Efficient k-Clique Densest Subgraph Discovery: Towards Bridging Practice and Theory

Yingli Zhou*
The Chinese University of Hong Kong, Shenzhen, China
yinglizhou@link.cuhk.edu.cn

Qingshuo Guo*
The Chinese University of Hong Kong, Shenzhen, China
qingshuoguo@link.cuhk.edu.cn

Yixiang Fang†
The Chinese University of Hong Kong, Shenzhen, China
fangyixiang@cuhk.edu.cn

ABSTRACT

Densest subgraph discovery (DSD) is a fundamental topic in graph mining. It has been studied for decades, and is widely used in various areas, including network science, biological analysis, and graph databases. As a typical problem of DSD, the k-clique densest subgraph (CDS) problem aims to detect a subgraph from a graph, such that the number of k-cliques over the number of its vertices is maximized. While the CDS problem has received plenty of attention in the literature, existing CDS algorithms that perform best in practice often have weaker theoretical guarantees, while those with the stronger theoretical assurances tend to perform worse in practice. Besides, all the existing CDS algorithms struggle with graphs with high degeneracy values, a characteristic commonly found in real-world graphs. To bridge the huge gap between practice and theory, in this paper, we first introduce a novel graph reduction technique, which locates the CDS into a very small subgraph, with non-trivial theoretical guarantees. We further propose a new efficient approximation algorithm by employing the state-of-the-art k-clique counting algorithm, which shares all the advantages of existing algorithms, achieving both strong practical efficiency and theoretical guarantees. Extensive experiments on 12 real-world large graphs demonstrate the high efficiency of our CDS algorithm. Particularly, our algorithm is up to four orders of magnitude faster than the state-of-the-art algorithm while maintaining the same accuracy guarantees and requiring much less memory.

PVLDB Reference Format:
Yingli Zhou, Qingshuo Guo, and Yixiang Fang. Efficient k-Clique Densest Subgraph Discovery: Towards Bridging Practice and Theory. PVLDB, 18(10): 3490 - 3503, 2025.
doi:10.14778/3748191.3748210

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/forxenn/ccas/tree/main.

1 INTRODUCTION

As a fundamental problem in graph mining, Densest Subgraph Discovery (DSD) has been studied for decades in the literature [3, 6, 13, 28, 54, 56, 63, 71, 73, 77, 91, 92], and widely used in many areas, such as biology [19, 30, 37, 66], network science [16, 24], and social network [4, 17, 32, 33, 42, 45, 87, 88]. The classic DSD problem [34] aims to find the subgraph with maximum edge-density, or the number of edges over the number of vertices within the subgraph, which is often called the edge-density-based densest subgraph (EDS). Recently, this problem has been generalized as the k-*Clique Densest Subgraph* (CDS) problem [27, 28, 36, 46, 50, 57, 68, 71, 76, 91], aiming to find the subgraph with the highest k-clique-density, which is the defined as the number of k-cliques over the number of vertices within it. Note that since an edge can be considered as a 2-clique, the EDS problem is a special case of the CDS problem with k=2. Similarly, the triangle-density densest subgraph problem [68] corresponds to the CDS problem with k=3. Among the three problems, CDS is particularly well-suited for networks where modeling higher-order and more complex relations is required, such as protein-protein and gene-gene interaction networks. For example, in Figure 1, the 3-clique density of the subgraph induced by $\{v_4, \cdots, v_8\}$ (in shaded region) is $\frac{4}{5}$, since it includes four 3-cliques and five vertices, and it is actually the 3-clique densest subgraph since there is no other subgraph with a higher 3-clique-density.

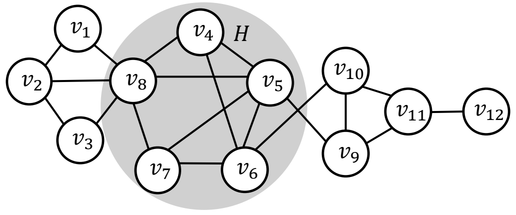


Figure 1: An example of the k-clique densest subgraph.

The CDS solutions have been widely used in many real-world application [27, 28, 36, 46, 50, 57, 71, 76]. For example, the CDS can be used to detect "near-cliques", and when k gets large, it is more likely to capture useful "near-cliques", which can help discover biologically relevant functional groups [19, 41, 76, 77], find social communities [4, 15, 77], and detect anomalies [31, 72, 85]. In many of these applications, finding a "near-clique" is very important since a "near-clique" can be considered a clique in the forming stage or one with missing edges due to data corruption. Besides, finding CDS is very useful in many graph data mining applications when k is relatively small. Specifically, it can help identify research communities in the DBLP network [28, 76, 77], detect subnetworks with a specific function in the biology network [28] and clusters in senators' networks on US bill voting [28, 76], and discover compact dense subgraphs from e-commerce and social networks [68].

• **Prior works.** While the CDS is very useful, it is computationally costly in both time and space. In the literature, various exact and approximation algorithms have been developed to solve the

*The first two authors contributed equally to this research.
†Yixiang Fang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 10 ISSN 2150-8097.
doi:10.14778/3748191.3748210

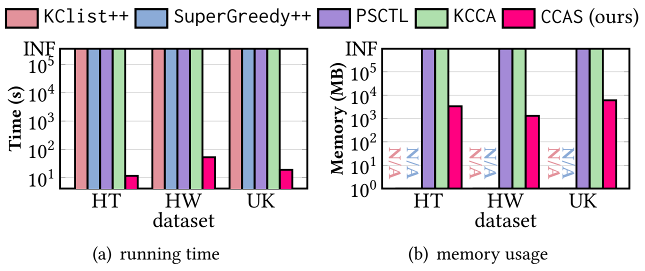


Figure 2: Time and memory costs of CDS algorithms.

CDS problem. The exact algorithms are often based on maximum flow [34, 57, 76], k-core [28], and convex programming [23, 36, 71]. The approximation algorithms are based on peeling [11, 13, 14, 76], k-core [28, 51], and convex programming [14, 23, 36, 71, 84, 91].

The five representative algorithms are `KClist++` [71], `SCTL` [36], `SuperGreedy++` [14], `KCCA` [91], and `PSCTL` [84]. Generally, these algorithms update the vertex weights via T iterations, and then the vertices with larger vertex weights are more likely to be included in the CDS. The first three algorithms are based on *clique enumeration* to update the vertex weights, while the remaining two are based on *clique counting*.

(1) Clique enumeration-based algorithms. In `KClist++` and `SCTL`, they first assign a weight 0 to every vertex v in the graph, initially. Then, they enumerate all the k-cliques in the graph for T iterations, and for each k-clique, they increase the minimum vertex weight in it by one. For `SuperGreedy++` algorithm, it iteratively removes the vertices via T iterations, and in each iteration, it removes the vertex with the smallest weight, and then updates the remaining vertex weights for subsequent computation. All of them are based on the *clique enumeration* to update the vertex weights. However, the numbers of k-cliques in real-world graphs increase dramatically even for relatively small values of k, as shown in [40, 82, 91]. For instance, on the DBLP dataset, there are over 10^{18} 15-cliques. Hence, enumerating almost all k-cliques is extremely costly, making the above three algorithms unscalable for processing large-scale graphs.

(2) Clique counting-based algorithms. To alleviate the above issues, Zhou et al. [91] first proposed a *clique counting*-based CDS algorithm (`KCCA`). Specifically, in each iteration, `KCCA` updates the weight of vertex with minimum weight in the graph, by calculating the number of k-cliques containing it. `KCCA` has achieved remarkable performance improvement compared to the *clique enumeration*-based algorithms. Recently, another clique counting-based algorithm, `PSCTL` [84], follows the same framework as `SCTL`, but utilizes the clique counting to update the vertex weights. As shown in our experiments, `KCCA` and `PSCTL` achieve comparable performance in terms of efficiency, memory usage, and accuracy, and both of them are extremely faster than the *clique enumeration*-based algorithms.

• **Motivation.** While the existing works above have achieved some positive progress, they still have some limitations, which can be summarized from two aspects:

(1) Practical performance. Since both `KCCA` and `PSCTL` employ local k-clique counting from `PIVOTER` [69], they inherit `PIVOTER`'s limitations. A key limitation of `PIVOTER` is that its time and space costs grow exponentially with the degeneracy value of the graph. As shown in Figure 2, they are intractable to handle graphs with high degeneracy values, where the degeneracy values of HT, HW, and UK datasets are 561, 2208, and 943 respectively. For instance, on the UK dataset (with 18 M vertices and 261 M edges), both `KCCA` and `PSCTL` cost over 1 TB of memory to build SCT, leading to the out-of-memory (OOM) issue. In addition, other algorithms encounter time limit issues and fail to complete within three days.

(2) Theory-practice gap. To obtain a $(1-\epsilon)$-approximation ratio solution, both `KCCA` and `PSCTL` need $\Omega\left(\frac{\Delta|\Psi_k(G)|}{\epsilon^2}\right)$ iterations theoretically, where Δ denotes the maximum number of k-cliques that share a vertex in G, and $|\Psi_k(G)|$denotes the number of k-cliques in G. While the `SuperGreedy++` only takes $\Omega\left(\frac{\Delta \log|\Psi_k(G)|}{\rho_k^*(G)\cdot\epsilon^2}\right)$ iterations, its practical performance is poor. Hence, it is essential to bridge the gap between practical performance and theoretical aspects: *Can we design an efficient CDS algorithm that shares the advantages of all the existing algorithms for processing real-world large graphs (e.g., graphs with high degeneracy values)?* In this paper, we show that it is possible to achieve this.

• **Our technical contributions.** To make the CDS algorithm more practical, we propose an elegant graph reduction technique based on the inclusion-exclusion principle, which first reduces our search space into a very smaller region, allowing the algorithm to obtain the optimal solution quickly. It plays a crucial role in making our algorithm handle the graph with high degeneracy values. For example, on the UK dataset with 18 million vertices, when k=7, it is reduced to a subgraph with just 1,665 vertices using our graph reduction technique, achieving a 10000× reduction.

Besides, from a theoretical perspective, we develop a simple yet efficient $(1-\epsilon)$-approximation algorithm, called k-Clique Counting And SuperGreedy++-based (`CCAS`) algorithm, with $0 < \epsilon < 1$, inspired by `SuperGreedy++` [14] which works in an iterative manner. Specifically, in each iteration of `SuperGreedy++`, it first assigns a weight to every vertex v in the graph, which is initialized to the number of k-cliques containing v, and then removes the vertex v with the smallest weight from the graph. Afterwards, for each k-clique containing v, it updates the weights of other vertices in this clique. We observe that in each iteration of `SuperGreedy++`, the change of vertex weight, can actually be calculated by using the k-cliques counting, which is often much faster than k-clique enumeration. Moreover, we bridge the connection between the vertex weights in `CCAS` and those in the convex-programming formulation of the CDS problem. We further design an early stop condition to obtain a solution with $(1-\epsilon)$-approximation ratio guarantee, which only needs a few iterations, rather than $\Omega\left(\frac{\Delta \log|\Psi_k(G)|}{\rho_k^*(G)\cdot\epsilon^2}\right)$ iterations which is needed by the original `SuperGreedy++` algorithm.

In addition, based on the nested relationship among the CDS's for different k-values, we propose an efficient algorithm that finds the CDS's for all the possible k values with non-trivial theoretical guarantees, and its running time cost is nearly the same as that of discovering the CDS for a single k value.

Extensive experimental evaluations on 12 real-world large graphs show that `CCAS` achieves both higher efficiency and scalability than the existing CDS algorithms on all datasets. Particularly, it is up to four orders of magnitude faster than the SOTA algorithms. Besides, it produces a near-optimal solution on all datasets.

In summary, our principal contributions are as follows.

- To locate the CDS into a small subgraph, we propose an effective graph reduction technique with a non-trivial theoretical guarantee.
- We develop an efficient CDS algorithm by employing the SOTA k-clique counting algorithm, achieving a better theoretical guarantee and practical performance.
- We conduct experiments on 12 real-world large graphs to demonstrate the efficiency and scalability of our algorithm.

Outline. We introduce the CDS problem in Section 2. Section 3 analyzes the limitations of state-of-the-art (SOTA) algorithms. We introduce our graph reduction method in Section 4, and present our `CCAS` algorithm in Section 5. We further extend `CCAS` for processing all k values in Section 6. The experimental results are reported in Section 7. We review the related works in Section 8 and conclude in Section 9. For lack of space, all detailed proofs in this paper are included in our technical report [64] and we only show the proof sketches for some key lemmas and theories.

2 PROBLEM DEFINITION

Table 1: Notations and meanings.

Notation	Meaning
$G = (V, E)$	a graph with vertex set V and edge set E
$G[S]$	the subgraph of G induced by vertices in S
$\Psi_k(G)$	the set of k-cliques in G
$\Psi_k(v, G)$	the set of k-cliques containing v in G
$\mathcal{D}_k(G)$	the k-clique densest subgraph of G
$\rho_k(H)$	the k-clique density of subgraph H
$\mathcal{S}_k(G)$	an approximate k-clique densest subgraph of G
$l(v)$	the weight of vertex v

We consider an unweighted and undirected graph G=(V, E), where V and E are the sets of vertices and edges in the graph, respectively. Denote by $n = |V|$ and $m = |E|$ ($n \leq m$) the numbers of vertices and edges in G respectively. Given a vertex set S, we use $G[S] = (S, E(S))$ to denote the subgraph of G induced by S, where $E(S)$= $\{(u, v) \in E \mid u, v \in S\}$ denotes the set of edges between vertices in S. For a given graph H, we also denote its sets of vertices and edges by $V(H)$ and $E(H)$, respectively.

A k-clique is a complete graph with a set C of k vertices where there is an edge between every pair of vertices. In this case without ambiguity, we simply refer to a k-clique by its set of vertices. We use $\Psi_k(G)$ to represent the set of k-cliques in G. Denote by $\Psi_k(G)$ = $\{C \subseteq V \mid C$ is a k-clique of $G\}$. For each vertex $v \in G$, we use $\Psi_k(v, G)$ to denote the set of k-cliques containing v in the graph G ($k \geq 3$). We define the k-clique engagement of v in G as the number of k-cliques containing v in G, i.e., $|\Psi_k(v, G)|$. We summarize the frequently used notations in Table 1.

We now formally present the definition of k-clique density.

Definition 1 (k-clique density [76]). *Given a subgraph H of a graph G and a positive integer k, the k-clique density of H, denoted by $\rho_k(H)$, is the average number of k-cliques per vertex in H, i.e.,*

$$\rho_k(H) = \frac{|\Psi_k(H)|}{|V(H)|}. \tag{1}$$

Definition 2 (k-clique densest subgraph [28, 36, 57, 71, 76]). *Given a graph G and a positive integer k, a subgraph H of G is the k-clique densest subgraph (k-CDS), denoted by $\mathcal{D}_k(G)$, if H has the maximum k-clique density among all subgraphs of G. We use $\rho_k^*(G)$ to denote the k-clique density of $\mathcal{D}_k(G)$.*

When k=2, $\mathcal{D}_2(G)$ is the classic edge-density-based densest subgraph (EDS) [34] that maximizes the edge-density, i.e., the average number of edges per vertex within the subgraph. In this work, we mainly focus on the cases when $k \geq 3$, and study the $(1-\epsilon)$-approximation solution ($0 < \epsilon < 1$). Since as shown in existing works [36, 71], approximation algorithms are not only significantly faster than exact algorithms, but also produce solutions with k-clique densities that are very close to those of the exact solutions.

Here, the approximation ratio of an algorithm is defined as the k-clique density of its solution over that of the CDS, which is at most 1.0. A $(1-\epsilon)$-approximation solution means that the k-clique density of the returned subgraph is at least $(1-\epsilon) \cdot \rho_k^*(G)$. Next, we formally present the definition of CDS problem.

Problem 1 (CDS problem [27, 28, 36, 46, 50, 57, 68, 71, 76, 84, 91]). *Given a graph G and an integer $k \geq 3$, find the subgraph of G, denoted by $\mathcal{D}_k(G)$, which has the highest k-clique density.*

Example 1. *In the graph G of Figure 1, there are 7 3-cliques, i.e., $C_1 = \{v_1, v_2, v_8\}$, $C_2 = \{v_2, v_3, v_8\}, \cdots, C_7 = \{v_9, v_{10}, v_{11}\}$. The subgraph H of $\{v_4, v_5, v_6, v_7, v_8, v_9\}$ contains four 3-cliques, so its 3-clique density is $\frac{4}{5}$. Clearly, H is the 3-clique densest subgraph since no other subgraph has a higher 3-clique density.*

3 ANALYSIS OF SOTA PRACTICAL AND THEORETIC ALGORITHMS

In this section, we first review the two SOTA CDS algorithms, i.e., `KCCA` and `SuperGreedy++`, that achieve the best practical and theoretical results respectively, and then discuss their limitations.

3.1 Review of `KCCA`

To our best knowledge, the CDS that achieves the best practical performance is `KCCA` [91], which is based on the interesting fact that k-clique counting is remarkably faster than clique enumeration. `KCCA` follows the `Frank-Wolfe` framework in the existing CDS algorithms [23, 71] and only needs to use k-clique counting to update the vertices' weights, instead of enumerating the k-cliques. Specifically, in each iteration, it updates the weight of the vertex by counting the number of k-cliques containing it, and also proposes a simultaneous vertex weight update strategy to speed up the convergence. Even within limited iterations, it can yield near-optimal approximation results [91]. Besides, recently, we have noticed that Ye et al. [84] proposed another k-clique counting-based algorithm `PSCTL`, which improves `SCTL` [36] by using k-clique counting, and achieves comparable performance with `KCCA`, as shown in our experiments. Since the above algorithms are based on the `Frank-Wolfe`, they need at least $\Omega\left(\frac{1}{\epsilon^2} \cdot \sqrt{k}\Delta|\Psi_k(G)|\right)$ iterations to obtain a $(1-\epsilon)$-approximation solution.

3.2 Review of `SuperGreedy++`

The algorithm that has the lowest theoretical number of iterations with an approximation ratio of $(1-\epsilon)$ is `SuperGreedy++` [14], which finds the densest subgraph that utilizes the generalized supermodular density as a density metric [14, 81]. Since k-clique density is

Algorithm 1: SuperGreedy++ [14]

```
input  : A graph $G$ and two positive integers $k$ and $T$
output : An approximate CDS $\mathcal{S}_k(G)$
1  foreach $v \in V(G)$ do $l^{(0)}(v) \leftarrow 0$; $\mathcal{S}_k(G) \leftarrow G$;
2  foreach $t \leftarrow 1, 2, 3, \cdots, T$ do
3      foreach $v \in V(G)$ do $l^{(t)}(v) \leftarrow l^{(t-1)}(v) + |\Psi_k(v, G)|$;
         $M \leftarrow G$ ;
4      foreach $i \leftarrow 1, 2, 3, \cdots, n$ do
5          $v_i \leftarrow \arg\min_{v \in V(M)} l^{(t-1)}(v)$;
6          update the vertex weights via $k$-clique enumeration;
7          foreach $C \in \Psi_k(v_i, M)$ do
8              foreach $u \in C$ and $u \neq v_i$ do
9                  $l^{(t)}(u) \leftarrow l^{(t)}(u) - 1$ ;
10         Remove $v_i$ and all its adjacent edges from $M$;
11         if $\rho_k(M) > \rho_k(\mathcal{S}_k(G))$ then $\mathcal{S}_k(G) \leftarrow M$;
12 return $\mathcal{S}_k(G)$
```

a specific kind of generalized supermodular density [81], the CDS problem can be solved by SuperGreedy++. Algorithm 1 presents a naive algorithm for finding the CDS using SuperGreedy++.

SuperGreedy++ works in an iterative manner. Specifically, in each iteration, for each vertex $v \in V(G)$, it first initializes the $l(v)$ as the sum of its k-clique engagement and the weight of v in the previous iteration (line 3), and then iteratively removes the vertex with the smallest weight (line 4 - 11). Afterwards, it updates the weights of the remaining vertices via k-**clique enumeration** (lines 7-9). Finally, the subgraph with the highest k-clique density in the whole peeling process is returned (line 12). Note that when the number of iterations is just one, it is the same as Greedy [76]. SuperGreedy++ is guaranteed to find a $(1-\epsilon)$-approximation solution after $\Omega\left(\frac{\Delta \log |\Psi_k(G)|}{\rho_k^*(G) \cdot \epsilon^2}\right)$ iterations, which is often much less than those Frank-Wolfe-based algorithms.

3.3 Limitations of SOTA CDS algorithms

The limitations of SOTA CDS algorithms achieving the best practical and theoretical results are as follows:

(1) Both KCCA *and* SuperGreedy++ *cannot efficiently process large real-world graphs with higher degeneracy values.* While KCCA has achieved remarkable performance on the CDS problem, it still has some limitations, since it employs the k-clique counting algorithm PIVOTER, which has a key limitation that the SCT in PIVOTER costs $O(n \cdot \delta \cdot 3^{\delta/3})$ time for local vertex counting on specified k, and has a space cost of $O(n \cdot 3^{\delta/3})$, where n and δ are the number of vertex and degeneracy value of the graph respectively. As shown in Figure 2, KCCA is costly on the graphs with high degeneracy values in both time and space.

(2) The gap between practical performance and theoretical guarantee. Under the same approximation ratio guarantee, KCCA runs empirically faster than SuperGreedy++, but needs a larger number of iterations in theoretical, while SuperGreedy++ needs fewer iterations, though it needs to enumerate all k-cliques in each iteration, which is very time-consuming; For instance, the SOTA k-clique enumeration algorithm EBBkC [80] needs at least one month to list all the 15-cliques on the DBLP network.

Hence, the limitations above motivate us to design an efficient algorithm that *shares all the advantages of the existing algorithms, a theoretically and practically better algorithm.*

4 OUR GRAPH REDUCTION ALGORITHM

In this section, we propose an effective graph reduction algorithm, which aims to significantly improve the practical performance of all CDS algorithms. It is based on an interesting fact that on most real-world graphs, the ratio of the number of vertices in the CDS comprises no more than 0.11% of the total vertices in the original graph, as shown in Table 2. This is mainly because the CDS problem aims to maximize the k-clique density where vertices with small clique engagements are excluded. Hence, this motivates us to locate the CDS into a relatively small sub-graph before running a CDS discovery algorithm.

Table 2: The ratio of the number of vertices in the CDS to that of the whole graph.

Datasets	# vertices	k=7		k=15	
		$\|V(D_k(G)\|$	ratio	$\|V(D_k(G)\|$	ratio
WT	120,834	127	0.105%	96	0.079%
DP	317,080	114	0.036%	114	0.036%
ZB	7,827,193	325	0.004%	325	0.004%
UK	18,483,190	944	0.005%	944	0.005%
FS	124,836,180	141	0.0001%	141	0.0001%

To achieve this goal, existing works have made some efforts [28, 36, 83, 91], where the two representative methods are k-core-based and k-clique engagement-based, respectively. Specifically, to find CDS with a given k, the former one [36, 91] locates the CDS into a $(k-1)$-core, which is much smaller than the original graph, while the latter one [28, 36, 84] utilizes the relationship between the k-clique engagement and the lower bound of the optimal density to reduce the search space, via the following lemma:

Lemma 4.1 ([36]). *Given a graph G, and a lower bound density of $\rho_k^*(G)$, $\underline{\rho_k}(G)$, then $\mathcal{D}_k(G)$ is contained in the subgraph $G_{\underline{\rho_k}(G)}$, which is induced by the vertices with its k-clique engagement $\geq \underline{\rho_k}(G)$, i.e., $\forall v \in V(G_{\underline{\rho_k}(G)})$, $\Psi_k(v, G_{\underline{\rho_k}(G)}) \geq \underline{\rho_k}(G)$. Here $G_{\underline{\rho_k}(G)}$ is referred to as the search scope w.r.t. the density $\underline{\rho_k}(G)$.*

However, both methods have their limitations: *(1) Effectiveness.* The k-core-based method removes only a small number of vertices, leaving a large search space. For example, on the UK dataset with 18 million vertices and 261 million edges, when setting k=7, the 7-core still contains 11 million vertices and 246 million edges, while the CDS contains only 944 vertices. *(2) Efficiency.* The k-clique engagement-based method requires local k-clique counting for each vertex, which is very costly for graphs with high degeneracy values. The SOTA k-clique counting method PIVOTER struggles to process large graphs (e.g., on the UK dataset, it takes over 100 hours to count local 7-cliques).

To overcome the above issues, we make two key observations: (1) large k k-cliques are composed of many smaller k-cliques, and (2) counting smaller k-cliques is significantly faster than counting larger k-cliques. For example, on the UK dataset, when k=3, the local triangle counting only takes around 21s, while it takes more than 100 hours to count 7-cliques. Motivated by these, we propose a Hierarchical Clique Graph Reduction (HCGR) algorithm that uses

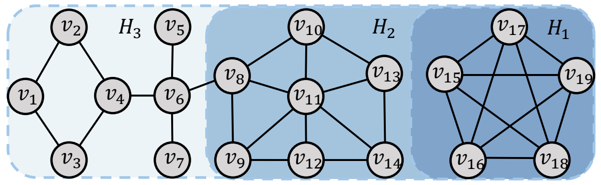


Figure 3: Illustrating our graph reduction technique.

k-clique engagement of small k values to approximate the clique engagement of large k values. We first introduce the relationship between clique numbers of two adjacent k values in Lemma 4.2.

Lemma 4.2. *Given a graph $G = (V, E)$, if $|\Psi_k(G)| \geq \binom{\mu}{k}$ where μ is an arbitrary integer no less than k (i.e., $\mu \geq k$), then we have $|\Psi_{k-1}(G)| \geq \binom{\mu}{k-1}$.*

Proof sketch. We prove the contrapositive: if $|\Psi_{k-1}(G)| < \binom{\mu}{k-1}$, then $|\Psi_k(G)| < \binom{\mu}{k}$. Since each k-clique is contained in some maximal clique, by applying the inclusion-exclusion principle over the set of maximal cliques $\mathcal{R}(G)$, the number of k-cliques in G can be calculated as:

$$|\Psi_k(G)| = \sum_{\mathcal{S} \subseteq \mathcal{R}(G)} (-1)^{|\mathcal{S}|} \binom{|\bigcap_{R \in \mathcal{S}} R|}{k}, \tag{2}$$

where $\mathcal{S}$ is a subset of $\mathcal{R}(G)$ and R denotes a maximal clique in G. Therefore, we can express $|\Psi_k(G)|$ and $|\Psi_{k-1}(G)|$ as linear combinations of combinatorial numbers, which are:

$$|\Psi_{k-1}(G)| = \sum_{i=k-1}^{\mu-1} \beta_i \binom{i}{k-1} \tag{3}$$

$$|\Psi_k(G)| = \sum_{i=k-1}^{\mu-1} \beta_i \binom{i}{k} \tag{4}$$

where, $\beta = (\beta_{k-1}, \beta_k, \cdots, \beta_{\mu-1}) \in \mathbb{Z}^{\mu-k+1}$.
Hence, we have $\frac{|\Psi_k(G)|}{|\Psi_{k-1}(G)|} < \frac{\mu-k+1}{k}$, and we conclue that:

$$|\Psi_k(G)| < \frac{\mu - k + 1}{k} |\Psi_{k-1}(G)| < \binom{\mu}{k} \tag{5}$$

□

The following lemma shows how to use r-clique engagement ($r < k$) to estimate the lower bound of k-clique engagement.

Lemma 4.3. *Given a graph $G = (V, E)$, $\forall v \in V$, if $|\Psi_k(v, G)| \geq \rho$, then for any $r \in \mathbb{N}$, with $3 \leq r \leq k$, we have $|\Psi_r(v, G)| \geq \binom{\mu}{r-1}$, where μ denotes the maximum integer such that $\binom{\mu}{k-1} \leq \rho$.*

The above lemma can be directly proved by using mathematical induction on Lemma 4.2, so we omit the details here.

Example 2. *Figure 3 demonstrates the effectiveness of our graph reduction technique compared to $(k-1)$-core reduction. Suppose we aim to find the 4-CDS. First, the search space is reduced to the 3-core, i.e., all vertices in H_3 can be excluded. However, vertices in H_2 are not in any 4-clique, they remain in the search space. By Lemma 4.3, since H_1 is a 5-clique that provides a density lower bound (i.e., 1) for the optimal solution, we can calculate the maximum μ for k=3, as $\binom{3}{4-1} = 1 \leq 1$. Thus, if a vertex is included in $\mathcal{D}_4(G)$, its 3-clique engagement must be at least $\binom{3}{2} = 3$, and vertices in H_2 are pruned, leaving only H_1 which is actually the 4-CDS.*

Based on the above discussions, we develop an effective algorithm that significantly reduces the search space, as shown in Algorithm 2. The correctness of this reduction is theoretically guaranteed by Lemma 4.3, ensuring that the results remain identical with or without reduction. Here, we employ the maximum clique size (denoted by ω) to estimate the lower bound of optimal density, i.e., $\underline{\rho_k}(G) = \binom{\omega}{k}$. Note that ω needs to be computed in advance. We would like to highlight that, despite being an NP-hard problem, the SOTA maximum clique computing algorithm [12] runs extremely fast in practice, thanks to the power-law distribution of vertex degrees in real-world graphs.

Algorithm 2: HCGR

```
Input  : A graph $G$, two positive integers $k$ and $r$, and the density
         lower bound $\underline{\rho_k}(G)$
Output : A reduced subgraph of $G$
1 $\mu \leftarrow$ the maximum integer such that $\binom{\mu}{k-1} \leq \underline{\rho_k}(G)$ ;
2 while True do
3     $v \leftarrow \arg\min_{u \in V(G)} |\Psi_r(u, G)|$ ;
4     if $|\Psi_r(v, G)| < \binom{\mu}{r-1}$ then
5         $G \leftarrow G[V(G) \setminus v]$ ;
6     else break ;
7 return $G$ ;
```

Given an integer k, HCGR first computes the maximum integer μ based on Lemma 4.3 (line 1). Next, it iteratively removes the vertices whose r-clique engagements are less than $\binom{\mu}{r-1}$ (lines 2-6). Note that in practice, choosing r=3 is sufficient to prune almost all vertices that are not in CDS, as shown in our experimental results later. Besides, local triangle counting can be finished in $O(\delta \cdot m)$ time, which is very efficient. When computing ω is unavailable, we adopt the heuristic algorithm [12] to obtain a relatively large clique size in $O(\delta \cdot m)$ time. That is, a decent lower bound of the optimal density can be estimated, allowing HCGR to remain effective.

5 OUR CCAS ALGORITHM

Inspired by SuperGreedy++, in this section we propose a simple yet effective approximation CDS algorithm by using k-clique counting, rather than k-clique enumeration. By combining the graph reduction algorithm above, our new CDS algorithm achieves a near-optimal solution with both strong practical performance and theoretical guarantees.

5.1 From clique enumeration to clique counting

Recall that in the t-iteration of SuperGreedy++ (see Algorithm 1), the increased weight of vertex v, i.e., $l^{(t)}(v) - l^{(t-1)}(v)$, is the k-clique engagement of v in the remaining graph when vertex v is peeled. After that, we reduce the weights of all the other vertices that share the k-cliques with vertex v (lines 7-9 of algorithm 1). When all the vertices in the graph have been processed in this fashion, one iteration is complete. Let $V_i = \{v_1, \cdots, v_i\}$ denote the set of the first i vertices removed during this iteration. Besides, for each vertex $v \notin V_i$, we denote by $Y_k(v, V_i)$ the number of k-clique engagements that v will lose after all vertices in V_i are removed

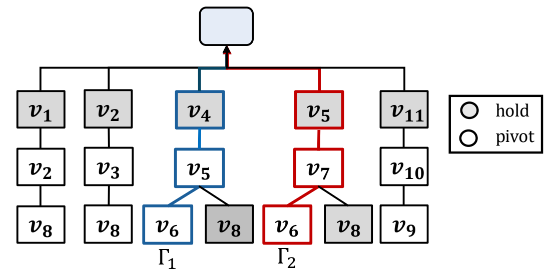


Figure 4: The SCT for the graph in Figure 1.

from $V(G)$, i.e.,

$$Y_k(v, V_i) = |\bigcup_{u \in V_i} \Psi_k(\{v, u\}, G)| \tag{6}$$

We define the residual clique engagement of v as:

$$\hat{l}(v \mid V_i) = \Psi_k(v, G) - Y_k(v, V_i) \tag{7}$$

Then, the difference between $\hat{l}(v \mid V_i)$ and $\hat{l}(v \mid V_{i+1})$ is:

$$\hat{l}(v \mid V_i) - \hat{l}(v \mid V_{i+1}) = Y_k(v, V_{i+1}) - Y_k(v, V_i) \tag{8}$$

We split $Y_k(v, V_{i+1})$ into two parts: those induced by V_i, and the new one by v_{i+1}. Thus,

$$Y_k(v, V_{i+1}) = |(\cup_{u \in V_i} \Psi_k(\{v, u\}, G)) \cup \Psi_k(\{v_{i+1}, v\}, G)| \tag{9}$$

Note that the cliques involving v and v_{i+1} are only affected by vertices not in V_i, we simplify:

$$\hat{l}(v \mid V_i) - \hat{l}(v \mid V_{i+1}) = |\Psi_k(\{v_{i+1}, v\}, G[V \backslash V_i])| \tag{10}$$

Therefore, when a vertex is removed from the graph, we can use local k-clique counting to efficiently maintain the k-clique engagements of the remaining vertices.

Next, we briefly introduce the SOTA k-clique counting algorithm PIVOTER [40], which implicitly constructs a succinct clique tree (SCT) to assign a unique representation for each k-clique. The SCT adapts the recursion tree of the Bron-Kerbosch algorithm for maximal clique enumeration [74]. Specifically, the SCT is built based on this recursion tree, by assigning each vertex a unique label, either "pivot" or "hold", each k-clique can be uniquely represented. The "pivot" is selected to compress the tree by skipping its neighbors during enumeration, while the non-neighbors of the pivot are called the "hold" vertices. Such a tree-shaped index has a virtual root node [1] connecting all second-level sub-trees.

Each root-to-leaf path Γ is uniquely encoded by the pivot vertices (denoted by $\mathcal{P}(\Gamma)$) and hold vertices (denoted by $\mathcal{H}(\Gamma)$) along the path [40]. In addition, we use $V(\Gamma)$ denotes all vertices in Γ, i.e., $V(\Gamma) = \mathcal{P}(\Gamma) \cup \mathcal{H}(\Gamma)$. The following lemma demonstrates how to count the number of k-cliques in each root-to-leaf path.

Lemma 5.1 ([40]). *Given a root-to-leaf path Γ, each k-clique must contain all vertices in $\mathcal{H}(\Gamma)$ and contain $k - |\mathcal{H}(\Gamma)|$ vertices in $\mathcal{P}(\Gamma)$. Each vertex in $\mathcal{P}(\Gamma)$ on this path is contained by $\binom{|\mathcal{P}(\Gamma)|-1}{k-|\mathcal{H}(\Gamma)|-1}$ k-cliques and each vertex in $\mathcal{H}(\Gamma)$ is contained by $\binom{|\mathcal{P}(\Gamma)|}{k-|\mathcal{H}(\Gamma)|}$ k-cliques.*

[1] To avoid ambiguity, we use "node" to represent "vertex" on the SCT, and use "vertex" to represent "vertex" in the graph.

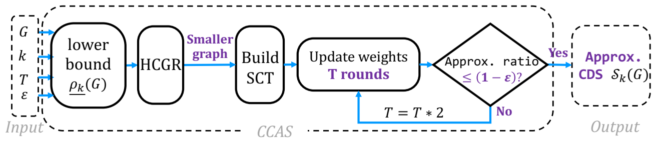


Figure 5: Workflow of CCAS.

Example 3. *Figure 4 shows the SCT of the graph in Figure 1, where each node shows the id of the vertex it stores. For instance, to count the 3-cliques containing v_6, we need to traverse two root-to-leaf paths $\Gamma_1 = \langle root, v_4, v_5, v_6 \rangle$ and $\Gamma_2 = \langle root, v_5, v_7, v_6 \rangle$. For Γ_1, since it has one hold vertex and two pivot vertices, there are $\binom{|\mathcal{P}(\Gamma)|-1}{k-|\mathcal{H}(\Gamma)|-1} = \binom{1}{1}$ = one 3-cliques containing v_6 in Γ_1. Similarly, there is $\binom{1}{1}$ = one 3-clique containing v_6 in Γ_2. In total, there are two 3-cliques containing v_6.*

5.2 Our CCAS algorithm

Based on the discussions above, we develop a new CDS algorithm, denoted by CCAS, by adapting the SCT for the local k-clique counting. Algorithm 3 shows CCAS, and we present its workflow in Figure 5. Specifically, to obtain a $(1 - \epsilon)$-approximate CDS $\mathcal{S}_k(G)$, CCAS takes four input parameters: G, k, T, and ϵ, and sequentially performs the following steps: 1) compute a lower bound $\underline{\rho_k}(G)$ of the optimal density, which is used in the graph reduction (line 1); 2) remove vertices not belonging to the CDS using the HCGR (line 2); 3) build the SCT (line 3) and based on which, update the vertex weights of T iterations via `updateIter` on the reduced graph from Step 2 (lines 12-28); 4) extract the subgraph with the highest density during Step 3 and verify whether it satisfies the $(1 - \epsilon)$-approximation; and 5) if so, output the CDS; otherwise, update T to $2 \times T$ and repeat from Step 3 until the error criterion is satisfied (lines 6-10).

In each iteration of `updateIter`, it first initializes $l(v)$ as the sum of its k-clique engagement and the weight of v in the previous iteration (line 14). Then, it iteratively removes the vertices with the smallest l values from the graph, and when a vertex v_i is removed, it needs to update the weights of v_i' neighbors in the remaining graph. To achieve this, it traverses all SCT paths containing v_i to update the corresponding vertex weights (lines 19 - 27). Since it only needs to update the vertex weights in the remaining graph, for each path Γ, it first modifies its hold and pivot vertex sets (lines 20 and 22), where v_i is considered as the hold vertex. Next, it updates the l value for each vertex in the remaining graph by its k-clique engagement on this path (lines 23 - 26). Finally, the subgraph with the highest density, during the whole process, would be returned as the candidate CDS.

Theorem 5.2. *Given a graph G with n vertices and degeneracy of δ, CCAS costs $O(\xi \cdot \delta)$ space, and $O(\xi \cdot \delta^3)$ time for each iteration, where ξ denotes the cardinality of SCT.*

Theorem 5.3. *CCAS takes $\Omega\left(\frac{\Delta \log |\Psi_k(G)|}{\rho_k^*(G) \cdot \epsilon^2}\right)$ iteration to obtain a $(1 - \epsilon)$-approximation solution.*

Generally, our algorithm can be viewed as a clique counting version of SuperGreedy++, so it achieves the same convergence rate as SuperGreedy++ [14].

Theorem 5.4. *CCAS reduces the overall time complexity at least by $O(\rho_k^*(G) \cdot \sqrt{k})$ over KCCA and by $O(\frac{(\frac{\delta}{2})^{k-2}}{\xi})$ over SuperGreedy++,*

Algorithm 3: CCAS

```
input  : A graph $G$ and two positive integers $k$, $T$ and a real value $\epsilon$
output : An approximate CDS $\mathcal{S}_k(G)$
1  $\underline{\rho_k}(G) \leftarrow$ compute the density lower bound of $\mathcal{D}_k(G)$;
2  $G \leftarrow$ HCGR($G$, $\underline{\rho_k}(G)$); // our graph reduction algorithm;
3  SCT $\leftarrow$ build_SCT($G$); // build the SCT for $G$;
4  $t_s \leftarrow 1$; $t_e \leftarrow T$; $f \leftarrow$ False;
5  foreach $v \in V(G)$ do $l^{(0)}(v) \leftarrow 0$; $\mathcal{S}_k(G) \leftarrow G$ ;
6  repeat
7      $\mathcal{S}_k(G) \leftarrow$ updateIter($G$, SCT, $t_s$, $t_e$);
8      if $\mathcal{S}_k(G)$ satisfies $(1-\epsilon)$-approx. ratio then $f \leftarrow$ True ;
9      else $t_s \leftarrow t_e + 1$; $t_e \leftarrow t_e * 2$ ;
10 until $f$=True;
11 return $\mathcal{S}_k(G)$;

12 Function updateIter($G$, $SCT$, $t_s$, $t_e$):
13     foreach $t \leftarrow t_s, t_{s+1}, \cdots, t_e$ do
14         foreach $v \in V(G)$ do $l^{(t)}(v) \leftarrow l^{(t-1)}(v) + |\Psi_k(v, G)|$ ;
15         $M \leftarrow G$ ;
16         foreach $i \leftarrow 1, 2, 3, \cdots, |V(G)|$ do
17             $v_i \leftarrow \arg\min_{v \in M} l^{(t)}(v)$ ;
18             remove $v_i$ and all its adjacent edges from $M$;
19             foreach root-to-leaf path $\Gamma \in SCT$ and $v_i \in \mathcal{V}(\Gamma)$ do
20                 $\mathcal{H} \leftarrow (\mathcal{H}(\Gamma) \cap V(M)) \cup \{v_i\}$ ;
21                 if $\mathcal{H}(\Gamma) \nsubseteq \mathcal{H}$ then continue; ;
22                 $\mathcal{P} \leftarrow \mathcal{P}(\Gamma) \cap V(M)$ ;
23                 foreach $v \in \mathcal{H}$ and $v \neq v_i$ do
24                     $l^{(t)}(v) \leftarrow l^{(t)}(v) - \binom{|\mathcal{P}|}{k-\mathcal{H}}$ ;
25                 foreach $u \in \mathcal{P}$ do
26                     $l^{(t)}(v) \leftarrow l^{(t)}(v) - \binom{|\mathcal{P}|-1}{k-\mathcal{H}-1}$ ;
27             if $\rho_k(M) > \rho_k(\mathcal{S}_k(G))$ then $\mathcal{S}_k(G) \leftarrow M$;
28     return $\mathcal{S}_k(G)$;
```

respectively, where all variables keep the same meanings as the THEOREM *5.2.*

The detailed proof is shown in our technical report [64].

Example 4. *Continue Example 3 with $k = 3$. Figure 6 shows the vertex weight update process in* CCAS *due to the removal of v_3. The first row shows the vertex weights $l^{(1)}(v)$ before removing v_6, where the gray shaded boxes denote those vertices that are removed. The blue shaded boxes contain the weights of the vertices on this path. Since v_6 has the minimum weight among all vertices in the first row, when we process Γ_1, we have $\mathcal{H} = \{v_4\}$, and $\mathcal{P} = \{v_5, v_6\}$, so v_4's weights are updated by the number of cliques containing them in this path, i.e., $2 - \binom{2-1}{3-1-1} = 1$, and v_5's weight is updated to $4 - \binom{2-2}{3-2-1} = 3$. Similarly, when we process Γ_2, v_5 and v_7's weights are updated by the number of cliques containing them in this path, i.e., $3 - \binom{2-1}{3-1-1} = 2$. Finally, v_6 is removed from the graph.*

5.3 Early stop criterion

Note that to obtain a $(1-\epsilon)$-approximation solution, CCAS requires $\Omega\left(\frac{\Delta \log |\Psi_k(G)|}{\rho_k^*(G) \cdot \epsilon^2}\right)$ iterations. However, the optimal density is often unknown in advance [23, 50, 71]. On the other hand, as shown in the existing works [71, 91, 92], to obtain a $(1-\epsilon)$-approximation solution, the practical number of iterations required is significantly lower than the theoretically estimated value. More specifically, these algorithms aim to optimize the convex programming (CP) formulation [71] of the CDS problem, where the CP formulation is shown in the following:

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
before removing v_6	1	1	0	2	4	2	2	2	1	0	0	0
process path Γ_1	1	1	0	1	3	2	2	2	1	0	0	0
process path Γ_2	1	1	0	1	2	2	1	2	1	0	0	0
remove v_6	1	1	0	1	2	2	1	2	1	0	0	0

Figure 6: Illustrating the weight update of CCAS.

$$
\begin{aligned}
\mathrm{CP}(G,k) \qquad & \min \max_{v \in V} r(v) \\
\text{s.t.} \qquad & r(v) = \sum_{C \in \Psi_k(v,\,G)} \alpha_v^C, && \forall v \in V \\
& \sum_{v \in C} \alpha_v^C = 1, && \forall C \in \Psi_k(G) \\
& \forall v \in C, \alpha_v^C \geq 0, && \forall C \in \Psi_k(G)
\end{aligned}
$$

where α_v^C indicates the weight assigned to v from a clique C containing it, and $r(v)$ is the weight sum received by v from all the k-cliques containing v. Here, the vector $\mathbf{r} = \begin{bmatrix} r(v_1) & r(v_2) & \cdots & r(v_n) \end{bmatrix}$. We observe that $\|\mathbf{r}\|_\infty = \max_{v \in V} r(v)$, which means that the objective function of CP(G,k) is: $\min \|\mathbf{r}\|_\infty$. Notice that in the CDS $\mathcal{D}_k(G)$, it is possible to distribute all cliques weights such that the weight sum received by each vertex is exactly $\rho_k^*(G)$, meaning that each vertex $v \in V(\mathcal{D}_k(G))$ has $r(v) = \rho_k^*(G)$. Indeed, $\|\vec{r}\|_\infty$ is a decent upper bound of $\rho_k^*(G)$, and tighter bounds can be derived via Lemma 13 in [71] by using the vector $\mathbf{r}$. This is useful in estimating the approximation ratio in practice when the exact solution is unavailable.

In our algorithm, each vertex v is assigned a weight $l(v)$, based on this, we aim to establish a connection between the weight $l(v)$ in our algorithm and the value $r(v)$ in the CP(G,k). More specifically, our goal is to leverage the vertex weight vector $\vec{l} = \left[l(v_1), l(v_2), \cdots, l(v_n)\right]$ used in our algorithm, as an upper bound of $\rho_k^*(G)$, and use it to estimate the approximation ratio of our algorithm. This provides a theoretical guarantee for CCAS, even when the optimal density is unknown, making it more practical for real-world scenarios.

In the t-th iteration of CCAS, each k-clique assigns its unit weight to the vertex that is removed first from the graph. Let $\pi^{(t)}(v)$ represents the removal order of vertex v, that is, if v is removed before u , then $\pi^{(t)}(v) < \pi^{(t)}(u)$. Expressed in terms of $\widehat{\alpha}_v^C$:

$$
\widehat{\alpha}_v^C = \begin{cases} 1 & \text{if } v = \arg\min_{u \in C} \pi^{(t)}(u) \\ 0 & \text{otherwise} \end{cases}
$$

By using it, for any vertex v, $l^{(t)}(v)$ can be updated by:

$$
l^{(t)}(v) = l^{(t-1)}(v) + \sum_{C \in \Psi_k(v,\,G)} \widehat{\alpha}_v^C
$$

By dividing both sides by t, we have:

$$\frac{l^{(t)}(v)}{t} = \frac{l^{(t-1)}(v)}{t} + \sum_{C \in \Psi_k(v,G)} \frac{\widehat{\alpha}_v^C}{t} = \sum_{C \in \Psi_k(v,G)} \alpha_v^{C^{(t)}},$$

where $\alpha_v^{C^{(t)}}$ retains the same meaning as in the formulation $\mathsf{CP}(G,k)$, representing the weight assigned to vertex v from a clique C containing it. Hence, in the t-th iteration $(\alpha^{(t)}, \frac{l^{(t)}}{t})$ is a feasible solution for $\mathsf{CP}(G,k)$. We further demonstrate that, even without knowing the optimal density, Algorithm 3 is guaranteed to achieve a $(1-\epsilon)$-approximation ratio solution within $\Omega\left(\frac{\log \epsilon}{\epsilon^2} \cdot \Delta|\Psi_k(G)|\right)$ iterations, outperforming KClist++, SCTL, and PSCTL.

That is, while our algorithm does not explicitly involve the α vector, it is implicitly embedded within the process, to be more specific, in the $t+1$-th iteration of CCAS, $\alpha^{(t+1)}$ is updated by:

$$\alpha^{(t+1)} = \frac{t}{t+1} \cdot \alpha^{(t)} + \frac{1}{t+1} \cdot \widehat{\alpha} \tag{11}$$

In fact, Equation (11) can be interpreted as updating the α vector using a convex combination with a learning rate of $\frac{1}{t+1}$, similar to existing CDS methods. Based on this, we can derive another convergence of CCAS, which is guaranteed to find a $(1+\epsilon)$-approximation density upper bound of the optimal solution after $\Omega\left(\frac{\log \epsilon}{\epsilon^2} \cdot \Delta|\Psi_k(G)|\right)$ iterations. Specifically, $\|\vec{l}/t\|_\infty$ is a decent upper bound, and tighter bounds can be derived via Lemma 13 in [71] by using vector $\vec{l}/t$ as the $\mathbf{r}$ vector in $\mathsf{CP}(G,k)$.

Remark. Our CCAS reduces the overall time complexity (the product of # of iterations and the time complexity per iteration) by a factor of $O(\rho_k^*(G) \cdot \sqrt{k})$ over KCCA and a factor of $O(\frac{(\frac{\delta}{2})^{k-2}}{\xi})$ over SuperGreedy++, respectively, as shown in Theorem 5.4. A detailed comparison of CCAS with the existing works in [64].

5.4 Limitations

While CCAS has achieved remarkable performance on the CDS problem, the method still has some limitations. Since our algorithm employs local k-clique counting from PIVOTER, it inherits PIVOTER's limitations. Specifically, for extremely dense graphs with heavily overlapping cliques, the effectiveness of our graph reduction technique HCGR diminishes, as fewer vertices can be pruned in advance. Thus, both the running time and memory usage of CCAS increase significantly, making it less suitable for such datasets. Note that this type of graph is notoriously challenging for many clique-based tasks (e.g., clique counting [40], listing [22, 80], and maximal/maximum clique discovery [12, 49, 53]), and the same difficulty applies to all existing CDS algorithms, including CCAS. Fortunately, most real-world graphs follow power-law degree distributions, which allows CCAS to be time- and space-efficient in practice.

6 A CDS ALGORITHM FOR ALL k VALUES

In practical applications, users may not know which k value best suits their needs, so they may have to explore the CDS's for a variety of k values. However, all the existing algorithms treat each k value independently, so it is inefficient to use them to compute the CDS's for various k values. To fill this gap, in this section we propose a novel algorithm for finding the CDS's for all the possible k values, by utilizing the power of our graph reduction technique.

We first present a novel lemma to demonstrate the density relationship between the large and smaller k values.

Lemma 6.1. *Given a graph G, and an approximate k-clique CDS, $\mathcal{S}_k(G)$, for any $k' < k$, we have $\rho_{k'}^*(G) \geq \frac{\binom{\gamma}{k'}}{|\mathcal{S}_k(G)|}$, where γ is the maximum integer such that $\binom{\gamma}{k} \leq \rho_k(\mathcal{S}_k(G)) \cdot |\mathcal{S}_k(G)|$.*

Using Lemma 6.1, we can leverage the solution for a larger k to generate lower bound densities for smaller $k' < k$. A straightforward approach is first to compute the ω-clique CDS, where ω is the maximum clique size in G. Then, for each smaller k, a density lower bound can be derived from $\mathcal{S}_\omega(G)$ and used in the graph reduction process. (Algorithm 2). However, this approach is inefficient and leads to redundant computations, since HCGR must be re-executed on the entire graph for each k. To enhance efficiency, we propose a novel algorithm based on the nested relationships among the graphs after reduction across different k values.

Theorem 6.2. *Given a graph G, an approximate k-clique CDS, $\mathcal{S}_k(G)$, for any integers $3 \leq x < k$, we use $\underline{\rho_x}(G)$ and $\underline{\rho_{x+1}}(G)$ to denote the lower bound of optimal densities obtained by the Lemma 6.1, then we have $\mu_x \leq \mu_{x+1}$, where μ_x and μ_{x+1} are derived by Lemma 4.3, using $\underline{\rho_x}(G)$ and $\underline{\rho_{x+1}}(G)$.*

Proof sketch. Recall that μ_{x+1} is the maximum integer satisfying:

$$\binom{\mu_{x+1}}{x} \leq \frac{\binom{\gamma}{x+1}}{|\mathcal{S}_\omega(G)|}. \tag{12}$$

To prove $\mu_x \leq \mu_{x+1}$, we need to show that:

$$\binom{\mu_x}{x} \leq \frac{\binom{\gamma}{x+1}}{|\mathcal{S}_\omega(G)|}. \tag{13}$$

This is equivalent to proving:

$$\mu_x \leq \gamma - \frac{\gamma+1}{x+1} \tag{14}$$

Since $\rho_\omega(\mathcal{S}_\omega(G)) \cdot |\mathcal{S}_\omega(G)| \geq 1$, we have $\gamma \geq \omega > x$. We rewrite γ as $p \cdot (x+1) + b$, where $p \geq 1$ and $0 \leq b \leq k$, giving:

$$\mu_x \leq p \cdot x + b - 1 \tag{15}$$

We prove this by contradiction. If $\mu_x > p \cdot x + b - 1$, it must violate the definition of μ_x.

Since $\binom{\mu_x}{x}$ is a non-decreasing function of μ_x, we only need to check $\mu_x = p \cdot x + b$. This leads to the inequality:

$$\binom{p \cdot x + b}{x} > \frac{\binom{p \cdot x + p + b}{x+1}}{|\mathcal{S}_\omega(G)|}. \tag{16}$$

Using the fact that $|\mathcal{S}_\omega(G)| \geq \gamma$, we derive:

$$\frac{(p \cdot x + b)!}{(p \cdot x + b - x + 1)!} > \frac{(p \cdot (x+1) + b - 1)!}{x \cdot (p \cdot (x+1) + b - x)!} \tag{17}$$

$$x > \prod_{i = p \cdot x + b - x + 2}^{p \cdot x + b - x + p} \frac{(i + x - 1)}{i} \tag{18}$$

For $p = 1$, the right-hand side is clearly smaller than x. For $p \geq 2$, the right-hand side is bounded by e.

Hence, we finish our proof since $x \geq 3 > e$. □

Theorem 6.2 enables us to adopt an incremental approach to find the CDS for all k values. Specifically, the subgraph produced by executing HCGR for a particular k value can be directly utilized in subsequent computations for larger k values. This means that we can perform these calculations on the reduced graph—or sketch—instead of the entire original graph, thereby significantly improving efficiency. When finding the CDS for an integer k, we set $r = k - 1$ instead of the default $r = 3$. This adjustment leads to more effective graph reduction by removing more vertices. Based on the discussions above, we develop a CDS algorithm for all k values, denoted by CCAS-A, which is presented in our technical report [64].

7 EXPERIMENTS

We now present the experimental results. Section 7.1 discusses the setup. We discuss the results in Sections 7.2 and 7.3.

Table 3: Datasets used in our experiments.

Dataset	Category	n	m	δ
bio-SC-GT (BG)	Biological	1,716	31,564	60
econ-beacxc (EB)	Economic	507	42,176	118
WikiTalk (WT)	Communication	120,834	237,551	54
Slashdot (SD)	Comments	77,360	469,180	54
DBLP (DP)	Collaboration	317,080	1,049,866	113
HepTh (HT)	Citation	22,908	2,444,798	561
Hollywood (HW)	Collaboration	1,069,126	56,306,652	2,208
zhishi-baidu (ZB)	Hyperlink	7,827,193	62,246,014	267
UK-2002 (UK)	Web	18,483,190	261,787,260	943
Arabic-2005 (AC)	Web	22,743,892	553,903,073	3,247
IT-2004 (IT)	Web	41,290,648	1,027,474,947	3,224
Friendster (FS)	Social	124,836,180	1,806,067,135	304
SK-2005-web (SK)	Web	50,636,059	1,810,063,330	4,510
UK-2006 (US)	Web	77,449,748	2,635,849,931	5,014

7.1 Setup

Datasets. We use 14 real-world datasets from various domains, which are downloaded from the Stanford Network Analysis Platform [62], Laboratory of Web Algorithmics [61], Network Repository [65], and Konect [44]. Their detailed descriptions can also be found on these websites. Table 3 reports the statistics of these graphs, where δ denotes the degeneracy of graph.

Competitors. We mainly compare our CCAS algorithm with the following approximation CDS algorithms:

- KClist++ [71]: the convex programming-based algorithm, which is briefly recapped in Section 1.
- PSCTL [83]: the improved version of SCTL, which is discussed in Section 3.
- SuperGreedy++ [14]: the algorithm needs the lowest theoretical number of iterations, as discussed in Section 3.
- ESuperGreedy++: the variant version of SuperGreedy++ by replacing KClist in which with the SOTA clique enumeration algorithm EBBkC [80].
- KCCA [91]: the SOTA approximation CDS algorithm, which is discussed in Section 3.

Note that CoreApp [28] is not included in our experiments, because its efficiency and accuracy were significantly outperformed by our compared algorithms [36, 91]. Besides, we compare PSCTL instead of SCTL, since the former one is a better version of SCTL in both efficiency and accuracy. We implement all the algorithms in C++ and run experiments on a machine having an Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz and 1TB of memory, with Ubuntu installed. If an algorithm cannot finish within 100 hours, we mark its running time as "INF" and its memory usage as "—". If an algorithm requires more than 1TB of memory, we mark its running time as "—" and its memory usage as "OOM" in the tables. In addition, if an algorithm encounters either "INF" or "OOM", we mark its accuracy as "—". In our experiments, we have already included the time cost of building the SCT and finding the maximum cliques in all results.

Table 4: Results on datasets with highest degeneracy values.

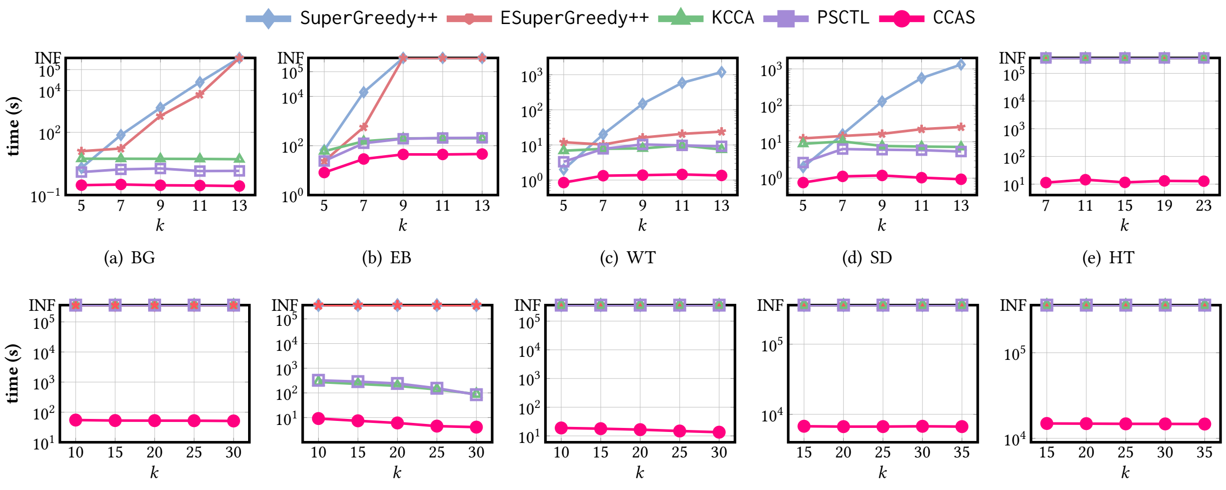
k	AC			IT		
	Time (s)	Accuracy	Memory (GB)	Time (s)	Accuracy	Memory (GB)
5	6,733	1.000	14.7	15,086	0.997	27.2
15	6,762	1.000	13.8	14,921	0.992	23.2
20	6,684	1.000	12.6	14,856	0.991	23.7
25	6,690	1.000	12.0	14,773	0.990	22.5
30	6,755	1.000	11.7	14,747	0.990	22.0
35	6,687	1.000	11.5	14,697	0.990	21.6
45	6,754	1.000	11.0	14,627	0.989	20.8
k	SK			US		
	Time (s)	Accuracy	Memory (GB)	Time (s)	Accuracy	Memory (GB)
5	47,654	1.000	48.0	119,171	0.986	70.1
15	48,499	1.000	46.0	39,708	0.999	66.3
20	47,462	1.000	44.8	39,901	0.999	63.7
25	48,050	1.000	43.5	39,816	0.999	60.3
30	47,758	1.000	41.6	39,877	0.999	56.9
35	48,015	1.000	39.6	39,773	0.999	55.7
45	47,529	1.000	37.9	41,022	0.999	53.5

Running details. For datasets where KCCA and PSCTL encounter out-of-memory (OOM) issues, we use their memory-friendly versions as an alternative. That is to say, during the SCT building process, we only keep the SCT nodes in the main memory, until their memory usage is close to 1TB, for the remaining nodes, we recompute them in each iteration to avoid the OOM problem. By default, both ϵ and T are set to 1 if not explicitly specified.

7.2 Overall comparison results

In this section, we extensively compare CCAS with the competitor algorithms from various angles.

1. Effect of k. Figure 7 depicts the average running time of all the CDS algorithms on ten datasets by varying the clique size k, where k=5∼30 and T=10. Clearly, CCAS is up to four orders of magnitude faster than KCCA and PSCTL, since it has the better theoretical guarantee and can find the CDS over the very small graph, whereas KCCA and PSCTL struggle to handle the graph with high degeneracy value, such as HW and UK datasets. In addition, even for the graphs with low degeneracy values, CCAS is still more efficient than the competitor methods, because, in each iteration, it requires just a straightforward operation, compared to KCCA and PSCTL. Moreover, KCCA and PSCTL achieve comparable performance on all datasets, since both of them are based on the Frank-Wolfe algorithm and local vertex k-clique counting. Moreover, all clique-counting-based algorithms are significantly faster than the enumeration-based methods, SuperGreedy++ and ESuperGreedy++. Notably, ESuperGreedy++ achieves up to 40× speedup over SuperGreedy++, owing to a more efficient clique enumeration algorithm. Hence, in the following experiments, we mainly compare PSCTL and KCCA.



(a) BG (b) EB (c) WT (d) SD (e) HT

(f) HW (g) ZB (h) UK (i) AC (j) IT

Figure 7: Effect of k on the efficiency of SuperGreedy++, ESuperGreedy++, KCCA, PSCTL, and CCAS.

In addition, we report the running time, accuracy, and memory cost of CCAS on the four graphs with largest degeneracy values in Table 4. Note that both PSCTL and KCCA not only require over TB of memory but also take more than 100 hours to find the CDS; thus, we omit their results. However, by using a few thousand seconds, CCAS can obtain solutions that are extremely close to optimal, and only take less than 80 GB of memory.

Table 5: Effect of ϵ and k. (Processing time (in seconds); Best performers are highlighted in bold.)

Dataset	Method	$k = 7$				$k = 15$			
		1	0.1	0.05	0.01	1	0.1	0.05	0.01
HT	PSCTL	79,752	79,752	155,885	268,701	INF	INF	INF	INF
	KCCA	76,191	76,191	150,316	261,231	INF	INF	INF	INF
	CCAS	**8.6**	**8.6**	**8.6**	**8.6**	**8.5**	**8.5**	**8.5**	**8.5**
WG	PSCTL	2.8	2.8	6.9	18.3	1.4	3.2	3.2	4.8
	KCCA	7.0	8.2	11.5	16.0	8.3	8.3	8.3	8.6
	CCAS	**0.2**	**0.2**	**0.3**	**0.3**	**0.2**	**0.2**	**0.2**	**0.2**
HW	PSCTL	INF	INF	INF	INF	INF	INF	INF	INF
	KCCA	INF	INF	INF	INF	INF	INF	INF	INF
	CCAS	**52.3**	**52.3**	**52.3**	**52.3**	**52.2**	**52.2**	**52.2**	**52.2**
ZB	PSCTL	89.4	123.5	182.1	507.6	100.4	143.9	143.9	634.7
	KCCA	185.5	261.5	261.5	339.2	150.2	176.9	176.9	405.3
	CCAS	**11.5**	**11.5**	**11.5**	**12.1**	**9.3**	**9.3**	**9.3**	**10.0**
UK	PSCTL	INF	INF	INF	INF	INF	INF	INF	INF
	KCCA	INF	INF	INF	INF	INF	INF	INF	INF
	CCAS	**78.6**	**78.6**	**78.6**	**78.6**	**20.0**	**20.0**	**20.0**	**20.0**
AC	PSCTL	INF	INF	INF	INF	INF	INF	INF	INF
	KCCA	INF	INF	INF	INF	INF	INF	INF	INF
	CCAS	**5,853**	**5,853**	**5,853**	**5,853**	**6,703**	**6,703**	**6,703**	**6,703**

2. Effect of ϵ. We evaluate the effect of ϵ using five datasets from different domains, where the values of ϵ are set to 1, 0.1, 0.05, and 0.01, respectively. Here, we only report the results for 0.1 and 0.01, and the other results are shown in our technical report. The experimental results are reported in Table 5, which clearly shows that CCAS outperforms the other algorithms on all datasets. Particularly, on HT dataset, CCAS is up to four orders of magnitude faster than both KClist++ and SCTL, and for around half of the datasets, CCAS is over three orders of magnitude faster than its competitors. In addition, on the datasets with high degeneracy values, both PSCTL and KCCA struggle to produce reasonable solutions within 100 hours in most cases, because the competitors typically contain a vast number of nodes in their SCTs.

Table 6: Running time of all k values. (Processing time (in seconds); Best performers are highlighted in bold.)

Dataset	Method	$\epsilon = 0.1$				$\epsilon = 0.01$			
		25%	50%	75%	100%	25%	50%	75%	100%
DP	PSCTL	6.2	10.2	13.8	16.9	9.1	13.2	16.7	19.8
	KCCA	6.4	7.1	7.5	7.7	8.1	8.8	9.3	9.5
	CCAS-A	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**
ZB	PSCTL	3,509	4,053	4,511	4,778	12,181	12,968	13,569	13,834
	KCCA	4,781	5,033	5,163	5,230	9,949	10,273	10,453	10,581
	CCAS-A	**66.0**	**103.0**	**136.7**	**161.7**	**106.0**	**166.6**	**217.3**	**247.5**
UK	PSCTL	INF	INF	INF	INF	INF	INF	INF	INF
	KCCA	INF	INF	INF	INF	INF	INF	INF	INF
	CCAS-A	**456.6**	**456.6**	**456.6**	**456.6**	**456.6**	**456.6**	**456.6**	**456.6**
FS	PSCTL	125,544	191,483	245,604	286,646	201,621	267,560	321,681	INF
	KCCA	INF	INF	INF	INF	INF	INF	INF	INF
	CCAS-A	**2,167**	**2,168**	**2,169**	**2,169**	**3,090**	**3,091**	**3,092**	**3,092**

3. Efficiency of finding the CDS's for all k. We compare the efficiency of our algorithm CCAS-A and others for finding the CDS's for all k values. Specifically, for each graph, we record the running time of those algorithms as they process the k values from 3 to 25% $\cdot\omega$, 50% $\cdot\omega$, 75% $\cdot\omega$, and 100% $\cdot\omega$ across six datasets in Table 6. We only present the results for ϵ=0.1 and 0.01, and the results for other approximate ratios are shown in our technical report [64]. We make the following observations and analysis: (1) the runtime of PSCTL and KCCA increases proportionally with the number of k values they process, while CCAS-A maintains a nearly constant runtime. This stable performance is because CCAS-A uses historical computation information to prune the search space; (2) for graphs with high degeneracy values, PSCTL and KCCA cannot complete within 100 hours, even for the 25% $\cdot\ \omega$ values; (3) CCAS-A almost takes the same time to handle one k value as it does ω values for some datasets, as shown in Figure 7.

4. Overall performance. We provide a comprehensive comparison of the CDS algorithms in terms of accuracy, running time (s), and memory usage (MB) across 10 datasets, as summarized

Table 7: Results across different CDS algorithms in terms of accuracy, running time (s), and memory usage (MB).

Dataset	k	Accuracy				Running time (s)				Memory (MB)			
		PSCTL	KCCA	SuperGreedy++	CCAS	PSCTL	KCCA	SuperGreedy++	CCAS	PSCTL	KCCA	SuperGreedy++	CCAS
EB	5	0.99	0.99	1.00	1.00	24.0	59.0	86.0	8.2	185	8,426	3	493
	7	1.00	0.99	1.00	1.00	126.0	151.0	19,076.0	28.9	1,291	9,157	4	890
	9	1.00	0.99	—	1.00	190.2	197.3	INF	44.4	2,540	9,378	—	1,068
	11	1.00	0.99	—	1.00	205.0	200.0	INF	44.2	2,541	9,396	—	1,086
	13	1.00	0.99	—	1.00	207.1	200.0	INF	46.0	2,540	9,396	—	1,086
ZB	10	0.99	0.99	—	0.99	322.0	276.0	INF	9.2	5,798	46,747	—	1,524
	15	0.99	0.99	—	0.99	288.0	230.2	INF	7.4	5,786	21,283	—	1,412
	20	0.99	0.99	—	0.99	242.0	192.0	INF	6.1	5,770	18,064	—	1,326
	25	0.99	0.99	—	0.99	154.0	138.3	INF	4.5	4,766	15,060	—	1,283
	30	0.99	0.99	—	0.99	85.0	89.0	INF	4.1	4,337	12,381	—	1,256
UK	10	—	—	—	1.00	INF	INF	INF	18.0	OOM	OOM	—	6,232
	15	—	—	—	1.00	INF	INF	INF	17.0	OOM	OOM	—	6,046
	20	—	—	—	1.00	INF	INF	INF	16.0	OOM	OOM	—	5,834
	25	—	—	—	1.00	INF	INF	INF	14.0	OOM	OOM	—	5,612
	30	—	—	—	1.00	INF	INF	INF	13.0	OOM	OOM	—	5,426

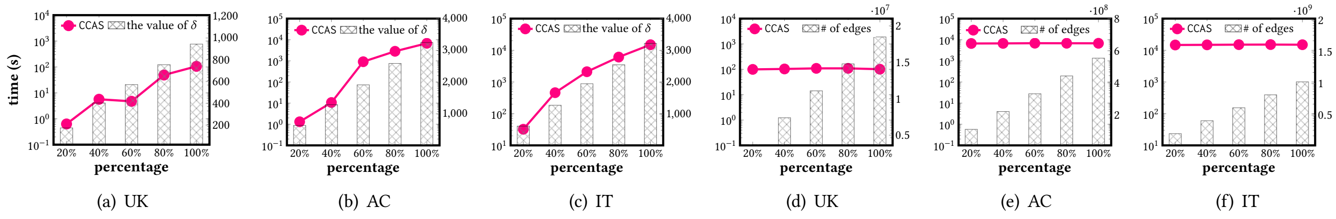


Figure 8: Scalability test for CCAS algorithm.

in Table 7, where $k = 5 \sim 30$ and $T = 10$. Due to the space limitation, we only provide the results on three datasets here, and move the remaining results and detailed analysis into our technical report [64]. We can make the following observations and analysis: (1) All methods achieve comparable performance in terms of accuracy (2) Our method, CCAS, is significantly faster than all competitors on all datasets and different k values. (3) SuperGreedy++ always consumes less memory than other methods due to its lightweight space complexity (i.e., $O(m)$). However, SuperGreedy++ is extremely time-consuming because it needs to enumerate all cliques. Among the three clique-counting-based algorithms, although they share the same theoretical space complexity, CCAS typically uses less memory in practice, thanks to our graph reduction technique, which effectively reduces memory usage.

7.3 Detailed analysis of CCAS

We perform an in-depth evaluation and analysis of CCAS.

1. Time cost of different steps in CCAS. Recall that CCAS sequentially performs the following three steps: (1) reducing the original graph by our proposed graph reduction technique (HCGR), (2) building the SCT (buildTree), and (3) updating vertex weights with T iterations via SCT updateIter. Figure 9 shows the time cost of these three steps on ten datasets, where $k = 15$, $T = 10$, and the graphs are assumed to be loaded into memory. We see that on the small datasets, the third step accounts for a relatively large portion of the total time. Besides, on the other datasets, HCGR and buildTree are the most computationally expensive steps. For example, the time cost of HCGR on the ZB and WG datasets is higher

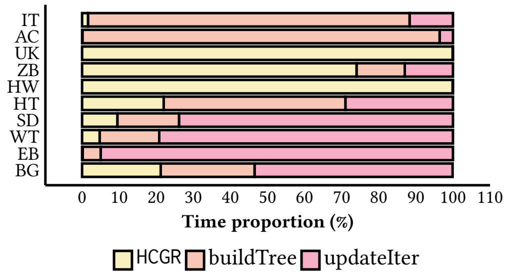
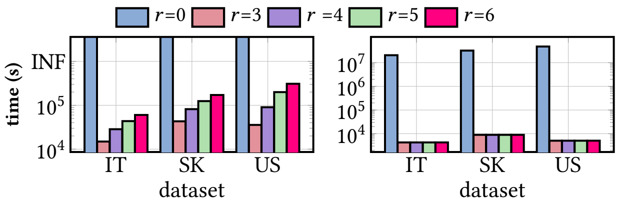


Figure 9: Proportion of time cost of each step in CCAS.

than other steps, and the time cost of buildTree on the AC and IT datasets is the most significant proportionally.

2. Scalability test. In this experiment, we evaluate the scalability of our CDS algorithm from two perspectives using k=7 and T=10: (1) For each graph, we create five induced subgraphs by randomly selecting 20%, 40%, 60%, 80%, and 100% of the vertices. We run CCAS on these subgraphs and report the average runtime and the resulting degeneracy value δ. (2) Second, to isolate the impact of the number of edges while maintaining a fixed dense region, we construct another set of five subgraphs by selecting the top 20% to 100% of edges based on their core numbers, where the core number of an edge is defined as the smaller core number of its two endpoints. This ensures that all subgraphs are sampled from the high-density region of the graph, preserving the core structure while gradually increasing the number of edges. We evaluate CCAS on these subgraphs and report the average runtime and resulting edge counts. The results on the three largest graphs are given in Figure 8, and the additional results are presented in our technical



(a) running time (b) # of vertices

Figure 10: Effectiveness of the graph reduction technique.

report [64]. Based on the above results, we make the following observations and analysis: (1) The running time of CCAS on all datasets is proportional to the value of δ, which is aligned with our time complexity analysis of CCAS. (2) When the value of δ is fixed, the number of edges does not highly affect the efficiency of CCAS.

3. Effectiveness of graph reduction algorithm. In this experiment, we evaluate the effectiveness of HCGR for $k = 15$ by varying r from 3 to 6. Setting r=τ means executing HCGR multiple times with k incrementing from 3 up to τ, where $r = 0$ corresponds to locating the graph into its $(k-1)$-core without using HCGR. As shown in Figure 10, we observe that: (1) Using HCGR significantly reduces the running time compared to not using it, achieving up to a 1,00× speedup on the IT dataset, while also pruning more vertices from the graph. (2) For HCGR, as the r increases, there is no huge decrease in running time and the number of remaining vertices. This indicates that setting $r = 3$ is sufficient to eliminate almost all the vertices that are not in the CDS. The results on other datasets are shown in our technical report

4.Parallelization of CCAS**.** We have implemented a parallel version of CCAS by paralleling the process of constructing the SCT and enumerating paths in the vertex update, denoted by CCAS-Par. In CCAS-Par, during the SCT building stage, each thread is assigned to construct a sub-tree of the SCT, while during the vertex weight updating stage, each thread is scheduled for updating the vertex weights within one path. The running time results of CCAS-Par, evaluated by varying the number of threads from 1 to 16 across five datasets, are presented in our technical report [64]. Clearly, as the number of threads increases, the overall runtime of CCAS-Par decreases, demonstrating strong parallel scalability. For example, on the AC and IT dataset, using 16 threads allows CCAS-Par to achieve self-speedups of 12 times.

8 RELATED WORKS

In this section, we first review the existing works of DSD problems, including EDS/CDS probelms and their variants, and then briefly review the related works of dense subgraph discovery.

• **EDS/CDS problems.** EDS problem aims to find the subgraph with the maximum average degree [2, 4–7, 10, 11, 25, 34, 35, 47, 59, 69, 73]. This problem can be solved by solving a parametric maximum-flow problem [34] with binary search. In general, exact EDS solutions are suitable for small graphs, but their performance declines for larger graphs. Consequently, researchers have turned to approximation algorithms [6, 13, 14, 28, 35, 43, 59] to enhance efficiency. The peeling algorithm for k-core decomposition runs in linear time and provides a 2-approximation [13]. The EDS problem can be formulated as a convex programming and solved by the linear programming solver [14, 23, 35, 39, 71]. Recently, Zhou et al. provided a comprehensive benchmark [92] for the EDS problem. The EDS problem has been generalized to the CDS problem, which can detect "near-clique" subgraphs [27, 28, 36, 50, 57, 76]. Notably, when $k = 2$, this problem reduces to the EDS problem. The maximum flow-based algorithm is extended to solve this problem [28, 57, 76]. Besides, the convex programming-based algorithms [14, 36, 71, 91] have been studied, which are extensively reviewed in Section 3. For more details, please refer to the recent survey of DSD [46, 50].

• **Variants of EDS/CDS problems.** Many variants of EDS problem have been studied [3, 20, 29, 54, 63, 69, 81]. The EDS problem has been extended for directed graph, which finds the directed densest subgraph [43, 51, 55, 56]. The densest k-subgraph problem (DkS) aims to maximize the number of edges in a subgraph with k vertices, which is NP-hard [29]. The top-k locally densest subgraph discovery problems find the locally dense regions [54, 63, 75]. The anchored densest subgraph problems [20, 21, 83] aim to maximize R-subgraph density of the subgraphs containing an anchored node set. Recently, the fair densest subgraph problem and diverse densest subgraph problems [1, 58, 60] have been explored to achieve equitable outcomes and overcome algorithmic bias. Besides, the variants of CDS problem have also been studied, such as pattern-based densest subgraph [28] and triangle densest subgraph [79].

• **Dense subgraph discovery.** Another group of works highly related to DSD is about dense subgraph discovery. Many cohesive subgraph models like k-core [9, 70], k-truss [18, 67, 87], k-ECC [38, 86], k-clique [22], quasi-clique [77, 78], and k-plex [8, 93] have been studied, which have found various applications in community search [26]. Besides, these models are extended to other types of graphs [48, 52, 89, 90]. Nevertheless, these works are different from DSD since they do not use the density definition as a key metric.

9 CONCLUSIONS

In this paper, we investigate the problem of efficient k-clique densest subgraph (CDS) discovery. Among existing CDS solutions, algorithms that perform well in practice often have weaker theoretical guarantees, while those with strong theoretical assurances tend to perform worse in practice. To improve the practical efficiency, we introduce a novel graph reduction technique that locates the CDS into a small subgraph with non-trivial theoretical guarantees. We further propose a new efficient approximation algorithm by employing the SOTA k-clique counting algorithm, achieving both strong practical efficiency and theoretical guarantees. Our experimental results on 14 real-world large graphs show that our proposed algorithm is highly efficient and achieves up to four orders of magnitude faster than the SOTA algorithms. In the future, we plan to design GPU-friendly algorithms for the CDS problem, explore I/O-efficient and distributed solutions, investigate time and memory-efficient CDS algorithms in very dense graphs, and develop efficient methods for CDS in large dynamic graphs.

10 ACKNOWLEDGMENTS

This work was supported in part by Guangdong Talent Program under Grant 2021QN02X826, Guangdong Provincial Key Laboratory of Mathematical Foundations for Artificial Intelligence (2023B1212010 001), Shenzhen Science and Technology Program under Grants JCYJ20220530143602006 and ZDSYS 20211021111415025, and Shenzhen Research Institute of Big Data under Grant SIF20240002.

REFERENCES

[1] Aris Anagnostopoulos, Luca Becchetti, Adriano Fazzone, Cristina Menghini, and Chris Schwiegelshohn. 2020. Spectral relaxations and fair densest subgraphs. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 35–44.

[2] Venkat Anantharam and Justin Salez. 2016. The densest subgraph problem in sparse random graphs. (2016).

[3] Reid Andersen and Kumar Chellapilla. 2009. Finding dense subgraphs with size bounds. In *International workshop on algorithms and models for the web-graph*. Springer, 25–37.

[4] Albert Angel, Nick Koudas, Nikos Sarkas, Divesh Srivastava, Michael Svendsen, and Srikanta Tirthapura. 2014. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *The VLDB journal* 23 (2014), 175–199.

[5] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. 2002. Complexity of finding dense subgraphs. *Discrete Applied Mathematics* 121, 1-3 (2002), 15–26.

[6] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest subgraph in streaming and mapreduce. *arXiv preprint arXiv:1201.6567* (2012).

[7] Oana Denisa Balalau, Francesco Bonchi, TH Hubert Chan, Francesco Gullo, and Mauro Sozio. 2015. Finding subgraphs with maximum total density and limited overlap. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. 379–388.

[8] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V Hicks. 2011. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research* 59, 1 (2011), 133–142.

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

[10] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. 2015. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 173–182.

[11] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos Tsourakakis, Di Wang, and Junxing Wang. 2020. Flowless: Extracting densest subgraphs without flow computations. In *Proceedings of The Web Conference 2020*. 573–583.

[12] Lijun Chang. 2019. Efficient maximum clique computation over large sparse graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 529–538.

[13] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International workshop on approximation algorithms for combinatorial optimization*. Springer, 84–95.

[14] Chandra Chekuri, Kent Quanrud, and Manuel R Torres. 2022. Densest subgraph: Supermodularity, iterative peeling, and flow. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1531–1555.

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

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

[17] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355.

[18] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008).

[19] Guangyu Cui, Yu Chen, De-Shuang Huang, and Kyungsook Han. 2008. An algorithm for finding functional modules and protein complexes in protein-protein interaction networks. *Journal of Biomedicine and Biotechnology* 2008 (2008).

[20] Yizhou Dai, Miao Qiao, and Lijun Chang. 2022. Anchored densest subgraph. In *Proceedings of the 2022 International Conference on Management of Data*. 1200–1213.

[21] Yizhou Dai, Miao Qiao, and Rong-Hua Li. 2024. On Density-based Local Community Search. *Proceedings of the ACM on Management of Data* 2, 2 (2024), 1–25.

[22] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference*. 589–598.

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

[24] Xiaoxi Du, Ruoming Jin, Liang Ding, Victor E Lee, and John H Thornton Jr. 2009. Migration motif: a spatial-temporal pattern mining approach for financial markets. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[25] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th international conference on world wide web*. 300–310.

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

[27] Yixiang Fang, Wensheng Luo, and Chenhao Ma. 2022. Densest subgraph discovery on large graphs: Applications, challenges, and techniques. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3766–3769.

[28] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks VS Lakshmanan, and Xuemin Lin. 2019. Efficient algorithms for densest subgraph discovery. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1719–1732.

[29] Uriel Feige, Michael Seltser, et al. 1997. *On the densest k-subgraph problem*. Citeseer.

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

[31] David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*. 721–732.

[32] Aristides Gionis, Flavio PP Junqueira, Vincent Leroy, Marco Serafini, and Ingmar Weber. 2013. Piggybacking on social networks. In *VLDB 2013-39th International Conference on Very Large Databases*, Vol. 6. 409–420.

[33] Aristides Gionis and Charalampos E Tsourakakis. 2015. Dense subgraph discovery: Kdd 2015 tutorial. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2313–2314.

[34] Andrew V Goldberg. 1984. Finding a maximum density subgraph. (1984).

[35] Elfarouk Harb, Kent Quanrud, and Chandra Chekuri. 2022. Faster and scalable algorithms for densest subgraph and decomposition. *Advances in Neural Information Processing Systems* 35 (2022), 26966–26979.

[36] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2023. Scaling Up k-Clique Densest Subgraph Detection. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[37] Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. 2005. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics* 21, suppl_1 (2005), i213–i221.

[38] Jiafeng Hu, Xiaowei Wu, Reynold Cheng, Siqiang Luo, and Yixiang Fang. 2016. Querying minimal steiner maximum-connected subgraphs in large graphs. In *CIKM*. 1241–1250.

[39] Martin Jaggi. 2013. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *International conference on machine learning*. PMLR, 427–435.

[40] Shweta Jain and C Seshadhri. 2020. The power of pivoting for exact clique counting. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 268–276.

[41] Shweta Jain and C Seshadhri. 2020. Provably and efficiently approximating near-cliques using the Turán shadow: PEANUTS. In *Proceedings of The Web Conference 2020*. 1966–1976.

[42] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 2009. 3-hop: a high-compression indexing scheme for reachability query. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 813–826.

[43] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International colloquium on automata, languages, and programming*. Springer, 597–608.

[44] Konect. 2006. Konect. http://konect.cc/networks/.

[45] Laks VS Lakshmanan. 2022. On a Quest for Combating Filter Bubbles and Misinformation. In *SIGMOD*. 2–2.

[46] Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. 2023. A survey on the densest subgraph problem and its variants. *arXiv preprint arXiv:2303.14467* (2023).

[47] Victor E Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. 2010. A survey of algorithms for dense subgraph discovery. *Managing and mining graph data* (2010), 303–336.

[48] Wensheng Luo, Yixiang Fang, Chunxu Lin, and Yingli Zhou. 2024. Efficient Parallel D-Core Decomposition at Scale. *Proceedings of the VLDB Endowment* 17, 10 (2024), 2654–2667.

[49] Wensheng Luo, Kenli Li, Xu Zhou, Yunjun Gao, and Keqin Li. 2022. Maximum Biplex Search over Bipartite Graphs. In *ICDE*. IEEE, 898–910.

[50] Wensheng Luo, Chenhao Ma, Yixiang Fang, and Laks VS Lakshman. 2023. A Survey of Densest Subgraph Discovery on Large Graphs. *arXiv preprint arXiv:2306.07927* (2023).

[51] Wensheng Luo, Zhuo Tang, Yixiang Fang, Chenhao Ma, and Xu Zhou. 2023. Scalable algorithms for densest subgraph discovery. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 287–300.

[52] Wensheng Luo, Qiaoyuan Yang, Yixiang Fang, and Xu Zhou. 2023. Efficient core maintenance in large bipartite graphs. *Proceedings of the ACM on Management of Data* 1, 3 (2023), 1–26.

[53] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum biclique search at billion scale. *PVLDB* 13, 9 (2020), 1359–1372.

[54] Chenhao Ma, Reynold Cheng, Laks VS Lakshmanan, and Xiaolin Han. 2022. Finding locally densest subgraphs: a convex programming approach. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2719–2732.

[55] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, and Xiaolin Han. 2022. A convex-programming approach for efficient directed densest

subgraph discovery. In *Proceedings of the 2022 International Conference on Management of Data*. 845–859.

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

[57] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. 2015. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 815–824.

[58] Atsushi Miyauchi, Tianyi Chen, Konstantinos Sotiropoulos, and Charalampos E Tsourakakis. 2023. Densest Diverse Subgraphs: How to Plan a Successful Cocktail Party with Diversity. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1710–1721.

[59] Ta Duy Nguyen and Alina Ene. 2024. Multiplicative Weights Update, Area Convexity and Random Coordinate Descent for Densest Subgraph Problems. In *Forty-first International Conference on Machine Learning*.

[60] Lutz Oettershagen, Honglian Wang, and Aristides Gionis. 2024. Finding Densest Subgraphs with Edge-Color Constraints. In *Proceedings of the ACM on Web Conference 2024*. 936–947.

[61] Laboratory of Web Algorithmics. 2013. Laboratory of Web Algorithmics Datasets. http://law.di.unimi.it/datasets.php.

[62] Stanford Network Analysis Project. 2009. SNAP. http://snap.stanford.edu/data/.

[63] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally densest subgraph discovery. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 965–974.

[64] The Technique Report. 2025. Efficient k-Clique Densest Subgraph Discovery: Towards Bridging Practice and Theory (technical report). https://github.com/forxenn/ccas/blob/main/TechnicalReport.pdf.

[65] Network Repository. 2014. Network Repository. https://networkrepository.com/network-data.php.

[66] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. 2010. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Research in Computational Molecular Biology: 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings 14*. Springer, 456–472.

[67] Kazumi Saito, Takeshi Yamada, and Kazuhiro Kazama. 2008. Extracting communities from complex networks by the k-dense method. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 91, 11 (2008), 3304–3311.

[68] Raman Samusevich, Maximilien Danisch, and Mauro Sozio. 2016. Local triangle-densest subgraphs. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 33–40.

[69] Saurabh Sawlani and Junxing Wang. 2020. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 181–193.

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

[71] Bintao Sun, Maximilien Danisch, TH Hubert Chan, and Mauro Sozio. 2020. Kclist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. *Proceedings of the VLDB Endowment (PVLDB)* (2020).

[72] Brian K Tanner, Gary Warner, Henry Stern, and Scott Olechowski. 2010. Koobface: The evolution of the social botnet. In *2010 eCrime Researchers Summit*. IEEE, 1–10.

[73] Nikolaj Tatti and Aristides Gionis. 2015. Density-friendly graph decomposition. In *Proceedings of the 24th International Conference on World Wide Web*. 1089–1099.

[74] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science* 363, 1 (2006), 28–42.

[75] Tran Ba Trung, Lijun Chang, Nguyen Tien Long, Kai Yao, and Huynh Thi Thanh Binh. 2023. Verification-free approaches to efficient locally densest subgraph discovery. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1–13.

[76] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*. 1122–1132.

[77] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. 2013. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 104–112.

[78] Charalampos E Tsourakakis. 2014. Mathematical and algorithmic analysis of network and biological data. *arXiv preprint arXiv:1407.0375* (2014).

[79] Charalampos E Tsourakakis. 2014. A novel approach to finding near-cliques: The triangle-densest subgraph problem. *arXiv preprint arXiv:1405.1477* (2014).

[80] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2024. Efficient k-Clique listing: an edge-oriented branching strategy. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–26.

[81] Yichen Xu, Chenhao Ma, Yixiang Fang, and Zhifeng Bao. 2023. Efficient and Effective Algorithms for Generalized Densest Subgraph Discovery. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[82] Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongzhi Chen, and Guoren Wang. 2022. Lightning Fast and Space Efficient k-clique Counting. In *Proceedings of the ACM Web Conference 2022*. 1191–1202.

[83] Xiaowei Ye, Rong-Hua Li, Lei Liang, Zhizhen Liu, Longlong Lin, and Guoren Wang. 2024. Efficient and Effective Anchored Densest Subgraph Search: A Convex-programming based Approach. (2024), xxx–xxx.

[84] Xiaowei Ye, Miao Qiao, Rong-Hua Li, Qi Zhang, and Guoren Wang. 2024. Scalable k-clique Densest Subgraph Search. *arXiv preprint arXiv:2403.05775* (2024).

[85] Kaiqiang Yu and Cheng Long. 2021. Graph Mining Meets Fake News Detection. In *Data Science for Fake News: Surveys and Perspectives*. Springer, 169–189.

[86] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2017. I/O efficient ECC graph decomposition via graph reduction. *The VLDB Journal* 26, 2 (2017), 275–300.

[87] Yang Zhang and Srinivasan Parthasarathy. 2012. Extracting analyzing and visualizing triangle k-core motifs within networks. In *2012 IEEE 28th international conference on data engineering*. IEEE, 1049–1060.

[88] Feng Zhao and Anthony KH Tung. 2012. Large scale cohesive subgraphs discovery for social network visual analysis. *Proceedings of the VLDB Endowment* 6, 2 (2012), 85–96.

[89] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential community search over large heterogeneous information networks. *VLDB* 16, 8 (2023), 2047–2060.

[90] Yingli Zhou, Yixiang Fang, Chenhao Ma, Tianci Hou, and Xin Huang. 2024. Efficient Maximal Motif-Clique Enumeration over Large Heterogeneous Information Networks. *Proceedings of the VLDB Endowment* 17, 11 (2024), 2946–2959.

[91] Yingli Zhou, Qingshuo Guo, Yixiang Fang, and Chenhao Ma. 2024. A Counting-based Approach for Efficient k-Clique Densest Subgraph Discovery. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[92] Yingli Zhou, Qingshuo Guo, Yi Yang, Yixiang Fang, Chenhao Ma, and Laks Lakshmanan. 2024. In-depth Analysis of Densest Subgraph Discovery in a Unified Framework. *arXiv preprint arXiv:2406.04738* (2024).

[93] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. 2021. Improving maximum k-plex solver via second-order reduction and graph color bounding. In *AAAI*, Vol. 35. 12453–12460.