteria in `CoreExact`. We create three variants of `CoreExact`, namely $P1$, $P2$, and $P3$, which only include *Pruning1*, *Pruning2*, and *Pruning3* respectively, while other steps are the same as those of `CoreExact`. Our experimental results (Figure 10) confirm that each of the pruning strategies makes a contribution to the efficiency of `CoreApp`. Most of the savings come from *Pruning1*; however, while the contribution of other pruning strategies is small on the As-733 and Ca-HepTh, *Pruning2* and *Pruning3* still make a non-trivial contribution on Ca-HepTh.

Finally, we examine the percentage of time cost of core decomposition in `CoreExact`. As shown in Table 3, the percentage is small and decreases with the h-clique size. Besides, cores are effective for locating the CDS in some small subgraphs. Thus, `CoreExact` achieves high efficiency, while incurring negligible overhead from core decomposition.

Table 3: % of time cost of core decomposition.

Dataset	edge	triangle	4-clique	5-clique	6-clique
As-733	57.14%	8.28%	0.31%	0.09%	0.04%
Ca-HepTh	69.74%	6.01%	2.32%	0.87%	0.65%

2 Approximation algorithms. We next report the efficiency results of approximation solutions on the five largest datasets. From Figures 8(f)-(j), we observe that core-based approximation algorithms (`IncApp` and `CoreApp`) are consistently faster than `Nucleus` and `PeelApp` [5]. This implies that for decomposing cores, our algorithm (Algorithm 3) which is almost the same as `IncApp` is faster than the nucleus decomposition algorithm. The average running time of `IncApp` is only 90% of that of `PeelApp`. Both algorithms iteratively remove vertices from the graph G. Particularly, `PeelApp` computes the density after removing each vertex, and only stops after G has no more vertices. However, `IncApp` does not compute the density, and stops after the $(k_{\max}, \Psi)$-core is discovered. `CoreApp` performs the best, as it finds the $(k_{\max}, \Psi)$-core in a top-down manner, and skips the computation of cores with smaller clique-core numbers. In our experiments, `CoreApp` is up to three and two orders of magnitude faster than `Nucleus` and `PeelApp` respectively.

---

[4]For exact algorithms, bars touching the upper boundaries mean that the corresponding algorithms cannot finish within 5 days.

[5]For approximation algorithms, bars touching the upper boundaries mean that the corresponding algorithms cannot finish within 2 days.
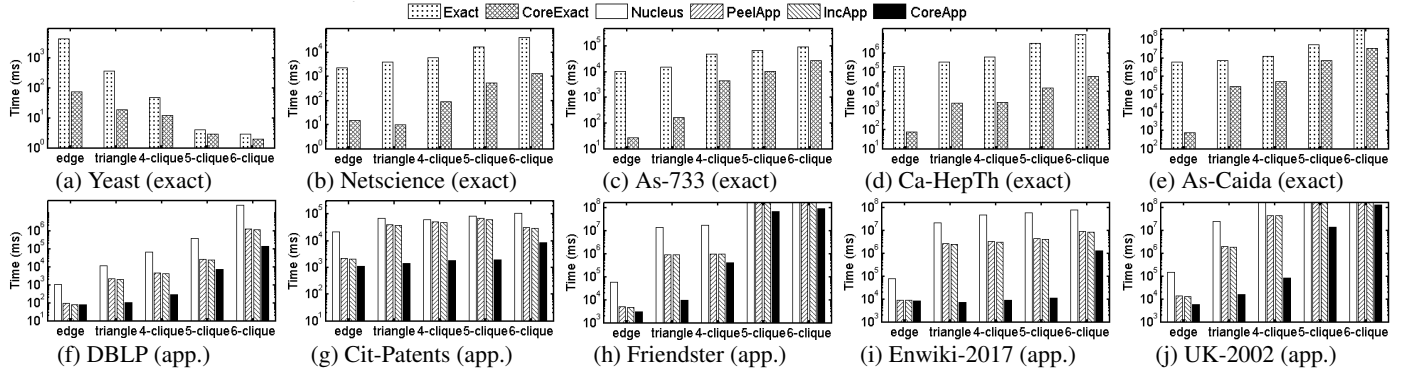
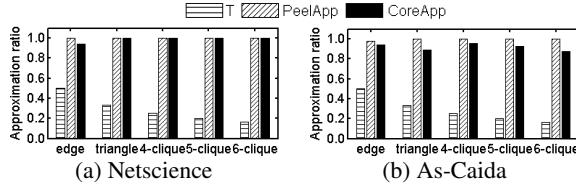Figure 8: Efficiency of exact and approximation CDS algorithms.
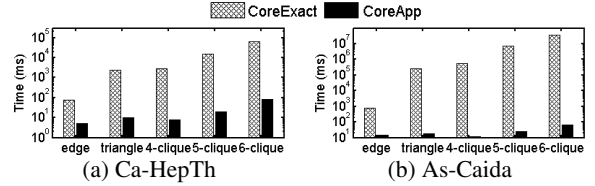

Figure 11: Approximation ratio.

Figure 12: CoreExact and CoreApp.

As the clique size (h) increases, the speedup of CoreApp over PeelApp decreases, because the proportion of clique instances in the densest subgraph becomes larger, increasing the time cost of computing (k_{max}, Ψ)-core. Meanwhile, the running time generally grows as the clique size (h) increases, except for the Cit-Patents dataset. This is because on Cit-Patents, the numbers of 5-cliques and 6-cliques are less than the number of 4-cliques. In addition, we compare CoreApp with EMcore for computing approximate EDS's on five largest datasets. As reported in Table 4, EMcore is slower than CoreApp, because it differs with CoreApp on four aspects as discussed in Section 6.2.

Table 4: Efficiency of EMcore and CoreApp (seconds).

Algo.	DBLP	CitPatents	FriendSter	Enwiki-2017	UK-2002
EMcore	0.091	1.132	3.143	8.543	7.543
CoreApp	0.077	1.021	2.986	8.139	5.825

We next report the theoretical ratio T (i.e., $\frac{1}{|V_\Psi|}$) and actual approximation ratios R of approximation methods. Since Nucleus, IncApp and CoreApp return the same (k_{max}, Ψ)-core, their R values are the same, so we only show results for CoreApp. As shown in Figure 11, R is often larger than T. Although CoreApp is slightly worse than PeelApp on 6/10 instances (the average ratio of CoreApp is 0.956 times that of PeelApp), they have the same theoretical guarantee and their actual ratios are close to 1.0 in most cases, so CoreApp produces high-quality results in practice.

In addition, we compare the efficiency of core-based exact and approximation approaches on two datasets. As shown in Figure 12, CoreApp is much faster than CoreExact. The reason is that CoreExact relies on not only core decomposition, but also computing the minimum st-cut from flow networks using binary search, whereas CoreApp just computes the (k_{max}, Ψ)-core directly.

Remark. For small-to-moderate-sized graphs (e.g., Ca-HepTh), CoreExact is the best choice, as it computes an exact result in a reasonable time. For larger graphs (e.g., UK-2002), CoreApp is a much better option since it achieves high accuracy and efficiency.

3 Random graphs. As depicted in Figures 13 and 14, for SSCA and R-MAT, the performance of our proposed solution is generally satisfactory. For example, the running time of CoreApp is 20 (resp., 201) times faster than PeelApp in SSCA (resp., R-MAT)

when Ψ is the triangle. For ER, the degree values of vertices are almost the same, and the k_{max}-core contains 96.8% of the vertices in the graph. This affects the pruning effectiveness of CoreApp, rendering a lower performance gain. All in all, our core-based algorithms favor real-world graphs.

4 Densities of CDS's. We next show the clique-densities of CDS's for different h-cliques ($h \geq 3$). Specifically, for each dataset, we first use CoreExact to compute its exact CDS's for different cliques, then compute the h-clique-densities of its EDS, and finally report the h-clique-densities of its EDS and CDS's in Table 5. Due to the space limitation, we only show the results on four small datasets (where S-DBLP is a sub-graph of the DBLP dataset used in Section 8.2). We remark that for Yeast dataset, the EDS does not contain any 4, 5, 6-clique, so its h-clique-density is 0.0 ($h \geq 4$). As we can see, for S-DBLP and Netscience, their CDS's are exactly the same as EDS. In fact, they are the maximal clique in the graph, which confirms the conclusion that CDS's can be used for identifying large near-cliques [65]. For Yeast and As-733, the clique-density values of CDS's are higher than those on the EDS.

8.2 DSD for Pattern-Densities

Next, we present the results for general patterns in Figure 7. For lack of space, we only report results on a subset of datasets. In addition, we perform case studies on real datasets for these patterns.

1 Exact algorithms. In Figure 15, we present the efficiency results of exact algorithms on two small datasets As-733 and Ca-HepTh. The bars touching the top of the figures mean that the corresponding algorithms cannot find densest subgraphs within 3 days, at which point we time them out. We can see that CorePExact is up to four orders of magnitude faster than PExact. For different patterns, their running times vary, because the number of pattern instances in the underlying graph for each pattern can be very different. For any two patterns Ψ_1 and Ψ_2 which are not "special patterns" (e.g., star and loop), we observe that if $|V_{\Psi_1}| = |V_{\Psi_2}|$ and $\Psi_1 \subseteq \Psi_2$, then it takes longer to find the densest subgraph w.r.t. Ψ_1 than w.r.t. Ψ_2. This is because the number of pattern instances of Ψ_1 is more than that of Ψ_2. For example, *c3-star* is a subgraph of *2-triangle* (with 4 vertices) and it takes more time to find the densest subgraph w.r.t. *c3-star* than *2-triangle*.
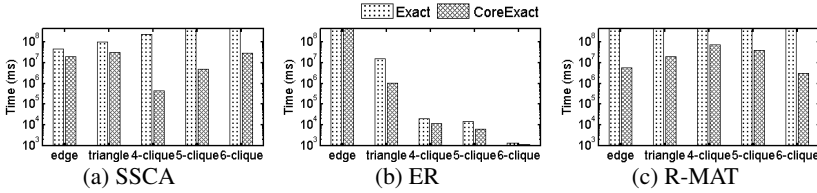
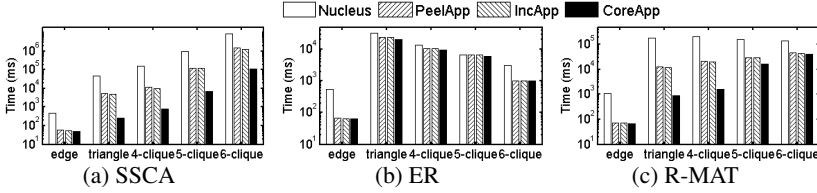**Figure 13: Efficiency of exact CDS algorithms on random graphs.**

Figure 15: Efficiency of exact PDS algorithms.

Figure 14: Efficiency of approximation CDS algorithms on random graphs.

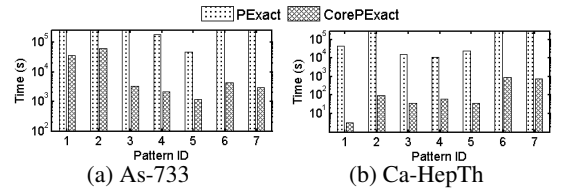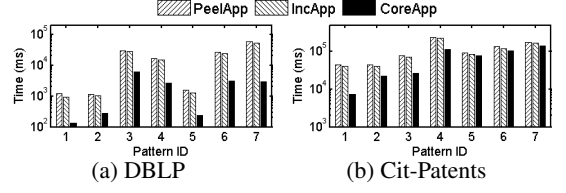**Figure 16: Efficiency of approx. PDS algorithms.**

Table 5: The edge-densities and clique-densities (pattern-densities) of CDS's (PDS's).

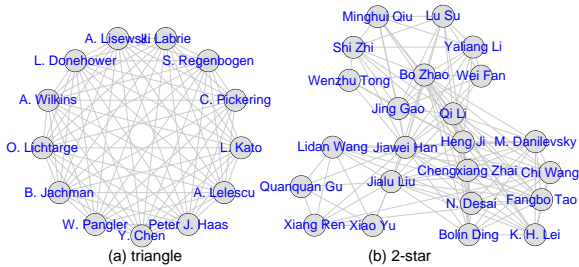| Dataset	edge	triangle		4-clique		5-clique		6-clique		2-star		diamond	
	ρ_{opt}	ρ_{opt}	$\rho(\text{EDS},\Psi)$	ρ_{opt}	$\rho(\text{EDS},\Psi)$	ρ_{opt}	$\rho(\text{EDS},\Psi)$	ρ_{opt}	$\rho(\text{EDS},\Psi)$	ρ_{opt}	$\rho(\text{EDS},\Psi)$	ρ_{opt}	$\rho(\text{EDS},\Psi)$
S-DBLP	6	22	22	55	55	99	99	132	132	73.5	66	165	165
Yeast	3.13	2.11	0.467	0.67	0.0	0.0	0.0	0.0	0.0	111.3	18.13	20	19.2
Netscience	9.50	57.25	57.25	242.3	242.3	775.2	775.2	1938	1938	171	171	726.8	726.8
As-733	8.19	31.43	31.35	68.67	67.94	92.78	90.23	79.37	75.13	826.3	153.8	3376	437.7

Figure 17: The densest subgraphs found in DBLP network, based on triangle and 2-star patterns.

2 Approximation algorithms. As shown in Figure 16, the running time of an approximation algorithm increases with the graph size in general. This is because computing the cores is more expensive for a larger graph. Again, CoreApp performs the fastest, and it is up to two orders of magnitude faster than PeelApp. For special patterns (star and diamond), we use optimized algorithms (details are in [27]) for core decomposition. Hence, they need less time cost than other more complicated patterns (e.g., 2-triangle).

3 Case studies. We use two real graphs, namely *S-DBLP* and *Yeast*. S-DBLP ($|V|$=478, $|E|$=1,086) is a sub-graph of the DBLP dataset. It is the co-authorship network of authors who published at least two DB/DM papers between 2013 and 2015. We consider two 3-vertex patterns, i.e., *triangle* and *2-star* (Figure 7). We use the exact algorithm to compute their PDS's, as depicted in Figure 17. In a *triangle* pattern, every pair of vertices is connected, so the PDS tends to be a near-clique [65]. The researchers involved in this PDS possess a close collaboration relationship: any two researchers have published papers together. The PDS for *2-star* is quite different from that of *triangle*. Particularly, researchers in the "central" part of the PDS formed by *2-star* tend to be group directors or senior researchers (e.g., Profs. Jiawei Han and Chengxiang Zhai), who are linked to their former students or postdocs. For this PDS, over half of the researchers worked in Prof. Han's lab before. Similarly, for Yeast, different PDS's can capture different semantics [27].

4 Densities of PDS's. In this experiment, we analyze the pattern-densities of PDS's for different patterns. Again, for each dataset,

we first compute its exact EDS and PDS's for all patterns, and then report the pattern-densities of its EDS and PDS's in Table 5. Due to the space limitation, we only show results of 2-star and diamond. As we can observe, for most of the datasets, the pattern-density values of PDS's are higher than those on the EDS.

9. CONCLUSIONS

The densest subgraph discovery (DSD) problem is fundamental to many graph applications. In this paper, we develop new algorithms to discover edge- and h-clique-based densest subgraphs, which are well studied in the literature. Our main observation is that densest subgraphs can be derived efficiently from k-cores. We extend k-core to (k, Ψ)-core by incorporating an h-clique Ψ. Based on (k, Ψ)-cores, we develop core-based exact and approximation solutions to the DSD problem. Moreover, we generalize the edge- and h-clique-density to pattern-density and show that our solutions can be easily adapted for finding pattern-density-based densest subgraphs. Extensive experiments show that our exact (resp., approximation) "core-based solutions" outperform existing algorithms by up to four orders (resp., two orders) of magnitude.

In the future, we will attempt to derive even tighter bounds for densities of (k, Ψ)-cores. We will also extend our core-based algorithms for finding densest subgraphs with size constraints. Another interesting research direction is to exploit our core-based techniques to speed up the randomized approximation algorithm in [49].

Acknowledgments

10. REFERENCES

[1] https://en.wikipedia.org/wiki/Clique_(graph_theory).

[2] R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan. Improved algorithms for bipartite network flow. *SIAM Journal on Computing*, 23(5):906–933, 1994.

[3] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 25–37, 2009.

[4] Y. Asahiro, R. Hassin, and K. Iwama. Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, 121(1):15–26, 2002.

[5] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

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

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

[8] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an o (n 1/4) approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010.

[9] E. T. Charalampos. *Mathematical and Algorithmic Analysis of Network and Biological Data*. PhD thesis, Carnegie Mellon University, 2013. 2.1.

[10] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 84–95. Springer, 2000.

[11] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *TKDE*, 24(7):1216–1230, 2012.

[12] Y. Chen, Y. Fang, R. Cheng, Y. Li, X. Chen, and J. Zhang. Exploring communities in large profiled graphs. *IEEE Transactions on Knowledge and Data Engineering*, 31(8):1624–1629, 2019.

[13] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

[14] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[15] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16, 2008.

[16] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *SIGMOD*, pages 277–288. ACM, 2013.

[17] M. Danisch, O. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs. In *WWW*, pages 589–598, 2018.

[18] M. Danisch, T.-H. H. Chan, and M. Sozio. Large scale density-friendly graph decomposition via convex programming. In *Proceedings of the 26th International Conference on World Wide Web*, pages 233–242, 2017.

[19] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, pages 300–310, 2015.

[20] Y. Fang, R. Cheng, Y. Chen, S. Luo, and J. Hu. Effective and efficient attributed community search. *The VLDB Journal*, 26(6):803–828, 2017.

[21] Y. Fang, R. Cheng, G. Cong, N. Mamoulis, and Y. Li. On spatial pattern matching. In *IEEE International Conference on Data Engineering (ICDE)*, pages 293–304. IEEE, 2018.

[22] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

[23] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

[24] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. *The VLDB Journal*, 2019.

[25] Y. Fang, Z. Wang, R. Cheng, X. Li, S. Luo, J. Hu, and X. Chen. On spatial-aware community search. *IEEE Transactions on Knowledge and Data Engineering*, 31(4):783–798, 2018.

[26] Y. Fang, Z. Wang, R. Cheng, H. Wang, and J. Hu. Effective and efficient community search over large directed graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2018.

[27] Y. Fang, K. Yu, R. Cheng, L. V. S. Lakshmanan, and X. Lin. Efficient algorithms for densest subgraph discovery (technical report). arXiv, 2019. https://arxiv.org/abs/1906.00341.

[28] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

[29] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

[30] A. Gionis, F. Junqueira, V. Leroy, M. Serafini, and I. Weber. Piggybacking on social networks. *PVLDB*, 6(6):409–420, 2013.

[31] A. Gionis and C. E. Tsourakakis. Dense subgraph discovery: Kdd 2015 tutorial. In *SIGKDD*, pages 2313–2314, NY, USA, 2015. ACM.

[32] A. V. Goldberg. *Finding a maximum density subgraph*. UC Berkeley, 1984.

[33] G. T. Heineman, G. Pollice, and S. Selkow. Chapter 8: Network flow algorithms. *Algorithms in a Nutshell*, pages 226–250, 2008.

[34] J. Hu, R. Cheng, K. C.-C. Chang, A. Sankar, Y. Fang, and B. Y. Lam. Discovering maximal motif cliques in large heterogeneous information networks. In *IEEE International Conference on Data Engineering (ICDE)*, pages 746–757. IEEE, 2019.

[35] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang. Querying minimal steiner maximum-connected subgraphs in large graphs. In *International on Conference on Information and Knowledge Management (CIKM)*, pages 1241–1250. ACM, 2016.

[36] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang. On minimal steiner maximum-connected subgraph queries. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2455–2469, 2017.

[37] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322. ACM, 2014.

[38] X. Huang and L. V. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.

[39] X. Huang, W. Lu, and L. V. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *Proceedings of the 2016 International Conference on Management of Data*, pages 77–90. ACM, 2016.

[40] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *WWW*, pages 495–505, 2015.

[41] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *SIGMOD*, pages 813–826. ACM, 2009.

[42] D. B. Johnson. Parallel algorithms for minimum cuts and maximum flows in planar networks. *Journal of the ACM (JACM)*, 34(4):950–967, 1987.

[43] R. Kannan and V. Vinay. *Analyzing the structure of large graphs*. Rheinische Friedrich-Wilhelms-Universität Bonn, 1999.

[44] S. Khuller and B. Saha. On finding dense subgraphs. *Automata, Languages and Programming*, pages 597–608, 2009.

[45] L. Lai, L. Qin, X. Lin, Y. Zhang, L. Chang, and S. Yang. Scalable distributed subgraph enumeration. *PVLDB*, 10(3):217–228, 2016.

[46] J. Leskovec, A. Singh, and J. Kleinberg. Patterns of influence in a recommendation network. In *PAKDD*, pages 380–389. Springer, 2006.

[47] L. Lü, T. Zhou, Q.-M. Zhang, and H. E. Stanley. The h-index of a network node and its relation to degree and coreness. *Nature communications*, 7:10168, 2016.

[48] A. Mandal and M. Al Hasan. A distributed k-core decomposition algorithm on spark. In *International Conference on Big Data*, pages 976–981. IEEE, 2017.

[49] M. Mitzenmacher, J. Pachocki, R. Peng, C. Tsourakakis, and S. C. Xu. Scalable large near-clique detection in large-scale networks via sampling. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 815–824. ACM, 2015.

[50] A. Montresor, F. De Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE Transactions on parallel and distributed systems*, 24(2):288–300, 2013.

[51] Y. Peng, Y. Zhang, W. Zhang, X. Lin, and L. Qin. Efficient probabilistic k-core computation on uncertain graphs. In *International Conference on Data Engineering (ICDE)*, pages 1192–1203. IEEE, 2018.

[52] T. L. Pham, I. Lavallee, M. Bui, and S. H. Do. A distributed algorithm for the maximum flow problem. In *International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 131–138. IEEE, 2005.

[53] M. Qiao, H. Zhang, and H. Cheng. Subgraph matching: on compression and computation. *PVLDB*, 11(2):176–188, 2017.

[54] L. Qin, R.-H. Li, L. Chang, and C. Zhang. Locally densest subgraph discovery. In *KDD*, pages 965–974. ACM, 2015.

[55] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *RECOMB*, volume 6044, pages 456–472, 2010.

[56] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu. The ubiquity of large graphs and surprising challenges of graph processing. *PVLDB*, 11(4):420–431, 2017.

[57] R. Samusevich, M. Danisch, and M. Sozio. Local triangle-densest subgraphs. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 33–40. IEEE, 2016.

[58] A. E. Sariyüce and A. Pinar. Fast hierarchy construction for dense subgraphs. *PVLDB*, 10(3):97–108, 2016.

[59] A. E. Sariyüce, C. Seshadhri, and A. Pinar. Local algorithms for hierarchical dense subgraph discovery. *PVLDB*,

[60] A. E. Sariyüce, C. Seshadhri, A. Pinar, and U. V. Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, pages 927–937, 2015.

[61] A. E. Sariyüce, C. Seshadhri, A. Pinar, and Ü. V. Çatalyürek. Nucleus decompositions for identifying hierarchy of dense subgraphs. *ACM Transactions on the Web (TWEB)*, 11(3):16, 2017.

[62] S. B. Seidman. Network structure and minimum degree. *Social networks*, 1983.

[63] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[64] N. Tatti and A. Gionis. Density-friendly graph decomposition. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1089–1099, 2015.

[65] C. Tsourakakis. The k-clique densest subgraph problem. In *WWW*, pages 1122–1132, 2015.

[66] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112. ACM, 2013.

[67] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *IEEE International Conference on Data Engineering (ICDE)*, pages 233–244. IEEE, 2018.

[68] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Vertex priority based butterfly counting for large-scale bipartite networks. *PVLDB*, 12(10):1139–1152, 2019.

[69] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: on free rider effect and its elimination. *PVLDB*, 8(7):798–809, 2015.

[70] S. Wuchty, Z. N. Oltvai, and A.-L. Barabási. Evolutionary conservation of motif constituents within the yeast protein interaction network. *Nature Genetics*, 35:176–179, 2003.

[71] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE*, pages 1049–1060, 2012.

[72] F. Zhao and A. K. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB*, 6(2):85–96, 2012.