



On flexible cohesive subgraph mining

Dandan Liu¹ · Zhaonian Zou¹

Received: 1 February 2021 / Revised: 1 February 2021 / Accepted: 27 April 2021 /
Published online: 5 June 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

When characterizing the cohesion of a subgraph, many cohesive subgraph models impose various constraints on the relationship between the minimum degree of vertices and the number of vertices in the subgraph. However, the constraints imposed by the clique, quasi-clique and k -core models are so rigid that they cannot simultaneously characterize cohesive subgraphs of various scales in a graph. This paper characterizes the flexibility of a cohesive subgraph model by the constraint it imposes on this relationship and proposes f -constraint core (f -CC), a new model that can flexibly characterize cohesive subgraphs of different scales. We formalize the FlexCS problem that finds r representative f -CCs in a graph that are most diversified. This problem is NP-hard and is even NP-hard to be approximated within $|V|^{1-\epsilon}$ for any $\epsilon > 0$. To solve the problem efficiently, a divide-and-conquer algorithm is proposed together with some effective techniques including f -CC generation, sub-problem filtering, early termination, and index-based connected component retrieval. Extensive experiments verify the flexibility of the f -CC model and the effectiveness of the proposed algorithm.

Keywords Graph data · Cohesive subgraph · Flexibility · Diversity

1 Introduction

Cohesive subgraph mining aims at finding cohesively connected vertices in a massive graph. It plays important roles in many applications such as community detection, graph compression and graph visualization [8]. So far, a considerable number of cohesive subgraph models have been proposed in the literature, e.g. *clique* [12], *quasi-clique* [13], *k-core* [17], *k-truss* [6], and *nucleu* [16].

This article belongs to the Topical Collection: *Special Issue on Large Scale Graph Data Analytics*
Guest Editors: Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang

✉ Zhaonian Zou
zouzou@hit.edu.cn

Dandan Liu
ddliu@hit.edu.cn

¹ Harbin Institute of Technology, Harbin, China

When characterizing the cohesion of a subgraph, many cohesive subgraph models commonly consider two key features, namely the *minimum degree of vertices* and the *number of vertices* in the subgraph. Different models often impose different constraints on the relationship between these two features. To be more specific, let us first review some well-known cohesive subgraph models in the literature and take a deeper look at their respective constraints imposed on the relationship.

Let $G = (V, E)$ be a graph, where V is the set of vertices, and E is the set of edges. Two vertices u and v are adjacent if $(u, v) \in E$. The subgraph of G induced by a vertex subset $Q \subseteq V$, denoted by $G[Q]$, is $(Q, E[Q])$, where $E[Q]$ is the set of edges with both endpoints in Q . Let us consider three classical cohesive subgraph models, namely *clique*, *quasi-clique* and *k-core*.

- A *clique* [12] in G is a set of vertices where every two vertices are adjacent to each other. Each vertex in a clique Q is of degree $|Q| - 1$ in $G[Q]$.
- A (degree-based) γ -*quasi-clique* [13] in G is a subset of vertices $Q \subseteq V$ where every vertex is adjacent to at least $\lceil \gamma \cdot (|Q| - 1) \rceil$ other vertices, where $\gamma \in [0, 1]$. Specially, a clique is a γ -quasi-clique for $\gamma = 1$.
- A k -*core* [17] in G is a subset of vertices $Q \subseteq V$ where every vertex is of degree at least k in $G[Q]$.

Figure 1 illustrates the relationships between the minimum degree of vertices and the number of vertices in a cohesive subgraph restricted by these models. For the clique and quasi-clique models, the minimum degree is required to be linear to the number of vertices, which is obviously very strict. In practice, it is nearly impossible to find a cohesive subgraph with hundreds of vertices in a real graph under these two models due to the sparsity of real graphs. Thus, the clique and quasi-clique models are only suitable for modeling small cohesive subgraphs. On the contrary, the k -core model imposes a very loose constraint: the minimum vertex degree is required to be at least k , which is independent of the number of vertices. In fact, in a real graph, k -cores are generally large and sparsely connected, and therefore, this model is only suitable for modeling large cohesive subgraphs.

Inspired by these observations, we characterize the *flexibility* of a cohesive subgraph model by the constraint it imposes on the relationship between the minimum degree of

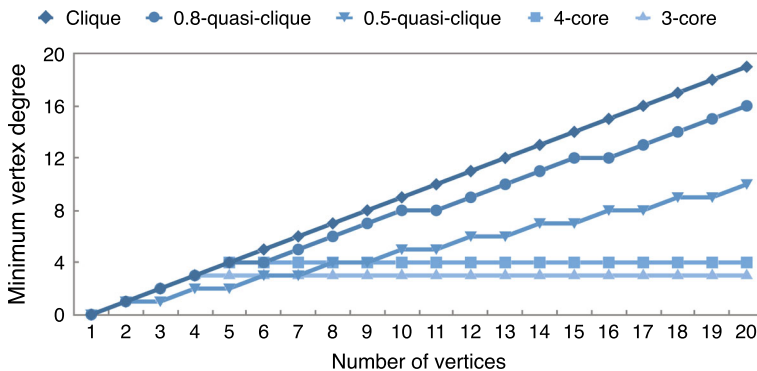


Figure 1 Relationships between the minimum degree of vertices and the number of vertices in a cohesive subgraph under various cohesive subgraph models

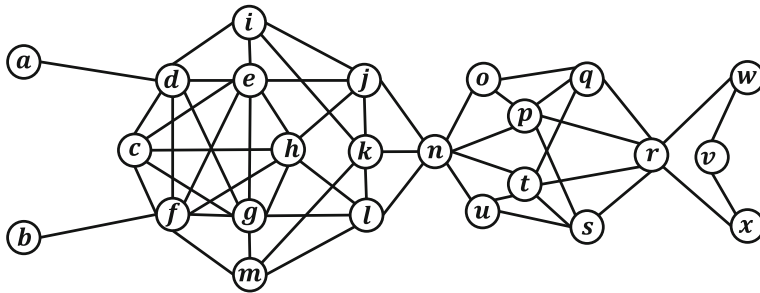


Figure 2 An example graph

vertices and the number of vertices. In fact, the clique/quasi-clique model and the k -core model stay at two extremes of flexibility. The rigidity of these models prevents them from simultaneously characterizing cohesive subgraphs varying from small scales to large scales, so each model is only suitable for a limited scope of applications. Therefore, we focus on *flexible cohesive subgraph models* which can be applied to flexibly model cohesive subgraphs in a wide range of applications. To the best of our knowledge, no prior work has been done on this problem in the literature.

This paper proposes *f -constraint core (f -CC)*, a new cohesive subgraph model. Given a graph $G = (V, E)$ and a monotonically increasing function $f : \mathbb{N} \rightarrow \mathbb{R}$, a subset of vertices $Q \subseteq V$ is an f -CC if every vertex in $G[Q]$ is adjacent to at least $\lceil f(|Q|) \rceil$ other vertices. Rather than a special new model, the f -CC model is a generalization of the clique, quasi-clique and k -core models. When $f(x) = x - 1$, an f -CC is a clique; When $f(x) = \gamma(x - 1)$ where $\gamma \in [0, 1]$, an f -CC is a γ -quasi-clique; When $f(x) = k$ where $k \in \mathbb{N}$, an f -CC is a k -core. Section 6.2 shows that by carefully specifying f , the f -CC model attains more flexibility than other classical models mentioned above. Here, we illustrate the flexibility of the f -CC model by an example.

Example 1 Consider the graph shown in Figure 2. First, we depict the constraints imposed by the clique, \sqrt{x} -CC and 3-core models in Figure 3. For the \sqrt{x} -CC model, as the number of vertices increases, the required minimum vertex degree increases more and more slowly, which is consistent with users’ expectations on cohesive subgraphs: When a cohesive subgraph is small (like a micro-cluster), the constraint on the minimum vertex degree should be strict; when it is large (like a community), this constraint should be relaxed.

Moreover, we illustrate some cohesive subgraphs characterized by these models, particularly, the maximum clique, two maximal \sqrt{x} -CCs and the maximal 3-core. Obviously, there are two groups of cohesively connected vertices in the graph, namely $S_1 = \{c, d, e, f, g, h, i, j, k, l, m\}$ and $S_2 = \{n, o, p, q, r, s, t, u\}$. The maximum clique only contains partial vertices in S_1 and excludes the vertices i, j, k, l, m that are also cohesively connected to the maximum clique. The maximal 3-core merges S_1 and S_2 . This is also not reasonable because S_1 and S_2 are only linked through the cut vertex n . The \sqrt{x} -CC model circumvents the shortcomings of the clique and k -core models. Two maximal \sqrt{x} -CCs shown in Figure 3 precisely characterize S_1 and S_2 , respectively.

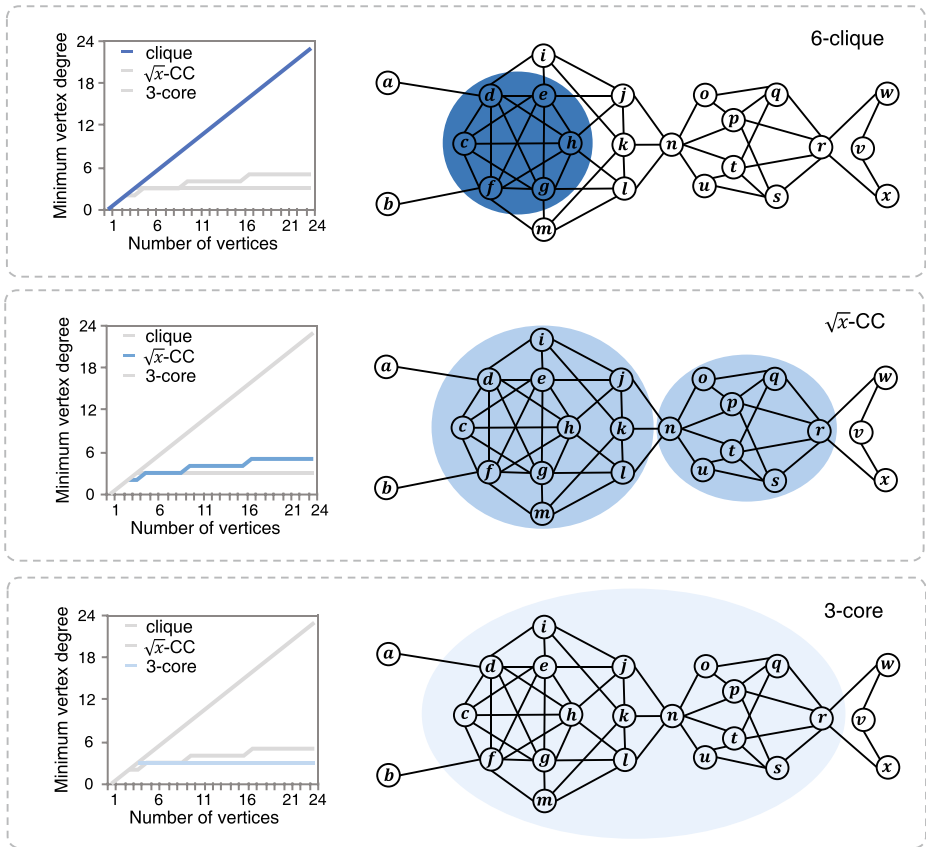


Figure 3 Relationships between the minimum degree of vertices and the number of vertices restricted by the clique model, the \sqrt{x} -CC model and the 3-core model, and the cohesive subgraphs characterized by these models

As with cliques and quasi-cliques, mining all f -CCs on a large graph is often a tough task due to the following reasons: (1) The number of all f -CCs can be exponential to the number of vertices in the graph; (2) There always exists huge overlap between f -CCs in the graph, which will be illustrated by the following example.

Example 2 In the example graph shown in Figure 2, there are a large number of \sqrt{x} -CCs, e.g. $\{c, d, e, f, g, h, i, j, k, l, m\}$, $\{d, e, g, h, j, k, l, n\}$, $\{e, f, g, h, j, k, l, n\}$, and so on. As illustrated in Figure 4, they are overlapped with each other significantly.

To avoid the huge but uninformative result set of the enumeration paradigm, [9, 19, 21] provide a way to formalize the problem that extracts not only individually large but also lowly overlapped cohesive subgraphs. They characterize the quality of the result set by its *diversity*, i.e., the number of vertices it covers, and extract r cohesive subgraphs that are most diversified. Motivated by this, we formalize the flexible cohesive subgraph mining (FlexCS) problem with respect to our f -CC model: We extract r representative f -CCs that attain the highest diversity, i.e., these f -CCs can cover the largest number of vertices. We

show that the FlexCS problem is NP-hard and is even NP-hard to be approximated within $|V|^{1-\epsilon}$ for any $\epsilon > 0$.

Due to the extremely high computational complexity of the FlexCS problem, we develop a heuristic algorithm that extracts a collection of qualified f -CCs in a divide-and-conquer manner and selects r representative f -CCs on the fly that approximately attain the highest diversity. To this end, the constraint on the relationship between the minimum vertex degree and the number of vertices is equivalently translated into a series of simple constraints. Based on these simple constraints, the FlexCS problem on the input graph G is decomposed into a series of sub-problems called ConsCS on the connected components of the maximal k -cores of G . An advanced filtering method is proposed to eliminate a portion of ConsCS problem instances that are unpromising to produce any informative f -CCs to improve the diversity of the result set. The remaining ConsCS problem instances are handled in different ways depending on the size of the input connected component. Problem instances on small connected components are easy to solve. For problem instances on large connected components, we develop a heuristic algorithm to generate qualified f -CCs. This algorithm takes both greediness and randomness into the f -CC generation process. In addition, we propose an early termination mechanism and an index-based connected component retrieval method to speed up the algorithm for the FlexCS problem.

The main contributions of this paper can be summarized as follows:

- We propose the f -CC model that can flexibly characterize cohesive subgraphs of various scales and formalize the FlexCS problem that extracts r representative f -CCs attaining the highest diversity.
- We prove the NP-hardness and the inapproximability of the FlexCS problem and develop a divide-and-conquer approach to solving it efficiently.
- Extensive experiments were conducted to verify that the f -CC model is flexible in simultaneously characterizing cohesive subgraphs varying from small scales to large scales, and the proposed algorithm can efficiently discover top- r f -CCs that approximately have the highest diversity.

The paper is organized as follows. Section 2 defines the f -CC model and formalizes the FlexCS problem. Section 3 presents the main procedure of our algorithm, followed by the important subroutines described in Section 4 and the optimization techniques in Section 5. The experiments are reported in Section 6. The related work is reviewed in Section 7. Finally, the paper is concluded in Section 8.

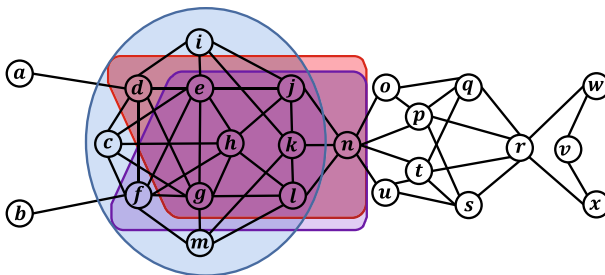


Figure 4 The f -CCs $\{c, d, e, f, g, h, i, j, k, l, m\}$, $\{d, e, g, h, j, k, l, n\}$ and $\{e, f, g, h, j, k, l, n\}$ have significant overlaps

2 Model and problem definition

This section introduces the f -constraint core (f -CC) model and the flexible cohesive subgraph mining (FlexCS) problem. In this paper, we only consider simple undirected graphs. Let $G = (V, E)$ be a graph. The *degree* of a vertex $v \in V$ in G , denoted by $d_G(v)$, is the number of edges adjacent to v in G . The neighborhood of a vertex $v \in V$ in G , denoted by $N_G(v)$, is the set of vertices connected to v by an edge in G . We have $d_G(v) = |N_G(v)|$. For a vertex subset $Q \subseteq V$, the subgraph of G induced by Q , denoted by $G[Q]$, is $(Q, E[Q])$, where $E[Q]$ is the set of edges with both endpoints in Q .

We first define the notion of f -constraint core (f -CC) that can flexibly characterize cohesive subgraphs of various scales simultaneously in a graph.

Definition 1 (f -constraint core) Given a graph $G = (V, E)$ and a monotonically increasing function $f : \mathbb{N} \rightarrow \mathbb{R}$, a subset of vertices $Q \subseteq V$ is an f -constraint core (f -CC for short) if the degree of every vertex in the induced subgraph $G[Q]$ is at least $\lceil f(|Q|) \rceil$.

Since connectivity is often expected in research and practice, we require that $G[Q]$ is connected. Note that the notion of f -CC is a generalization of the notions of clique, quasi-clique and k -core. Specifically,

- For $f(x) = x - 1$, an f -CC is a clique.
- For $f(x) = \gamma(x - 1)$ where $0 \leq \gamma \leq 1$, an f -CC is a γ -quasi-clique.
- For $f(x) = k$ where $k \in \mathbb{N}$, an f -CC is a k -core.

Let $F_f(G)$ be the set of all f -CCs in G . As mentioned in Section 1, $|F_f(G)|$ could be exponential to the size of G , so we usually cannot expect to retrieve all f -CCs in G . In addition, f -CCs in G often overlap with each other, so it is not a rational choice to return all f -CCs in $F_f(G)$ due to the huge redundancy. Therefore, we try to find a small number of informative f -CCs that are qualified to represent $F_f(G)$ as the existing work [9, 19, 21] did. Similar to [9, 19, 21], we use the diversity (i.e., the size of the coverage of the result set defined in Definition 2) to measure the quality of the result set.

Definition 2 (coverage) Given a set of f -CCs $\mathcal{R} = \{Q_1, Q_2, \dots\}$ in graph G , the coverage of \mathcal{R} , denoted by $\text{cov}(\mathcal{R})$, is the set of vertices in G covered by f -CCs in \mathcal{R} , i.e., $\text{cov}(\mathcal{R}) = \bigcup_{Q \in \mathcal{R}} Q$.

Apparently, the smaller $|\text{cov}(F_f(G)) - \text{cov}(\mathcal{R})|$ is, the better \mathcal{R} approximates $F_f(G)$. Thus, the problem in this paper called the *flexible cohesive subgraph mining* (FlexCS) can be formally defined as follows.

Problem Definition 1 Given a graph $G = (V, E)$, a monotonically increasing function $f : \mathbb{N} \rightarrow \mathbb{R}$ and $r \in \mathbb{N}$, the goal of the flexible cohesive subgraph mining (FlexCS) problem is to find a subset $\mathcal{R} \subseteq F_f(G)$ such that $|\mathcal{R}| = r$ and $|\text{cov}(\mathcal{R})|$ is maximized (i.e., $|\text{cov}(F_f(G)) - \text{cov}(\mathcal{R})|$ is minimized).

Example 3 Given the example graph shown in Figure 2, $f(x) = \sqrt{x}$ and $r = 2$, two \sqrt{x} -CCs $\{c, d, e, f, g, h, i, j, k, l, m\}$ and $\{n, o, p, q, r, s, t, u\}$ shown in Figure 3 are returned as \mathcal{R} , and $|\text{cov}(\mathcal{R})| = 19$, which is maximized.

Theorem 1 *The FlexCS problem is NP-hard and is even NP-hard to be approximated within $|V|^{1-\epsilon}$ for all $\epsilon > 0$.*

Proof Let η -APPROX- Π denote the approximation version of an optimization problem Π with approximation ratio η . Since an f -CC is a clique when $f(x) = x - 1$, the maximum clique problem (MCP) and the η -APPROX-MCP problem are special cases of FlexCS and η -APPROX-FlexCS, respectively. Therefore, MCP and η -APPROX-MCP are polynomial-time many-to-one reducible to FlexCS and η -APPROX-FlexCS, respectively. Due to the NP-hardness of MCP [10], FlexCS is also NP-hard. In addition, for all $\epsilon > 0$, it is NP-hard to approximate MCP within $|V|^{1-\epsilon}$ [23], so does FlexCS. The theorem thus holds. \square

3 Our method

This section introduces our method to solve the FlexCS problem. Due to the huge search space of the enumeration process for enumerating all f -CCs and its exponential-scale output, we cannot extend the clique enumeration paradigm to enumerate all f -CCs in a graph G and then choose r of them that cover the largest number of vertices. Therefore, we develop a heuristic algorithm that first finds a set of qualified f -CCs $S \subseteq F_f(G)$ and then chooses r f -CCs attaining the highest diversity from S . In the rest of this section, we present two key components of our method, namely the qualified f -CCs generation (Section 3.1) and the diversified top- r f -CCs selection (Section 3.2).

3.1 Qualified f -CCs generation

The first component of our method is to find a set of f -CCs $S \subseteq F_f(G)$ that is qualified to represent $F_f(G)$, from which the diversified top- r f -CCs are further selected. To this end, we adopt a divide-and-conquer approach to generating S . The generation process is decomposed into solving a series of sub-problem instances, and a core-based filtering method is developed to eliminate some useless sub-problem instances that are unpromising to return informative f -CCs.

3.1.1 Problem decomposition

To begin with, we introduce a generalized inverse function of a monotonically increasing function F presented in [7] as follows:

$$F^{-1}(x) = \sup\{y | F(y) \leq x\} \quad (1)$$

Based on this generalized inverse function, we have the following theorem:

Theorem 2 *Given a graph $G = (V, E)$ and a monotonically increasing function $f : \mathbb{N} \rightarrow \mathbb{R}$, let $g : \mathbb{N} \rightarrow \mathbb{N}$ be a function such that $g(x) = \lceil f(x) \rceil$. For a vertex subset $Q \subseteq V$, the following conditions are equivalent:*

Condition 1. Q is an f -CC;

Condition 2. The minimum degree of vertices in $G[Q]$ is at least $g(|Q|)$;

Condition 3. $|Q| \leq g^{-1}(k)$, where k is the minimum degree of vertices in $G[Q]$ and g^{-1} is the generalized inverse function of g defined by (1).

Proof Condition 1 and Condition 2 are equivalent due to Definition 1. Condition 2 and Condition 3 are equivalent due to (1). According to the transitivity, the theorem holds. \square

Based on Theorem 2, the degree constraint imposed on vertices in an f -CC can be translated into the following pair of constraints parameterized by k :

\mathbf{C}_1^k (**Minimum degree constraint**): The degree of every vertex in $G[Q]$ is at least k .

\mathbf{C}_2^k (**Maximum size constraint**): $|Q| \leq g^{-1}(k)$.

For brevity, we use \mathbf{C}^k to represent both \mathbf{C}_1^k and \mathbf{C}_2^k . Certainly, an f -CC must satisfy \mathbf{C}^k for some $1 \leq k \leq |V| - 1$. Hence, we define the following sub-problem called the *constrained cohesive subgraph mining* (ConsCS) problem:

Problem Definition 2 Given a graph $G = (V, E)$, a monotonically increasing function $f : \mathbb{N} \rightarrow \mathbb{R}$ and $k \in [1, |V| - 1]$, the goal of the constrained cohesive subgraph mining (ConsCS) problem is to find the set of f -CCs in G satisfying \mathbf{C}^k .

Theorem 3 Given a graph G , for $k \in [1, |V| - 1]$, let $F_{f,k}(G)$ be the set of f -CCs satisfying constraint \mathbf{C}^k in G . We have

$$F_f(G) = \bigcup_{k=1}^{|V|-1} F_{f,k}(G). \quad (2)$$

Proof (1) Obviously, by definition, $\bigcup_{k=1}^{|V|-1} F_{f,k}(G) \subseteq F_f(G)$. (2) For each f -CC $Q \in F_f(G)$, let k be the minimum degree of vertices in $G[Q]$. Definitely, $1 \leq k \leq |V| - 1$. Based on Theorem 2, Q satisfies \mathbf{C}^k . Therefore, $Q \in F_{f,k}(G)$. Thus, we have $F_f(G) \subseteq \bigcup_{k=1}^{|V|-1} F_{f,k}(G)$. Therefore, the theorem holds. \square

According to Theorem 3, $F_f(G)$ can be obtained by solving each ConsCS problem instance $\langle G, f, k \rangle$ for $k = 1, 2, \dots, |V| - 1$. Note that, given a ConsCS instance $\langle G, f, k \rangle$, the number of f -CCs in G satisfying \mathbf{C}^k can be exponential to the size of G . It is nearly impossible to enumerate all such f -CCs in G . Therefore, we cannot merge all exact result sets of ConsCS instances as the qualified f -CCs set S and further select diversified top- r f -CCs from it. To tackle this, we develop a heuristic method to approximately extract representative f -CCs that attain high diversity (i.e., cover a large number of vertices) for solving the ConsCS problem, which will be described in Section 4. And then, we merge all these result sets of ConsCS instances to obtain S .

3.1.2 Elimination of ConsCS problem instances

Next, we present a filtering method to eliminate some ConsCS problem instances that are unpromising to produce informative f -CCs that have contributions to the diversity of the final result set. This method is based on the k -core model [17]: a set Q of vertices is a k -

core of a graph G if every vertex in $G[Q]$ is of degree at least k . A k -core is *maximal* if none of its proper supersets is a k -core. A key concept related to the k -core model is shown in Definition 3.

Definition 3 (core number) The core number of G , denoted by $\kappa(G)$, is the largest integer k such that the maximal k -core of G is nonempty. The core number of a vertex $v \in V$ in G , denoted by $\kappa_G(v)$, is the largest integer k such that there exists a nonempty maximal k -core containing v in G . $\kappa_G(v)$ can be simplified as $\kappa(v)$ if G is clear in the context.

The k -core model has two elegant properties:

- *Uniqueness*: The maximal k -core is unique.
- *Hierarchy*: The maximal k -core is a subset of the maximal k' -core if $k' \leq k$.

In the following, we use K_k^G to denote the maximal k -core of G . When G is clear in the context, K_k^G can be simplified as K_k .

Lemma 1 Any f -CC satisfying the constraint \mathbf{C}^k is contained in a connected component of the maximal k -core K_k of G .

Proof The lemma holds due to the minimum degree constraint \mathbf{C}_1^k . □

Based on Lemma 1, the ConsCS problem instance $\langle G, f, k \rangle$ can be solved on $G[K_k]$ instead of on G . Moreover, since f -CCs are connected, we can solve the instance $\langle G[K_k], f, k \rangle$ on each connected component of $G[K_k]$ independently.

Lemma 2 Let C be a connected component of the maximal k -core K_k of G . Given the ConsCS problem instance $\langle G[C], f, k \rangle$, if $|C| \leq g^{-1}(k)$, C is an f -CC in the result and is a superset of any other f -CC in the result.

Proof The lemma holds due to the maximum size constraint \mathbf{C}_2^k and Lemma 1. □

Since the goal of our FlexCS problem is to extract representative f -CCs that attain high diversity and avoid uninformative ones, it is not reasonable to include two f -CCs with inclusion relationship in the result set. Accordingly, also for speeding up the proceeding diversified top- r f -CCs selection, it is necessary to keep S succinct, i.e., for two f -CCs Q_1 and Q_2 in S , if $Q_1 \subseteq Q_2$, Q_1 can be removed from S without sacrificing diversity but making S smaller. According to Lemma 2, if there exists a connected component C in $G[K_k]$ with $|C| \leq g^{-1}(k)$, C is certainly an f -CC, and any f -CC in $G[C]$ is a subset of C . If C is already contained in S , any subset of C can be removed from S . All ConsCS problem instances $\langle G[C'], f, k' \rangle$ for $C' \subseteq C$ and $k \leq k'$ can thus be eliminated. Consequently, we have the following **core-based filtering** method:

1. We compute all nonempty maximal k -cores in G for $k \geq 1$ by the fast core decomposition algorithm [3].
2. We deal with each maximal k -core in ascending order of k . For each maximal k -core K_k with $1 \leq k \leq \kappa(G)$, we first compute its connected components. Then, for each

connected component C , if $|C| \leq g^{-1}(k)$, we add C into S and remove all vertices in C from the k' -cores for $k' \geq k$.

Now, we only need to solve the ConsCS problem instances on each remaining connected component of each nonempty maximal k -core after applying the core-based filtering.

Example 4 Consider the example graph shown in Figure 2 and $f(x) = \sqrt{x}$. When $k = 4$, the maximum size constraint w.r.t. k is $|Q| \leq g^{-1}(k) = 16$. Since the only one connected component of the maximal 4-core is $K_4 = \{c, d, e, f, g, h, i, j, k, l, m\}$ with size 11, K_4 is an f -CC. We add K_4 into S and all ConsCS problem instances $\langle G[C'], f, k' \rangle$ for $C' \subseteq K_4$ and $k' \geq 4$ can be eliminated.

3.2 Diversified top- r f -CCs selection

Algorithm 1 Update.

Input: The temporary result set \mathcal{R} and a newly discovered f -CC Q

Output: The updated result set \mathcal{R}

- 1: **if** $|\mathcal{R}| < r$ **then**
 - 2: $\mathcal{R} \leftarrow \mathcal{R} \cup \{Q\}$
 - 3: **else if** $|\mathcal{R}| = r$ **then**
 - 4: $Q^* \leftarrow \arg \min_{Q' \in \mathcal{R}} |\text{cov}(\mathcal{R}) - \text{cov}(\mathcal{R} - \{Q'\})|$
 - 5: **if** $|\text{cov}(\mathcal{R} - \{Q^*\}) \cup Q| \geq \left(1 + \frac{1}{r}\right) |\text{cov}(\mathcal{R})|$ **then**
 - 6: $\mathcal{R} \leftarrow (\mathcal{R} - \{Q^*\}) \cup \{Q\}$
 - 7: **return** \mathcal{R}
-

The second component of our method is to choose from S , the set of qualified f -CCs, r most diversified ones that can cover the largest number of vertices. Because it is extremely expensive to enumerate all r -combinations of f -CCs in S and then select the most diversified one, we design a method that approximately identifies the best r -combination of f -CCs. Our algorithm maintains a set \mathcal{R} of temporary diversified top- r f -CCs. Initially, $\mathcal{R} = \emptyset$. When a new f -CC Q is found, we use Procedure Update (Algorithm 1) to update \mathcal{R} with Q according to the following rules:

- R₁:** If $|\mathcal{R}| < r$, we add Q to \mathcal{R} (lines 1–2).
- R₂:** If $|\mathcal{R}| = r$, we replace an f -CC in \mathcal{R} by Q if such replacement can significantly expand $\text{cov}(\mathcal{R})$ (lines 3–6). Specifically, let $Q^* \in \mathcal{R}$ be the f -CC that covers the least number of vertices exclusively by itself among all f -CCs in \mathcal{R} . Replace Q^* by Q if \mathcal{R} can cover at least $|\text{cov}(\mathcal{R})|/r$ more vertices after applying the replacement.

Theorem 4 Let $S = \{Q_1, Q_2, \dots, Q_n\}$ be a set of f -CCs and $\mathcal{R}^* \subseteq S$ be the set of r f -CCs in S that covers the largest number of vertices. Let \mathcal{R} be the final set of r f -CCs updated with Q_1, Q_2, \dots, Q_n in an arbitrary order according to the rules **R₁** and **R₂**. We have $|\text{cov}(\mathcal{R})| \geq |\text{cov}(\mathcal{R}^*)|/4$.

Proof For the proof of Theorem 4, please refer to [2]. □

By implementing the PNP-Index introduced in [21], the space and time complexity of Update is $O(r|V|)$ and $O(|V|)$, respectively [21].

3.3 Description of algorithm

Algorithm 2 FlexCS.

Input: A graph G , a monotonically increasing function f and an integer r

Output: The set \mathcal{R} of r f -CCs that approximately attains the highest diversity

```

1:  $\mathcal{R} \leftarrow \emptyset, S \leftarrow \emptyset$ 
2:  $g(x) \leftarrow \lceil f(x) \rceil$ 
3:  $\{K_1, K_2, \dots, K_{\kappa(G)}\} \leftarrow \text{CORE}(G)$  // core decomposition
   /* Lines 4–12 perform the core-based filtering */
4:  $k \leftarrow 1$ 
5: while  $k \leq \kappa(G)$  and  $|K_k| > 0$  do
6:    $\{C_1, C_2, \dots, C_{m'}\} \leftarrow \text{CC}(G[K_k])$  // connected components
7:   for  $i \leftarrow 1, \dots, m'$  do
8:     if  $|C_i| \leq g^{-1}(k)$  then
9:        $S \leftarrow S \cup \{C_i\}$ 
10:    for  $k' \leftarrow k, \dots, \kappa(G)$  do
11:       $K_{k'} \leftarrow K_{k'} - C_i$  // trimming  $k'$ -core
12:     $k \leftarrow k + 1$ 
   /* Lines 13–18 generate qualified  $f$ -CCs */
13:  $k_{\max} \leftarrow k$ 
14: for  $k \leftarrow k_{\max}, \dots, 1$  do
15:    $\{C_1, C_2, \dots, C_{m'}\} \leftarrow \text{CC}(G[K_k])$  // connected components
16:   for  $i \leftarrow 1, \dots, m'$  do
17:      $T \leftarrow \text{ConsCS}(G[C_i], g, k)$ 
18:      $S \leftarrow S \cup T$ 
19: for all  $Q \in S$  (in an arbitrary order) do
20:    $\mathcal{R} \leftarrow \text{Update}(\mathcal{R}, Q)$ 
21: return  $\mathcal{R}$ 

```

Our algorithm for solving the FlexCS problem is presented in Algorithm 2. It first initializes the result set \mathcal{R} and the set S of qualified f -CCs to be empty (line 1). For ease of presentation, let $g(x) = \lceil f(x) \rceil$. Then, it computes all nonempty maximal k -cores $K_1, K_2, \dots, K_{\kappa(G)}$ in G by the fast core decomposition algorithm CORE [3] (line 3). Next, the algorithm carries out the core-based filtering (lines 4–12). The filtering process stops when all vertices in the maximal k -core K_k have been removed. After applying the core-based filtering, the algorithm finds qualified f -CCs on the updated maximal k -cores (lines 13–18) in descending order of k . It first recomputes the connected components in K_k (line 15). Then, it invokes Procedure ConsCS to solve the ConsCS problem instance on each connected component of K_k (line 17) and adds the discovered f -CCs to S (line 18). The details of Procedure ConsCS will be presented in Section 4. After S has been obtained, the algorithm uses each f -CC in S to update \mathcal{R} (lines 19–20). Finally, \mathcal{R} is outputted (line 21).

The space and time complexity of Algorithm 2 are $O(r|V|)$ and $O(|V|^3)$, respectively. Since the analysis of the space and time complexity of Algorithm 2 requires the procedures and the results presented in Sections 4 and 5, we give the analysis in Appendix A.

4 Solving the ConsCS problem

In this section, we present our algorithm for solving the ConsCS problem. We only study the problem instances $\langle G[C], f, k \rangle$ for $|C| > g^{-1}(k)$ because the other instances for $|C| \leq g^{-1}(k)$ have been eliminated by the core-based filtering. In the following, we present how to generate an f -CC in Section 4.1 and describe the algorithm for solving the ConsCS problem in Section 4.2.

4.1 f -CC generation

We first introduce how to generate an f -CC in $G[C]$ satisfying the constraint \mathbf{C}^k enforced in Section 3.1. To avoid the huge search space of the enumeration paradigm and the large overlap between the enumerated f -CCs, we design a heuristic method inspired by [15] that leverages both greediness and randomness: It first generates a series of vertex subsets by progressively adding vertices starting from a seed vertex and then outputs the most diversified one.

Algorithm 3 FccGen.

Input: The induced subgraph $G[C]$ of a connected component C of K_k , a constraint function $g(x) = \lceil f(x) \rceil$, a minimum degree threshold k , and a seed vertex s

Output: An f -CC in $G[C]$ including s with minimum degree at least k or an empty set

```

1:  $Q \leftarrow \{s\}$ ,  $Q_{max} \leftarrow \emptyset$ 
2:  $Q_k \leftarrow \{v \in Q \mid d_{G[Q]}(v) < k\}$ 
3: while  $|Q| < g^{-1}(k)$  do
4:   if  $Q_k \neq \emptyset$  then
5:      $d_{min} \leftarrow \min_{v \in C-Q} d_{Q_k}(v)$ 
6:      $d_{max} \leftarrow \max_{v \in C-Q} d_{Q_k}(v)$ 
7:      $L \leftarrow \{v \in C - Q \mid d_{Q_k}(v) \geq (1 - \alpha)d_{min} + \alpha d_{max}\}$ 
8:     Select  $v$  at random from  $L$ 
9:   else
10:     $d_{min} \leftarrow \min_{v \in C-Q} d_Q(v)$ 
11:     $d_{max} \leftarrow \max_{v \in C-Q} d_Q(v)$ 
12:     $L \leftarrow \{v \in C - Q \mid d_Q(v) \geq (1 - \alpha)d_{min} + \alpha d_{max}\}$ 
13:    Select  $v$  at random from  $L$ 
14:     $Q \leftarrow Q \cup \{v\}$ 
15:     $Q_k \leftarrow \{v \in Q \mid d_{G[Q]}(v) < k\}$ 
16:    if  $Q_k = \emptyset$  then
17:       $Q_{max} \leftarrow Q$ 
18: return  $Q_{max}$ 

```

Algorithm 3 shows the proposed algorithm FccGen to generate an f -CC starting from a seed vertex s . The selection of seeds will be discussed in Section 4.2. At first, the current

selected vertex subset Q and the maximum f -CC discovered so far Q_{max} are initialized in line 1. To make all vertices in $G[Q]$ have their degrees at least k as early as possible, we design the following strategy to add a new vertex to Q . Let $Q_k \subseteq Q$ be the set of vertices of degree less than k in $G[Q]$ (line 2). For a vertex $v \in C - Q$, let $d_{Q_k}(v)$ be the number of vertices in Q_k that are adjacent to v in $G[C]$. The larger $d_{Q_k}(v)$ is, the more vertices in Q_k can have their degrees increased after adding v to Q . However, selecting v with the largest $d_{Q_k}(v)$ may lead the search to a sub-optimal cohesive subgraph. Hence, we select v with $d_{Q_k}(v) \geq (1 - \alpha) \min_{u \in C - Q} d_{Q_k}(u) + \alpha \max_{u \in C - Q} d_{Q_k}(u)$ uniformly at random, where $\alpha \in [0, 1]$ (lines 5–8). Parameter α controls the trade-off between greediness and randomness: when $\alpha = 1$, the vertex v with the largest $d_{Q_k}(v)$ is selected; when $\alpha = 0$, an arbitrary vertex adjacent to a vertex in Q_k is chosen randomly.

When the degree of every vertex in $G[Q]$ becomes at least k , the generation step continues adding new vertices to Q to form an even larger f -CC. We now have $Q_k = \emptyset$, so the heuristic strategy described above is not applicable any more. Alternatively, we select a vertex v with $d_Q(v) \geq (1 - \alpha) \min_{u \in C - Q} d_Q(u) + \alpha \max_{u \in C - Q} d_Q(u)$ uniformly at random, where $\alpha \in [0, 1]$ (lines 10–13), since v is expected to be cohesively linked to Q .

As long as a new vertex v has been selected, we add it to Q (line 14) and recalculate Q_k (line 15). When Q_k is empty, Q is the current largest f -CC satisfying the constraint \mathbf{C}^k . We hence copy Q to Q_{max} (lines 16–17). According to the maximum size constraint \mathbf{C}_2^k , the generation process stops when $|Q| = g^{-1}(k)$ (line 3). Q_{max} must be a subset of C and satisfy the constraint \mathbf{C}^k , so it is returned as the result (line 18).

We now analyze the space and time complexity of FccGen. Apparently, the procedure only requires $O(|C|)$ space to keep Q , Q_{max} and candidate vertices in C to be added to Q . Therefore, the space complexity of FccGen is $O(|C|)$. Then, we consider its time complexity. In each iteration, the procedure selects a vertex from $C - Q$ to add to Q , which runs in $O(|C|)$ time. Since the generation process consists of at most $|C|$ iterations, it therefore takes $O(|C|^2)$ time. Overall, the time complexity of FccGen is $O(|C|^2)$.

4.2 Algorithm for the ConsCS problem

As presented in Section 4.1, the f -CC generation procedure FccGen starts from a seed vertex and generates an f -CC in the proximity of the seed. Since FccGen is fast and randomized, we can repeat it many times starting from different seeds to get more diversified f -CCs.

To approximately solve a ConsCS problem instance $\langle G[C], f, k \rangle$, a trivial method is to designate each vertex in C as a seed and generate an f -CC starting from the seed. However, this method often carries out a lot of redundant computation and produces many overlapping f -CCs. For efficiency consideration, we adopt the following seed selection strategies to reduce redundancy.

- The vertices in a generated f -CC will not be chosen as seeds any more.
- The vertices in the maximal $(k+1)$ -core K_{k+1} of G will not be selected as seeds because these vertices have been considered as seeds when solving the problem instances $\langle G[C'], f, k' \rangle$ for $k' \geq k + 1$ and $C' \subseteq C$.

Algorithm 4 ConsCS.

Input: The induced subgraph $G[C]$ of a connected component C of K_k , a constraint function $g(x) = \lceil f(x) \rceil$ and a minimum degree threshold k

Output: The set of discovered f -CCs in $G[C]$ with minimum degree at least k

```

1:  $T \leftarrow \emptyset$ 
2:  $S \leftarrow C - K_{k+1}$  // initial seed set
3: for  $s \in S$  do
4:    $Q \leftarrow \text{FccGen}(G[C], g, k, s)$ 
5:    $T \leftarrow T \cup \{Q\}$ 
6:    $S \leftarrow S - Q$ 
7: return  $T$ 

```

Based on the f -CC generation procedure FccGen presented in Algorithm 3 and the seed selection strategies described above, we develop an algorithm for approximately solving the ConsCS problem in Algorithm 4. The procedure is self-explanatory. We now analyze the space and time complexity of ConsCS. As analyzed in Section 4.1, the FccGen procedure invoked in line 4 requires $O(|C|)$ space and $O(|C|^2)$ time, so the space complexity of ConsCS is $O(|C|)$. Since the for loop in lines 3–6 is executed at most $|C|$ times, and each loop runs in $O(|C|^2)$ time, the time complexity of ConsCS is $O(|C|^3)$.

Example 5 Consider the example graph shown in Figure 2 and a ConsCS problem instance $\langle G[K_3], f(x) = \sqrt{x}, k = 3 \rangle$ on the connected 3-core K_3 . We consider $S = \{n, o, p, q, r, s, t, u\}$ as initial seeds since $\{c, d, e, f, g, h, i, j, k, l, m\} \subseteq K_4$. First, we select the vertex n as seed and execute FccGen. Assume that FccGen returns $Q_1 = \{d, e, g, h, j, k, l, n\}$, we add Q_1 into T and remove n from S . Then, we select the vertex o as seed and execute FccGen. Assume that FccGen returns $Q_2 = \{n, o, p, q, r, s, t, u\}$, we add Q_2 into T and remove vertices in Q_2 from S . Now, $S = \emptyset$, the procedure terminates and returns T .

5 Optimization techniques

This section introduces some optimization techniques that help speed up the proposed FlexCS algorithm.

5.1 On-the-fly result set updating

For ease of presentation, in FlexCS (Algorithm 2), we keep the qualified f -CCs in the set S in memory (lines 9 and 18) and later update the result set \mathcal{R} with the f -CCs in S at the end of the algorithm (lines 19–20). In fact, it is not necessary to keep S in memory. In our implementation, whenever a new f -CC Q is found in line 9 of FlexCS (Algorithm 2) or in line 4 of ConsCS (Algorithm 4), we immediately update \mathcal{R} with Q using Procedure Update (Algorithm 1). The on-the-fly updating of \mathcal{R} is correct because the order in which the f -CCs in S are used to update \mathcal{R} does not affect the approximation ratio of \mathcal{R} (see Theorem 4). The on-the-fly updating of \mathcal{R} can significantly reduce memory overhead because $|S|$ is usually very large for a large input graph G .

5.2 Early termination mechanism

With the on-the-fly result set updating, the result set \mathcal{R} can be filled with r f -CCs at an early stage of the FlexCS algorithm (Algorithm 2). It enables us to design a mechanism to early terminate the execution of ConsCS (Algorithm 4) and FlexCS (Algorithm 2) if they are not possible to generate any useful f -CCs to improve the coverage $\text{cov}(\mathcal{R})$ of \mathcal{R} . For ease of notation, let $\text{minCov}(\mathcal{R})$ denote the least number of vertices exclusively contained in an individual f -CC in \mathcal{R} , that is, $\text{minCov}(\mathcal{R}) = \min_{Q \in \mathcal{R}} |\text{cov}(\mathcal{R}) - \text{cov}(\mathcal{R} - \{Q\})|$. We have the following theorem:

Theorem 5 *Given an instance $\langle G[C], f, k \rangle$ of the ConsCS problem, none of the f -CCs generated by Procedure ConsCS (Algorithm 4) on $G[C]$ can update the result set \mathcal{R} if the following conditions hold:*

$$\begin{cases} |\mathcal{R}| = r, \\ \text{minCov}(\mathcal{R}) + \frac{1}{r}|\text{cov}(\mathcal{R})| > g^{-1}(k). \end{cases} \tag{3}$$

Proof Let Q^* be the f -CC in \mathcal{R} that covers the least number of vertices exclusively by itself, i.e., $|\text{cov}(\mathcal{R}) - \text{cov}(\mathcal{R} - \{Q^*\})| = \text{minCov}(\mathcal{R})$. If the conditions presented in (3) hold, for each f -CC Q in the result set of the ConsCS problem instance $\langle G[C], f, k \rangle$, replacing Q^* with Q leads to a new result set \mathcal{R}' such that:

$$\begin{aligned} |\text{cov}(\mathcal{R}')| &= |\text{cov}(\mathcal{R} - \{Q^*\}) \cup \{Q\}| \\ &\leq |\text{cov}(\mathcal{R})| - \text{minCov}(\mathcal{R}) + g^{-1}(k) \text{ // According to } \mathbf{C}_2^k \\ &< \left(1 + \frac{1}{r}\right) |\text{cov}(\mathcal{R})|. \end{aligned}$$

According to the rule \mathbf{R}_2 presented in Section 3.2, Q cannot be used to update \mathcal{R} . The theorem thus holds. □

Theorem 5 naturally indicates the following early termination mechanism for ConsCS (Algorithm 4). In Algorithm 4, we check the conditions presented in (3) before executing FccGen in line 4 and terminate Algorithm 4 if the conditions hold because no more useful f -CCs can be generated to update \mathcal{R} .

We can further extend this early termination mechanism to FlexCS (Algorithm 2). In Algorithm 2, we check the conditions presented in (3) before executing line 15 and terminate Algorithm 2 if the conditions hold. If Algorithm 2 is early terminated for some value of k , it completely avoids executing ConsCS on the connected components of the maximal k' -cores for $k' \leq k$ because no more useful f -CCs can be generated on any connected component in these maximal k' -cores to update \mathcal{R} . This also explains why k is considered from k_{\max} downward to 1 in line 14 of Algorithm 2 (just for early termination).

5.3 Index-based connected component retrieval

Note that FlexCS (Algorithm 2) computes the connected components of each maximal k -core K_k in G twice (line 6 and line 15). In line 6, it computes the connected components of K_k to realize the core-based filtering (see Section 3.1.2) that removes small connected components of K_k (in ascending order of k). Since K_k is changed in line 11, we must re-compute the connected components of K_k in line 15 and execute ConsCS to generate

qualified f -CCs on each large connected component of K_k (in descending order of k). Obviously, large amount of redundant computation has to be carried out in line 15.

To reduce such redundant computation, one available alternative is to compute all connected components of K_k in line 6 and save them in memory for future use. However, this approach requires $O(\kappa(G)|V|)$ space to keep all large-sized connected components, where $\kappa(G)$ is the core number of G (Definition 3). To make a trade-off between time and space, we design the *connected component index* (CCIndex for short) to facilitate the retrieval of the connected components of K_k . The CCIndex only requires $O(|V|)$ space, and with the help of it, we can avoid the redundant computation caused by re-computing connected components performed in line 15 of Algorithm 2.

5.3.1 Connected component index (CCIndex)

A CCIndex is composed of two arrays A and CC :

1. The array A holds all vertices in G in a specific order which will be introduced below.
2. The array CC serves as an access path to the information stored in A .

The key to designing the CCIndex is the ordering of vertices in the array A . The central idea is that all vertices in a connected component C of the maximal k -core K_k are consecutively placed in A so that C can be efficiently retrieved by doing a sequential read on A . To retrieve C , we only need to know the offset s_C of the first vertex in C within A and the size $|C|$ of C because $C = \{A[i] | s_C \leq i < s_C + |C|\}$. Such ordering of vertices in A can be realized based on the hierarchical structure described in the following lemma.

Lemma 3 *Each connected component of the maximal k -core K_k is a subset of a connected component of the maximal k' -core $K_{k'}$ if $k' \leq k$.*

Proof Let C be a connected component of K_k and C' be a connected component of $K_{k'}$ that overlaps with C . Suppose $C \not\subseteq C'$. We have that $C \cup C'$ is a k' -core and $C \cup C'$ is connected. It contradicts with the fact that C' is a connected component of $K_{k'}$. Thus, the lemma holds. \square

Figure 5 illustrates how we order the vertices in the array A according to the hierarchical structure described in Lemma 3. By leveraging the hierarchical structure among connected components, the overlaps between connected components are stored in A only once. Therefore, the array A can be regarded as a redundancy-free compression of all connected components of all maximal k -cores of G . The details of constructing the array A will be presented in the next subsection.

After the order of vertices in A has been determined, the information required for retrieving each connected component is kept in the array CC . As mentioned above, for each connected component C of the maximal k -core K_k , it can be identified by the offset s_C of the first vertex in C within A and the size $|C|$ of C . The pair $(s_C, |C|)$ is thus recorded in the element $CC[k]$. Since K_k may be composed of several connected components, $CC[k]$ actually records a list of pairs $(s_C, |C|)$ for all connected components C in K_k .

Example 6 Figure 6 shows a CCIndex for the example graph given in Figure 2. The vertices in the array A are ordered such that all vertices in a connected component of the maximal k -core K_k are consecutively placed in A . For $k = 4$, the maximal 4-core in the graph is

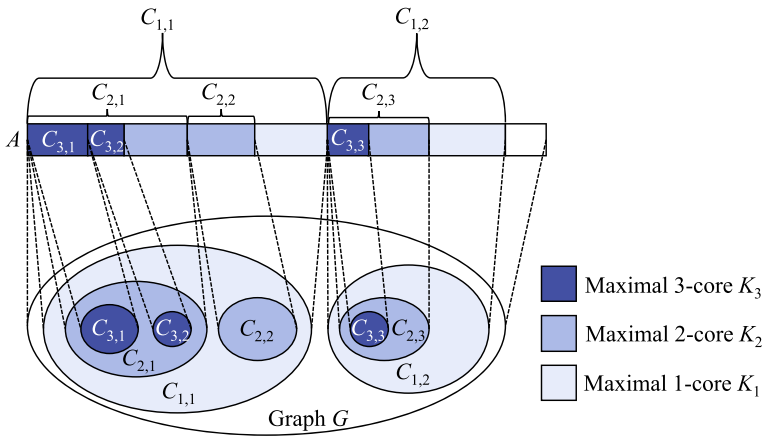


Figure 5 Illustration of the ordering of vertices in the array A in the CCIndex. The maximal 1-core K_1 of graph G has two connected components $C_{1,1}$ and $C_{1,2}$; The maximal 2-core K_2 has three connected components $C_{2,1}$, $C_{2,2}$ and $C_{2,3}$; The maximal 3-core K_3 has three connected components $C_{3,1}$, $C_{3,2}$ and $C_{3,3}$. The hierarchical structure of these connected components is illustrated by the nested circles. The dotted lines depict the mapping from a connected component to a sub-array of A

$K_4 = \{c, d, e, f, g, h, i, j, k, l, m\}$. There is only one connected component in K_4 , i.e., K_4 itself. The size of K_4 is 11, and the first vertex (m) in K_4 within A is placed in $A[0]$. Therefore, the pair $(0, 11)$ is recorded in the element $CC[4]$. According to the pair $(0, 11)$, this connected component can be retrieved as $A[0 \dots 10]$.

Furthermore, to efficiently support the core-based filtering (Section 3.1.2) and the early termination mechanism (Section 5.2), the pairs $(s_C, |C|)$ in $CC[k]$ are sorted in ascending order of $|C|$. The core-based filtering can be implemented by examining each pair $(s_C, |C|)$ in $CC[k]$ and removing it from $CC[k]$ if $|C| \leq g^{-1}(k)$. Procedure ConsCS is then executed on each connected component C represented by each remaining pair $(s_C, |C|)$ in $CC[k]$.

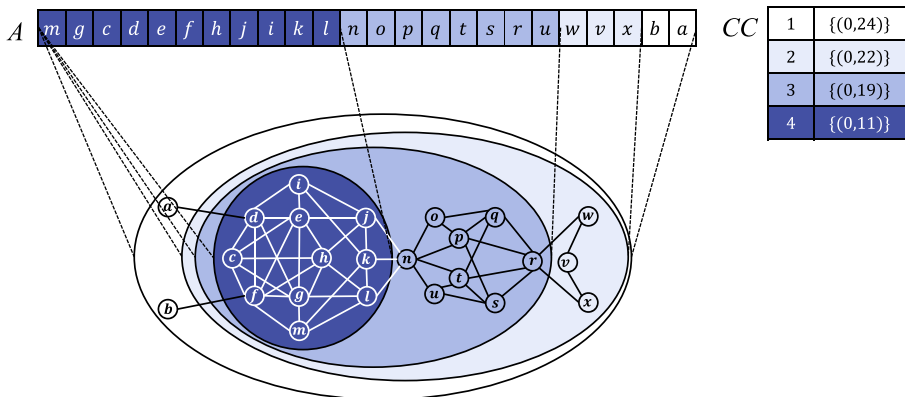


Figure 6 A CCIndex for the example graph given in Figure 2

5.3.2 Construction of CCIndex

Algorithm 5 CCIndex.

Input: A graph $G = (V, E)$, the core number $\kappa(G)$ of G and the core number $\kappa(v)$ of every vertex $v \in V$

Output: A CCIndex of G

- 1: Initialize arrays A and CC
 /* Lines 2–5 add all vertices in V to A */
- 2: $i \leftarrow 1$
- 3: **for all** $v \in V$ **do**
- 4: $A[i] \leftarrow v$
- 5: $i \leftarrow i + 1$
- 6: Reorder($A, CC, 0$)
- 7: **return** A and CC

Procedure Reorder($A[s : t], CC, k$)

- 8: $CC[k] \leftarrow CC[k] \cup \{(s, t - s + 1)\}$
 /* Lines 9–16 move all vertices $v \in A[s : t]$ with $\kappa(v) > k$ to the front of $A[s : t]$ */
- 9: $i \leftarrow s, j \leftarrow t, l \leftarrow 0$
- 10: **while** $i \leq j$ **do**
- 11: **if** $\kappa(A[i]) = k$ **then**
- 12: Swap($A[i], A[j]$)
- 13: $j \leftarrow j - 1$
- 14: **else**
- 15: $i \leftarrow i + 1$
- 16: $l \leftarrow l + 1$
- 17: **if** $l = 0$ **then**
- 18: **return**
- 19: $\{C_1, C_2, \dots, C_m\} \leftarrow CC(G[\{A[i] | s \leq i \leq s + l - 1\}])$
 /* Lines 20–24 store each connected component C_i consecutively in $A[s : s + l - 1]$ */
- 20: $j \leftarrow s$
- 21: **for** $i \leftarrow 1$ **to** m **do**
- 22: **for all** $v \in C_i$ **do**
- 23: $A[j] \leftarrow v$
- 24: $j \leftarrow j + 1$
 /* Lines 25–29 recursively reorder the vertices in $A[s : s + l - 1]$ */
- 25: $s' \leftarrow s$
- 26: **for** $i \leftarrow 1$ **to** m **do**
- 27: $t' \leftarrow s' + |C_i| - 1$
- 28: Reorder($A[s' : t'], CC, k + 1$)
- 29: $s' \leftarrow t' + 1$

The algorithm for constructing a CCIndex is presented in Algorithm 5. The input of the algorithm includes the graph $G = (V, E)$, the core number $\kappa(G)$ of G and the core number $\kappa(v)$ of every vertex $v \in V$. The output of the algorithm is a CCIndex of G . The algorithm works as follows. First, we set up the array A and the array CC in the CCIndex (line 1). The length of A is $|V|$, and the length of CC is $\kappa(G)$. For ease of notion, let $A[s : t]$ denote the sub-array of A starting from $A[s]$ and ending at $A[t]$, where $1 \leq s \leq t \leq |V|$. Then, we

add all vertices in V to A in an arbitrary order (lines 2–5). The key step of the algorithm is reordering the vertices in A (line 6) so that they satisfy the criterion of ordering described in Section 5.3.1.

The reordering of A is realized by the recursive procedure **Reorder**. The procedure takes as input a sub-array $A[s : t]$, the array CC and an integer $k \in [0, \kappa(G)]$. As will be shown later, this procedure has the following invariant: *all vertices in $A[s : t]$ form a connected component of the maximal k -core K_k of G for $k = 1, 2, \dots, \kappa(G)$* . Therefore, we have $\kappa(v) \geq k$ for all $v \in A[s : t]$. The procedure works as follows. First, the vertices in $A[s : t]$ form a connected component of K_k , and its size is $t - s + 1$. To enable fast access to this connected component, we add the pair $(s, t - s + 1)$ to the element $CC[k]$ of the array CC (line 8). Then, we move all vertices $v \in A[s : t]$ with $\kappa(v) > k$ to the front of $A[s : t]$ (lines 9–16). Suppose there are l vertices $v \in A[s : t]$ with $\kappa(v) > k$.

If $l = 0$, all vertices in $A[s : t]$ have core number k . According to the invariant of the procedure, all vertices in $A[s : t]$ form a connected component of K_k . This connected component is now consecutively placed in $A[s : t]$, which satisfies the criterion of the ordering described in Section 5.3.1. Since all vertices in $A[s : t]$ have core number k , none of the connected components of $K_{k'}$ with $k' > k$ is nested in $A[s : t]$. Therefore, the reordering of $A[s : t]$ is done, and the procedure returns (line 18).

If $l > 0$, some of the connected components of $K_{k'}$ with $k' > k$ are nested in $A[s : s + l - 1]$, so we must recursively reorder the vertices in $A[s : s + l - 1]$. First, we compute the connected components of the subgraph of G induced by the vertices in $A[s : s + l - 1]$ (line 19). Let C_1, C_2, \dots, C_m be these connected components. Then, we reorder the vertices in $A[s : s + l - 1]$ so that the vertices in each connected component C_i are consecutively placed in A . Particularly, we move the vertices in C_i to the sub-array $A[s_i : t_i]$, where $s_i = s + \sum_{j=1}^{i-1} |C_j|$ and $t_i = s_i + |C_i| - 1$ (lines 20–24). The order of vertices in $A[s_i : t_i]$ is arbitrary. Next, for $i = 1, 2, \dots, m$, we recursively reorder the vertices in $A[s_i : t_i]$ by calling Procedure **Reorder** with input $A[s_i : t_i]$, CC and $k + 1$ (lines 25–29).

In line 6 of the **CCIndex** construction algorithm (Algorithm 5), we call Procedure **Reorder** with input A , CC and $k = 0$ to start reordering all vertices in A . When the procedure returns, the construction of the **CCIndex** is completed. The algorithm outputs the arrays A and CC (line 7).

The correctness of Algorithm 5 is ensured by the invariant of Procedure **Reorder** stated in the following theorem.

Theorem 6 *All vertices in $A[s : t]$ form a connected component of the maximal k -core K_k of G for $k = 1, 2, \dots, \kappa(G)$.*

Proof We prove this theorem by induction on k . First, we consider the inductive base for $k = 1$. When Procedure **Reorder** is called at line 6 of Algorithm 5, it moves all vertices v in A with $\kappa(v) \geq 1$ to $A[1 : l]$ (lines 9–16) and computes the connected components C_1, C_2, \dots, C_m of the subgraph of G induced by the vertices in $A[1 : l]$ (line 19). Certainly, $K_1 = C_1 \cup C_2 \cup \dots \cup C_m$. Therefore, C_i is a connected component of K_1 for $1 \leq i \leq m$. Each connected component C_i is consecutively stored in $A[s_i : t_i]$ (lines 20–24), and $A[s_i : t_i]$ is recursively reordered by calling Procedure **Reorder** with input $A[s_i : t_i]$ and $k = 1$ (line 28). Therefore, all vertices in $A[s_i : t_i]$ form a connected component of K_1 . The base case holds.

Assume that when Procedure **Reorder** is called with input $A[s : t]$ and k , all vertices in $A[s : t]$ form a connected component of K_k . Certainly, we have $\kappa(v) \geq k$ for all vertices v in

$A[s : t]$. Procedure **Reorder** moves all vertices v in $A[s : t]$ with $\kappa(v) > k$ to $A[s : s + l - 1]$ (lines 9–16) and computes the connected components C_1, C_2, \dots, C_m of the subgraph of G induced by the vertices in $A[s : s + l - 1]$ (line 19). We have $\kappa(v) \geq k + 1$ for all $v \in C_i$. Therefore, C_i is a connected component of K_{k+1} . Procedure **Reorder** consecutively stores C_i in $A[s_i : t_i]$ (lines 20–24) and reorders $A[s_i : t_i]$ by recursively calling Procedure **Reorder** with input $A[s_i : t_i]$ and $k + 1$ (line 28). Thus, all vertices in $A[s_i : t_i]$ form a connected component of K_{k+1} . The induction holds. \square

Given the graph $G = (V, E)$, Algorithm 5 outputs the arrays A and CC in the $CCIndex$ of G . The array A stores all vertices in V , so it takes $O(|V|)$ space. The array CC stores a pair of integers $(s_C, |C|)$ for each connected component C of each maximal k -core in G . As will be shown in the following theorem, there are totally at most $|V|$ such connected components. Thus, CC requires $O(|V|)$ space. Consequently, it takes $O(|V|)$ space to store the $CCIndex$.

Theorem 7 *Given the graph $G = (V, E)$, let n_k be the number of connected components of the maximal k -core K_k of G . We have $\sum_{i=0}^{\kappa(G)} n_k \leq |V|$.*

Proof According to the definition of k -core, a k -core must be composed of at least $k + 1$ vertices. Therefore, $K_{\kappa(G)}$ contains at least $(\kappa(G) + 1)n_{\kappa(G)}$ vertices. For $0 \leq k < \kappa(G)$, by Lemma 3, there are at least $\max\{0, (n_k - n_{k+1})\}$ connected components in K_k that do not contain any connected component in K_{k+1} . Each of these connected components is a k -core, which contains at least $k + 1$ vertices. Therefore, $|K_k - K_{k+1}| \geq (k + 1) \max\{0, (n_k - n_{k+1})\}$. Hence, we have

$$\begin{aligned}
 |V| &= |K_{\kappa(G)}| + \sum_{k=0}^{\kappa(G)-1} |K_k - K_{k+1}| \\
 &\geq (\kappa(G) + 1)n_{\kappa(G)} + \sum_{k=0}^{\kappa(G)-1} (k + 1) \max\{0, (n_k - n_{k+1})\} \\
 &\geq n_{\kappa(G)} + n_{\kappa(G)-1} + \dots + n_1 + n_0.
 \end{aligned}
 \tag{4}$$

The last inequality (4) is derived based on the arguments shown in Appendix B. Thus, the theorem holds. \square

We now analyze the space complexity of Algorithm 5. Given as input the sub-array $A[s : t]$, Procedure **Reorder** needs $O(l)$ space to compute and keep the connected components C_1, C_2, \dots, C_m of the subgraph of G induced by the vertices in $A[s : s + l - 1]$ (line 19). We have $l \leq t - s + 1 \leq |V|$. This space can be released as soon as C_1, C_2, \dots, C_m have been consecutively stored in $A[s : s + l - 1]$ (lines 20–24). In addition to the $O(|V|)$ space taken by the $CCIndex$, Algorithm 5 totally requires $O(|V|)$ space.

Next, we analyze the time complexity of Algorithm 5. It takes $O(|V|)$ time to add all vertices in V to the array A (lines 2–5). The execution time of line 6 is $O(\kappa(G)(|V| + |E|))$, which is analyzed as follows. For $0 \leq k \leq \kappa(G)$, let $C_1^k, C_2^k, \dots, C_{n_k}^k$ be all connected components of K_k . We have $\sum_{i=1}^{n_k} |C_i^k| = |K_k|$. Suppose C_i^k has been consecutively placed

in $A[s_i^k : t_i^k]$. Given $A[s_i^k : t_i^k]$ and k as input, Procedure Reorder takes $O(|C_i^k|)$ time to move all vertices v in $A[s_i^k : t_i^k]$ with $\kappa(v) > k$ to the front of $A[s_i^k : t_i^k]$ (lines 9–16). If Procedure Reorder does not return in line 18, it computes connected components of K_{k+1} nested in C_i (line 19) and places these connected components consecutively in $A[s_i^k : t_i^k]$ (lines 20–24). For all $C_1^k, C_2^k, \dots, C_{n_k}^k$, it totally takes $O(\sum_{i=1}^{n_k} |C_i^k| = |K_k|)$ time to execute lines 9–16, $O(\sum_{i=1}^{n_{k+1}} |C_i^{k+1}| = |K_{k+1}|)$ time to execute lines 20–24 and $O(\sum_{i=1}^{n_{k+1}} (|C_i^{k+1}| + |E[C_i^{k+1}]|)) = O(|K_{k+1}| + |E[K_{k+1}]|)$ time to execute line 19, where $E[S]$ represents the set of edges in G with both endpoints in S . By Theorem 7, Algorithm 5 adds at most $|V|$ integer pairs to the array CC (line 8), which totally takes $O(|V|)$ time. Therefore, line 6 runs in $O(|V| + \sum_{k=0}^{\kappa(G)} (|K_k| + |E[K_k]|)) = O(\kappa(G)(|V| + |E|))$ time. Consequently, the time complexity of Algorithm 5 is $O(\kappa(G)(|V| + |E|))$.

6 Experiments

In this section, we evaluate the proposed f -CC model and the FlexCS algorithm. First, we show the flexibility of the f -CC model compared with the clique model and the k -core model. Then, we evaluate the practical performance of the algorithm FlexCS with respect to its parameters. Next, we examine the effects of the optimization techniques proposed in Section 5. Finally, we verify the effectiveness of the algorithm FlexCS by comparing its results with a theoretical bound.

6.1 Experiment setting

Algorithms Our FlexCS algorithm is compared with the diversified top- r cliques mining algorithm EnumKOpt [21] that finds r cliques covering the largest number of vertices and the core decomposition algorithm CORE [3] that computes all nonempty maximal k -cores. EnumKOpt uses the Update procedure (Algorithm 1) to maintain a set \mathcal{R} of temporary diversified top- r cliques except that replacement only takes place if it makes \mathcal{R} cover at least $\eta|\text{cov}(\mathcal{R})|/r$ more vertices. When $\eta = 1$, it is exactly Procedure Update (Algorithm 1). To be consistent with [21], we set $\eta = 0.3$. In addition, since CORE returns all nonempty maximal k -cores, we only save top- r largest ones as results for ease of comparison. In the rest of this section, we use \mathcal{R}_f , \mathcal{R}_{cli} and \mathcal{R}_{core} to represent the outputs of FlexCS, EnumKOpt and CORE, respectively.

Parameters The FlexCS algorithm has 4 parameters.

- r : The number of f -CCs to be found.
- α : The parameter used to tune the trade-off between randomization and greediness in Procedure FccGen (see Section 4.1).
- t : The number of threads used to execute Procedure FccGen on each maximal k -core in parallel.
- β : To attain more flexibility, we normally expect that the constraint imposed by function f is stricter on small subgraphs but looser on large subgraphs. Therefore, we tested various constraint functions with such trends in the experiments, including exponential functions in forms of $f(x) = x^{1/\beta}$ and logarithmic functions in forms of $f(x) = \log_{\beta}(x)$, where $\beta \geq 1$. We obtain similar results for all these functions. Therefore, we only report the results for $f(x) = x^{1/\beta}$ in the rest of this section, where $\beta \geq 1$.

The default values of r , α , t and β are set to 60, 0.9, 16 and 2, respectively. To avoid uninformative small clusters, we ignore cohesive subgraphs consisting of less than 5 vertices.

Datasets Our experiments were performed on 9 real graphs obtained from the SNAP project (<http://snap.stanford.edu>). We regard the core number $\kappa(G)$ of a graph G as an important feature because our algorithm processes each maximal k -core in G in sequence. In addition, $\kappa(G)$ can reflect the density of G to some extent, which has essential impacts on most cohesive subgraph mining algorithms. Since we only consider simple graphs, we eliminated self-loops and multiple edges from all these graphs. Table 1 lists the statistics of these graphs. Note that the graph ca-GrQc is very small, which is only used for purpose of visualization in Section 6.2.

Experiment setup All the algorithms were implemented in C++ and compiled with the same optimization level. All experiments were carried out on a server with an Intel Xeon CPU (2.10GHz and 8 cores) and 220GB of RAM, running 64-bit Ubuntu 18.04. In our experiments, Procedure FccGen (Section 4.1) was implemented and executed using multi-threads.

6.2 Flexibility of the f -CC model

We first compare the flexibility of the f -CC model, the clique model and the k -core model. To evaluate the flexibility of these models, we first compare the sizes of cohesive subgraphs discovered by FlexCS, EnumKOpt and CORE. Then, we take a deeper look at the vertices covered by \mathcal{R}_f , \mathcal{R}_{cli} and \mathcal{R}_{core} .

6.2.1 On the comparison of cohesive subgraph size

We first examine the sizes of cohesive subgraphs discovered by FlexCS, EnumKOpt and CORE. We remove the maximal 0-core and the maximal 1-core from \mathcal{R}_{core} because they are nearly the entire graph. Figure 7 shows the sizes of f -CCs, cliques and k -cores in \mathcal{R}_f , \mathcal{R}_{cli} and \mathcal{R}_{core} in ascending order. We have the following observations: (1) The cliques found by EnumKOpt are always small, but the sizes of f -CCs discovered by FlexCS are distributed in a wide range. Both small f -CCs with tens of vertices and large f -CCs with hundreds of

Table 1 Statistics of graphs used in experiments

Graph	#Vertices	#Edges	$\kappa(G)$
Com-Amazon	334,863	925,872	6
Gemsec-Deezer (HU)	47,538	222,887	11
Ca-CondMat	23133	93,439	25
Gemsec-Facebook (New Sites)	27,917	205,964	31
Ca-GrQc	5,241	14,484	43
Email-Enron	36,692	183,831	43
Musae-facebook	22,470	170,823	56
Ca-AstroPh	18,771	198,050	56
Com-DBLP	317,080	1,049,866	113

vertices can be characterized simultaneously. (2) Compared with f -CCs, the k -cores found by CORE are much larger. Note that the sizes of cohesive subgraphs are shown in log scale in Figure 7b. Therefore, the k -core model is not suitable for characterizing small clusters. In conclusion, the f -CC model attains more flexibility than both the clique model and the k -core model.

6.2.2 On the comparison of cohesive subgraph coverage

In this section, we take a deeper look at the vertices covered by the cohesive subgraphs discovered under different models. Let $\text{cov}(\mathcal{R}_{cli})$ and $\text{cov}(\mathcal{R}_{core})$ be the coverage of \mathcal{R}_{cli} and \mathcal{R}_{core} , respectively. For ease of comparison, we introduce two measures: *precision* and *recall*. Precision is defined as $\frac{|\text{cov}(\mathcal{R}_f) \cap \text{cov}(\mathcal{R}_o)|}{|\text{cov}(\mathcal{R}_f)|}$, and recall is defined as $\frac{|\text{cov}(\mathcal{R}_f) \cap \text{cov}(\mathcal{R}_o)|}{|\text{cov}(\mathcal{R}_o)|}$, where $\mathcal{R}_o \in \{\mathcal{R}_{cli}, \mathcal{R}_{core}\}$.

\mathcal{R}_f v.s. \mathcal{R}_{cli} When comparing the vertices covered by f -CCs found by FlexCS and cliques discovered by EnumKOpt, we vary β from 1.5 to 2.5. Table 2 shows the results. We can see that the precision is smaller than 0.5 in most cases. Actually, \mathcal{R}_{cli} only contains a small proportion of vertices covered by \mathcal{R}_f due to the strong constraint imposed by the clique model. Nevertheless, the recall is always large for most graphs, i.e., the discovered f -CCs can cover a large number of vertices in cliques. Especially, on com-DBLP, \mathcal{R}_f covers more than 89 percent of vertices in \mathcal{R}_{cli} for all β . However, for com-Amazon and gemsec-Deezer(HU) whose core numbers are relatively small, they usually contain a large number of small f -CCs and cliques. \mathcal{R}_{cli} and \mathcal{R}_f may cover different parts of the graph, and therefore the recall is small. Moreover, we can find that as β increases, the precision decreases, but the recall increases. It is consistent with our expectation that larger β leads to larger f -CCs, which will further result in a more diversified \mathcal{R}_f that definitely contains more cliques.

\mathcal{R}_f v.s. \mathcal{R}_{core} When comparing vertices covered by f -CCs found by FlexCS and k -cores discovered by CORE, it is meaningless to compare \mathcal{R}_f with some k -core since f -CCs are extracted from all maximal k -cores. Hence, we calculate the recall of \mathcal{R}_f and each k -core for $k \leq \kappa(G)$ and then examine the variation of recall w.r.t. k to reveal the relationships between f -CCs and k -cores.

The experimental results on gemsec-Deezer (HU) and com-DBLP are shown in Figure 8. We can see that the recall increases with k , that is, \mathcal{R}_f can basically cover k -cores for large k . Specifically, \mathcal{R}_f covers all vertices in the 9-core on gemsec-Deezer (HU) and the 26-core on com-DBLP. However, \mathcal{R}_f covers fewer vertices on k -cores with small k . The observation

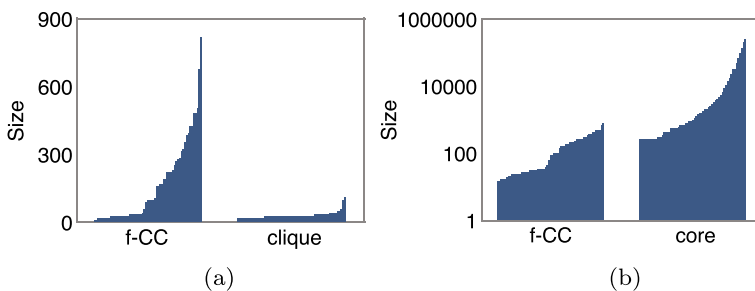


Figure 7 Sizes of cohesive subgraphs discovered by FlexCS, EnumKOpt and CORE on com-DBLP. (a) f -CCs v.s. cliques; (b) f -CCs v.s. cores

Table 2 Comparison between \mathcal{R}_f and \mathcal{R}_{cli}

Graph	$ \text{cov}(\mathcal{R}_{cli}) $	β	$ \text{cov}(\mathcal{R}_f) $	Precision	Recall
Com-Amazon	372	1.5	411	0.017	0.019
		2	1059	0.007	0.019
		2.5	2216	0.005	0.030
Gemsec-Deezer (HU)	402	1.5	486	0.272	0.328
		2	962	0.185	0.443
		2.5	1420	0.105	0.371
Ca-CondMat	787	1.5	1130	0.476	0.684
		2	2109	0.296	0.794
		2.5	2900	0.233	0.860
Gemsec-Facebook (New Sites)	816	1.5	1441	0.362	0.638
		2	2749	0.242	0.814
		2.5	4035	0.173	0.857
Email-Enron	551	1.5	946	0.325	0.557
		2	2005	0.204	0.744
		2.5	3437	0.139	0.866
Musae-facebook	941	1.5	2138	0.333	0.758
		2	3817	0.207	0.842
		2.5	4831	0.174	0.896
Ca-AstroPh	1888	1.5	2673	0.591	0.837
		2	3670	0.472	0.917
		2.5	4742	0.384	0.965
Com-DBLP	1949	1.5	2973	0.585	0.892
		2	4938	0.393	0.996
		2.5	6977	0.279	1.000

is consistent with our motivation: f -CCs can cover most vertices in cohesively connected regions but exclude those in sparse regions.

Visualization To better illustrate the differences between cohesive subgraphs characterized by these models, we visualize \mathcal{R}_f , \mathcal{R}_{cli} and \mathcal{R}_{core} obtained on the smallest dataset ca-GrQc in Figure 9. The vertices in $\text{cov}(\mathcal{R}_f) \cap \text{cov}(\mathcal{R}_o)$, $\text{cov}(\mathcal{R}_f) - \text{cov}(\mathcal{R}_o)$ and $\text{cov}(\mathcal{R}_o) - \text{cov}(\mathcal{R}_f)$ are colored in red, green and blue, respectively, where $\mathcal{R}_o = \mathcal{R}_{cli}$ or \mathcal{R}_{core} . Note that we eliminate all discovered cohesive subgraphs with size smaller than 6 for sake of fairness.

Figure 9a shows the vertices covered by \mathcal{R}_f and \mathcal{R}_{cli} . We can see that \mathcal{R}_f covers a large portion of vertices in \mathcal{R}_{cli} . Note that the green vertices in $\text{cov}(\mathcal{R}_f) - \text{cov}(\mathcal{R}_{cli})$ are also cohesively connected to each other and/or to the red vertices in $\text{cov}(\mathcal{R}_f) \cap \text{cov}(\mathcal{R}_{cli})$, but they are excluded by \mathcal{R}_{cli} . Figure 9b shows the vertices covered by \mathcal{R}_f and \mathcal{R}_{core} . The blue vertices in $\text{cov}(\mathcal{R}_{core}) - \text{cov}(\mathcal{R}_f)$ are always sparsely connected compared with the

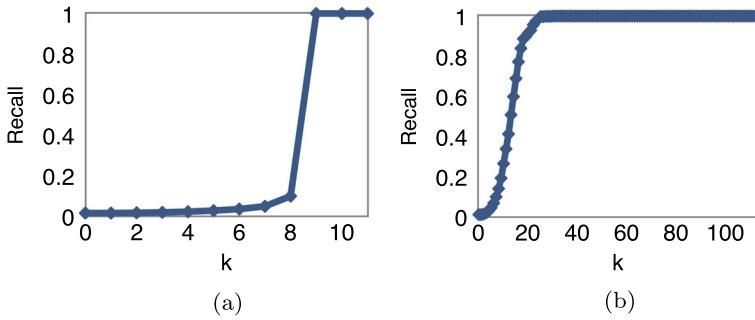


Figure 8 Recall of \mathcal{R}_f and k -core w.r.t. k on graphs gemsec-Deezer(HU) and com-DBLP. (a) Gemsec-Deezer(HU); (b) Com-DBLP

red vertices in $\text{cov}(\mathcal{R}_f) \cap \text{cov}(\mathcal{R}_{core})$ and the green vertices in $\text{cov}(\mathcal{R}_f) - \text{cov}(\mathcal{R}_{core})$, and fortunately they are excluded from the f -CCs discovered by FlexCS. The vertices in each f -CC in \mathcal{R}_f are guaranteed to be cohesively connected.

6.3 Performance of FlexCS

Next, we evaluate the performance of the proposed FlexCS algorithm. In the experiments, we use two evaluation metrics: *execution time* and *result coverage size (RCS)* $|\text{cov}(\mathcal{R}_f)|$. For ease of presentation, let \mathcal{R} denote the temporary result set at some moment of the execution of FlexCS. In the rest of this section, we only present some representative results for each experiment to save space. Note that, when evaluating the effects of a parameter, the other parameters are fixed to their default values.

Execution time and RCS w.r.t. r Parameter r is the number of f -CCs to be found. Figure 10 shows the execution time and the RCS of FlexCS w.r.t. r on gemsec-Facebook and com-DBLP. Both RCS and execution time increase with r . This is because larger r leads to more f -CCs in \mathcal{R} , and therefore \mathcal{R}_f certainly covers more vertices. While, maintaining \mathcal{R} becomes more costly as r increases, and the early termination mechanism proposed in Section 5.2 becomes more difficult to take effects.

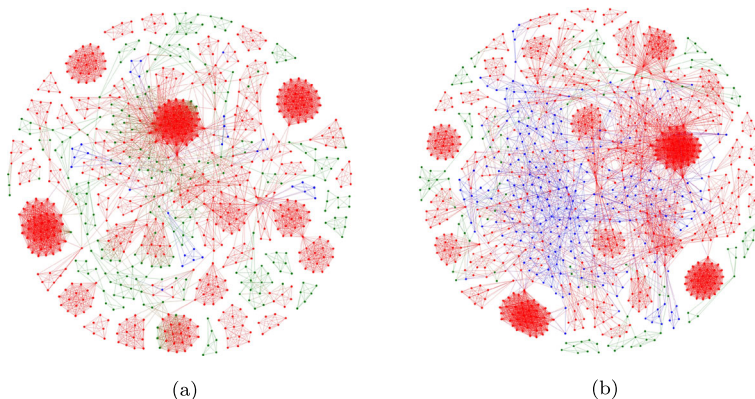


Figure 9 Visualization of \mathcal{R}_f , \mathcal{R}_{cli} and \mathcal{R}_{core} on ca-GrQc. (a) \mathcal{R}_f v.s. \mathcal{R}_{cli} ; (b) \mathcal{R}_f v.s. \mathcal{R}_{core}

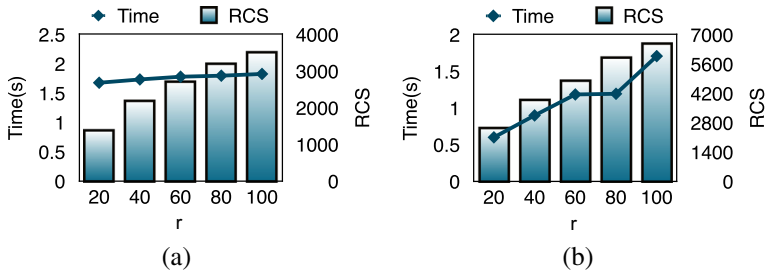


Figure 10 Execution time and RCS w.r.t. r on graphs gemsec-Facebook and com-DBLP. (a) Gemsec-Facebook; (b) Com-DBLP

Execution time and RCS w.r.t. α Parameter α tunes the trade-off between randomization and greediness in Procedure FccGen (see Section 4.1). Figure 11 shows the execution time and the RCS of FlexCS w.r.t. α on gemsec-Deezer (HU) and ca-AstroPh. We have the following observations: (1) In most cases, the RCS increases with α . When α becomes larger, our algorithm tends to generate an f -CC more greedily, so it is more possible to obtain f -CCs satisfying the minimum degree constraint C_1^k to fill \mathcal{R} . (2) The execution time decreases with α . Since larger α increases the chance to find valid f -CCs, \mathcal{R} is filled with f -CCs earlier. Therefore, the early termination mechanism is more likely to take effects early.

Execution time and RCS w.r.t. t Parameter t is the number of threads used to execute Procedure FccGen on each maximal k -core in parallel. Figure 12 shows the execution time and the RCS of FlexCS w.r.t. t on musae-facebook and ca-CondMat. We have the following observations: (1) The RCS fluctuates slightly with t . It is because FlexCS is a randomized algorithm. Indeed, t has no impact on the RCS; (2) The execution time decreases as t increases, which is consistent with our expectation that executing Procedure FccGen in parallel with multiple threads can decrease the execution time to a large extent. Note that, for graph Ca-CondMat, the execution time slightly increases when using 16 threads compared with the one using 8 threads. This is because \mathcal{R} can be updated by only one thread at a time. Too many threads increase the contention on updating \mathcal{R} .

Execution time and RCS w.r.t. β Recall that the form of constraint function used in the experiments is $f(x) = x^{1/\beta}$, where $\beta \geq 1$. Figure 13 shows the execution time and RCS w.r.t. β on com-Amazon and email-Enron. We have the following observations: (1) The RCS increases with β . Apparently, a larger β imposes a looser constraint on f -CCs, which leads to more diversified and larger f -CCs. (2) The execution time also increases with β . According

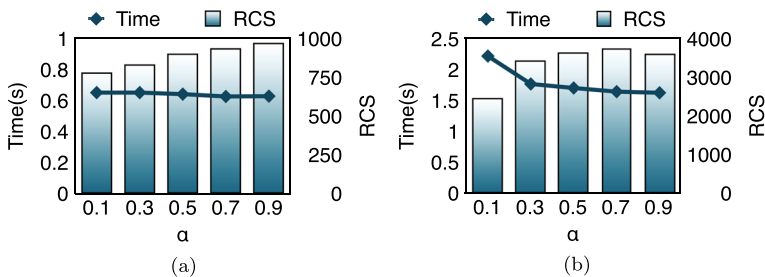


Figure 11 Execution time and RCS w.r.t. α on graphs gemsec-Deezer(HU) and ca-AstroPh. (a) Gemsec-Deezer(HU); (b) Ca-AstroPh

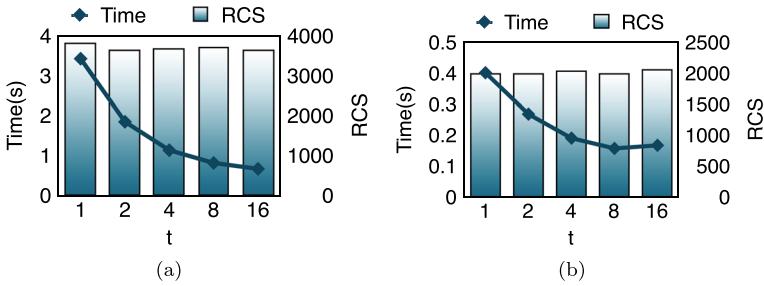


Figure 12 Execution time and RCS w.r.t. t on graphs musae-facebook and ca-CondMat. (a) Musae-facebook; (b) Ca-CondMat

to C_2^k , the maximum size of f -CCs w.r.t. each minimum vertex degree increases exponentially as β increases. The number of iterations in FccGen thus increases exponentially as well, and the execution time increases significantly.

6.4 Effects of optimization techniques

We also evaluate the effects of the optimization techniques proposed in Section 5, particularly, the early termination mechanism (Section 5.2) and the CCIndex-based connected component retrieval (Section 5.3).

6.4.1 Effects of early termination mechanism

In this experiment, we evaluate the effects of the early termination mechanism (Section 5.2) on the performance of FlexCS. Table 3 shows the experimental results. We have the following observations: (1) For graphs com-Amazon and com-DBLP, the early termination mechanism can significantly decrease the execution time. For graph com-Amazon with small core number $\kappa(G)$, there are lots of small f -CCs that are unable to update \mathcal{R} . Therefore, many ConsCS problem instances and many executions of FccGen can be eliminated. While, for graph com-DBLP whose core number $\kappa(G)$ is large, \mathcal{R} is filled with large f -CCs early, which helps filter out ConsCS instances on k -cores with small k . (2) For other graphs, the effects of the early termination mechanism are not so significant. This is because the number of vertices covered by \mathcal{R} is not very large, so there always exist f -CCs in the graph that can update \mathcal{R} .

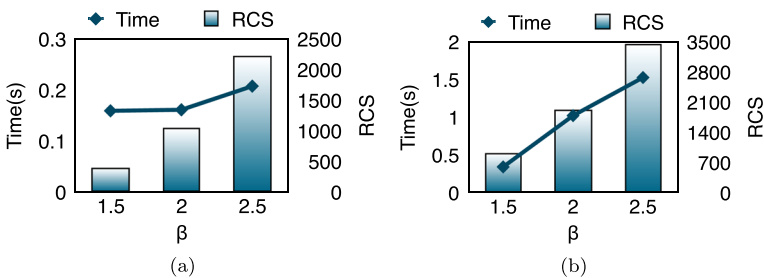


Figure 13 Execution time and RCS w.r.t. β on graphs com-Amazon and email-Enron. (a) Com-Amazon; (b) Email-Enron

Table 3 Execution time of FlexCS without/with early termination (abbr. E.T.)

Graph	Time (w/o E.T.) (s)	Time (w/ E.T.) (s)	Ratio
Com-Amazon	19.383	0.162	0.008
Gemsec-Deezer(HU)	0.671	0.629	0.937
Ca-CondMat	0.237	0.168	0.709
Gemsec-Facebook(New Sites)	1.872	1.782	0.952
Email-Enron	1.167	1.028	0.881
Musae-facebook	0.767	0.676	0.881
Ca-AstroPh	1.687	1.627	0.964
Com-DBLP	14.823	1.187	0.080

Ratio represents the ratio between the execution time with early termination and the execution time without early termination

6.4.2 Effects of CCIIndex

In this experiment, we evaluate the effects of the CCIIndex (Section 5.3) on the performance of FlexCS. The experimental results are shown in Tables 4 and 5. We can see from Table 4 that the CCIIndex brings a decrease in the execution time of FlexCS for every tested graph. The performance improvements achieved by the CCIIndex seem not significant because the computation of connected components only takes a small portion of execution time of FlexCS (about 50% for com-Amazon and up to 10% for other graphs). Let us take a deeper look at the impacts of the CCIIndex on the time of connected component computation and retrieval in FlexCS. From Table 5, we can see that the CCIIndex helps decrease the time for component computation and retrieval by 15.1–66.7 percent on the tested graphs. These experimental results verify the effectiveness of the CCIIndex.

6.5 Effectiveness of FlexCS

To evaluate the effectiveness of FlexCS, we compare RCS with a theoretical upper bound, which is derived in the following way: We compute the maximum number of disjoint f -CCs with size $g^{-1}(k)$ for $k = \kappa(G), \dots, 1$ and accumulate their sizes until the top- r f -CCs

Table 4 Execution time of FlexCS without/with the CCIIndex

Graph	Time (w/o CCIIndex) (s)	Time (w/ CCIIndex) (s)	Ratio
Com-Amazon	0.172	0.158	0.919
Gemsec-Deezer (HU)	0.665	0.644	0.968
Ca-CondMat	0.175	0.166	0.949
Gemsec-Facebook (New Sites)	1.823	1.793	0.984
Email-Enron	1.079	1.035	0.959
Musae-facebook	0.765	0.689	0.901
Ca-AstroPh	1.782	1.632	0.916
Com-DBLP	1.391	1.216	0.874

Ratio represents the ratio between the execution time with CCIIndex and the execution time without CCIIndex

Table 5 Time of connected component computation and retrieval (CC-time for short) without/with the CCIndex

Graph	CC-time (w/o CCIndex) (s)	CC-time (w/ CCIndex) (s)	Ratio
Com-Amazon	0.086	0.073	0.849
Gemsec-Deezer (HU)	0.028	0.011	0.393
Ca-CondMat	0.003	0.001	0.333
Gemsec-Facebook (New Sites)	0.019	0.010	0.526
Email-Enron	0.006	0.003	0.500
Musae-facebook	0.011	0.005	0.455
Ca-AstroPh	0.019	0.007	0.368
Com-DBLP	0.118	0.088	0.746

Ratio represents the ratio between the CC-time with CCIndex and the CC-time without CCIndex

have been found. The correctness of such paradigm is guaranteed by Lemma 1 and the increasing monotonicity of f . Table 6 shows the results. We can see that the RCS is larger than 25 percent of the upper bound on each tested graph. Note that, the upper bound is the theoretical maximum regardless of the topological structure and connectivity of the graph, which is basically unreachable in real-world graphs.

7 Related work

7.1 Cohesive subgraph models

Different cohesive subgraph models impose constraints on different aspects of cohesive subgraphs. The existing models can be mainly categorized into three classes. The *density-based models* [11] impose constraints on the density of a subgraph. The *distance-based models* [11] impose constraints on the distance between any pair of vertices in a subgraph. The *connectivity-based models* [5] impose constraints on the edge/vertex connectivity of a subgraph. None of these models is flexible in characterizing multi-scale cohesive subgraphs simultaneously. We explain this with a specific model in each class. First, as a typical

Table 6 Comparison between RCS and its upper bound

Graph	RCS	Upper Bound	Ratio
Com-Amazon	1064	1622	0.656
Gemsec-Deezer (HU)	971	3865	0.251
Ca-CondMat	2064	4962	0.416
Gemsec-Facebook (New Sites)	2707	9923	0.273
Email-Enron	1944	6983	0.278
Musae-facebook	3625	8657	0.419
Ca-AstroPh	3616	10082	0.359
Com-DBLP	4807	12464	0.386

Ratio represents the ratio of the RCS and its upper bound

density-based model, a density-based γ -quasi-clique [1] with n vertices must be internally linked by at least $\gamma \binom{n}{2}$ edges, where $\gamma \in [0, 1]$. It is so strict that the model only fits for characterizing small cohesive subgraphs in a large graph due to the sparsity of real graphs. Second, the k -club model [14] is a typical distance-based model, in which every pair of vertices is within distance k . The size of a k -club generally grows exponentially as k increases because of the small-world nature of real graphs [18]. Therefore, the k -club model is not possible to simultaneously characterize both small and large cohesive subgraphs in a graph. Third, the k -ECC model [5] is a typical connectivity-based model. It requires that the edge connectivity of a cohesive subgraph is at least k . This model focuses on topological connectivity and is often applied to the scenarios where the edge/vertex cut size is the key consideration. Compared with these models, our f -CC model focuses on the flexibility of characterizing any relationship between the minimum degree and the number of vertices in a subgraph in various scales.

7.2 Diversified cohesive subgraph mining

Considerable work has been done on mining diversified cohesive subgraphs. An I/O-efficient approximation algorithm for mining diversified top- r cliques is proposed in [21]. Yang et al. [20] extends the notion of degree-based quasi-clique to temporal graphs and discovers diversified dense temporal subgraphs by a divide-and-conquer algorithm. Zhu et al. [22] extends the notion of k -core to multi-layer graphs and finds diversified top- r k -cores in a multi-layer graph. Recently, [19] devises a local search algorithm for the traditional diversified top- r cliques mining, which outperforms [21] on large sparse graphs. Fei et al. [9] proposes a formal concept analysis approach for diversified top- r cliques detection in Social Internet of Things. Since the models used to characterize cohesive subgraphs on these various kinds of graphs are the inflexible clique, quasi-clique or k -core models (or extended from these models), none of these algorithms fits for finding cohesive subgraphs of various scales simultaneously.

7.3 Redundancy-free cohesive subgraph mining

To discover meaningful (i.e., informative, less overlapping) cohesive subgraphs in a massive graph, one way is to maximize the diversity of the result set, which is adopted by our work and the work reviewed in Section 7.2. The other way is to reduce the redundancy of subgraphs in the result set. Boden et al. [4] defines a redundancy relationship between two cross-layer quasi-cliques in a multi-layer graph and designs algorithms to mine those cross-layer quasi-cliques that are redundancy-free and with high quality. This work provides us a new direction to investigate the problem of mining meaningful flexible cohesive subgraphs from the perspective of minimizing the redundancy in the result set. We will investigate this direction in our future work.

8 Conclusions and future work

The f -CC model is flexible in characterizing the relationship between the minimum degree of vertices and the number of vertices in a cohesive subgraph. The f -CC model is a generalization of the clique, quasi-clique and k -core models. The FlexCS problem is NP-hard and

is even NP-hard to be approximated within $|V|^{1-\epsilon}$ for all $\epsilon > 0$. The proposed FlexCS algorithm is efficient in finding r representative f -CCs with high diversity. It outperforms the diversified top- r cliques mining algorithm and the core decomposition algorithm in discovering multi-scale cohesive subgraphs in one batch. In the future, we plan to investigate how to define and solve the problem of mining meaningful flexible cohesive subgraphs from the perspective of minimizing the redundancy of subgraphs in the result set. In addition, the problem of flexible cohesive subgraph searching given a set of query vertices is also worth to study as a further work since users often pay more attention to some specific vertices other than the entire graph in many real world applications.

Appendix A Space and time complexity of Algorithm 2 (FlexCS)

We first analyze the space complexity of Algorithm 2. Given the input graph $G = (V, E)$, it takes $O(|V|)$ space to run the fast core decomposition algorithm CORE [3] on G and keep its results (line 3). By using the CCIndex proposed in Section 5.3, it takes $O(|V|)$ space to compute and maintain the connected components of each maximal k -core of G in the CCIndex (lines 6 and 15, see Section 5.3.2). As analyzed in Section 4, Procedure ConsCS invoked in line 17 takes $O(|V|)$ space. As mentioned in Section 3.2, the space for maintaining \mathcal{R} (lines 19–20) is $O(r|V|)$. By using the on-the-fly updating of the result set \mathcal{R} (Section 5.1), the qualified f -CCs are not necessary to be stored in the main memory. Consequently, the space complexity of Algorithm 2 is $O(r|V|)$.

Then, we analyze the time complexity of Algorithm 2. The fast core decomposition algorithm CORE [3] runs in $O(|V| + |E|)$ time (line 3). By using the CCIndex proposed in Section 5.3, it takes $O(\kappa(G)(|V| + |E|))$ time to compute the connected components of each maximal k -core of G for $1 \leq k \leq \kappa(G)$ (lines 6 and 15). Moreover, the ConsCS procedure invoked in line 17 discovers an f -CC in $O(|V|^2)$ time (see Section 4). As mentioned in Section 3.2, the Update procedure invoked in line 20 runs in $O(|V|)$ time. According to the seed selection strategies introduced in Section 4.2, at most $|V|$ f -CCs are generated in line 9 or by ConsCS in line 17. Thus, the total time for solving all ConsCS instances and updating \mathcal{R} with newly generated f -CCs are $O(|V|^3)$ and $O(|V|^2)$, respectively. Therefore, the time complexity of FlexCS is $O(|V|^3)$.

Appendix B Derivation of inequality (4)

First, let us consider the special case for $n_0 \geq n_1 \geq \dots \geq n_{\kappa(G)}$. We have

$$\begin{aligned} & (\kappa(G) + 1)n_{\kappa(G)} + \sum_{k=0}^{\kappa(G)-1} (k + 1) \max\{0, (n_k - n_{k+1})\} \\ &= (\kappa(G) + 1)n_{\kappa(G)} + \sum_{k=0}^{\kappa(G)-1} (k + 1)(n_k - n_{k+1}) \\ &= \sum_{k=0}^{\kappa(G)} n_k. \end{aligned}$$

Then, we consider the cases for $n_0 \geq n_1 \geq \dots \geq n_i, n_{i+1} \geq n_{i+2} \geq \dots \geq n_{\kappa(G)}$ and $n_i < n_{i+1}$, where $0 \leq i < \kappa(G)$. We have

$$\begin{aligned}
& (\kappa(G) + 1)n_{\kappa(G)} + \sum_{k=0}^{\kappa(G)-1} (k + 1) \max\{0, (n_k - n_{k+1})\} \\
&= (\kappa(G) + 1)n_{\kappa(G)} + \sum_{k=0}^{i-1} (k + 1)(n_k - n_{k+1}) + \sum_{k=i+1}^{\kappa(G)-1} (k + 1)(n_k - n_{k+1}) \\
&= \sum_{k=0}^{i-1} n_k + \sum_{k=i+2}^{\kappa(G)} n_k + (i + 2)n_{i+1} - i \cdot n_i \\
&> \sum_{k=0}^{\kappa(G)} n_k.
\end{aligned}$$

The last inequality holds because $(i + 2)n_{i+1} - i \cdot n_i > 2n_{i+1} > n_{i+1} + n_i$.

For the cases where there are $m > 1$ integers $i_1, i_2, \dots, i_m \in [0, \kappa(G) - 1]$ such that $n_{i_j} < n_{i_{j+1}}$ for all $j = 1, 2, \dots, m$, the proof is simply a generalization to the one given above for $m = 1$. The proof for the generalized cases is omitted.

Acknowledgements This work was partially supported by the National Natural Science Foundation of China (No. 62072138) and the Open Research Projects of Zhejiang Lab (No. 2021KC0AB02).

References

1. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection[C]. In: Latin American Symposium on Theoretical Informatics, pp. 598–612. Springer, Berlin Heidelberg (2002)
2. Ausiello, G., Boria, N., Giannakos, A., et al.: Online maximum k-coverage[J]. *Discret. Appl. Math.* **160**(13-14), 1901–1913 (2012)
3. Batagelj, V., Zaversnik, M.: An O(m) algorithm for cores decomposition of networks[J]. arXiv:0310049 (2003)
4. Boden, B., Günnemann, S., Hoffmann, H., et al: Mining coherent subgraphs in multi-layer graphs with edge labels[C]. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1258–1266 (2012)
5. Chang, L., Yu, J.X., Qin, L., et al: Efficiently computing k-edge connected components via graph decomposition[C]. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 205–216 (2013)
6. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis[J]. National security agency technical report. **16**, 3–29 (2008)
7. Feng, C., Wang, H., Tu, X.M., et al: A note on generalized inverses of distribution function and quantile transformation[J] (2012)
8. Gionis, A., Tsourakakis, C.E.: Dense subgraph discovery: Kdd 2015 tutorial[C]. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 2313–2314 (2015)
9. Hao, F., Pei, Z., Yang, L.T.: Diversified top-k maximal clique detection in social internet of things[J]. *Futur. Gener. Comput. Syst.* **107**, 408–417 (2020)
10. Hartmanis, J.: Computers and intractability: a guide to the theory of NP-completeness (michael r. gary and david s. johnson)[J]. *SIAM Rev.* **24**(1), 90 (1982)
11. Lee, V.E., Ruan, N., Jin, R., et al: A Survey of Algorithms for Dense Subgraph discovery[M]. *Managing and Mining Graph Data*, pp. 303–336. Springer, Boston, MA (2010)
12. Luce, R.D., Perry, A.D.: A method of matrix analysis of group structure[J]. *Psychometrika* **14**(2), 95–116 (1949)
13. Matsuda, H., Ishihara, T., Hashimoto, A.: Classifying molecular sequences using a linkage graph with their pairwise similarities[J]. *Theor. Comput. Sci.* **210**(2), 305–325 (1999)
14. Mokken, R.J.: Cliques, clubs and clans[J]. *Qual. Quant.* **13**(2), 161–173 (1979)

15. Pardalos, J., Resende, M.: On maximum clique problems in very large graphs[J]. DIMACS Ser. **50**, 119–130 (1999)
16. Sariyuce, A.E., Seshadhri, C., Pinar, A., et al: Finding the hierarchy of dense subgraphs using nucleus decompositions[C]. In: Proceedings of the 24th International Conference on World Wide Web, pp. 927–937 (2015)
17. Seidman, S.B.: Network structure and minimum degree[J]. Soc. Netw. **5**(3), 269–287 (1983)
18. Watts, D.J., Strogatz, S.H.: Collective dynamics of ‘small-world’ networks[J]. Nature **393**(6684), 440–442 (1998)
19. Wu, J., Li, C.M., Jiang, L., et al: Local search for diversified Top-k clique search problem[J]. Comput. Oper. Res. **116**, 104867 (2020)
20. Yang, Y., Yan, D., Wu, H., et al: Diversified temporal subgraph pattern mining[C]. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1965–1974 (2016)
21. Yuan, L., Qin, L., Lin, X., et al: Diversified top-k clique search[J]. VLDB J. **25**(2), 171–196 (2016)
22. Zhu, R., Zou, Z., Li, J.: Diversified coherent core search on multi-layer graphs[C]/2018. In: IEEE 34th International Conference on Data Engineering (ICDE), pp. 701–712. IEEE (2018)
23. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number[C]. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, pp. 681–690 (2006)

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.