



# Discovering key users for defending network structural stability

Fan Zhang<sup>1</sup> · Jiadong Xie<sup>2</sup> · Kai Wang<sup>3</sup> · Shiyu Yang<sup>1</sup> · Yu Jiang<sup>1</sup>

Received: 21 March 2021 / Revised: 27 May 2021 / Accepted: 1 June 2021 /  
Published online: 19 June 2021

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

## Abstract

The structural stability of a network reflects the ability of the network to maintain a sustainable service. As the leave of some users will significantly break network stability, it is critical to discover these key users for defending network stability. The model of  $k$ -core, a maximal subgraph where each vertex has at least  $k$  neighbors in the subgraph, is often used to measure the stability of a network. Besides, the coreness of a vertex, the largest  $k$  such that the  $k$ -core contains the vertex, is validated as the “best practice” for measuring the engagement of the vertex. In this paper, we propose and study the collapsed coreness problem: finding  $b$  vertices (users) s.t. the deletion of these vertices leads to the largest coreness loss (the total decrease of coreness from every vertex). We prove that the problem is NP-hard and hard to approximate. We show that the collapsed coreness is more effective and challenging than the existing models. An efficient greedy algorithm is proposed with powerful pruning rules. The algorithm is adapted to find the key users within a given time limit. Extensive experiments on 12 real-life graphs demonstrate the effectiveness of our model and the efficiency of the proposed method.

**Keywords** Network stability · Core decomposition · Graph edit

---

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

✉ Yu Jiang  
jjiangyu@gzhu.edu.cn

Fan Zhang  
zhangf@gzhu.edu.cn

Jiadong Xie  
xiejiadong0623@gmail.com

Kai Wang  
kai.wang@unsw.edu.au

Shiyu Yang  
syyang@gzhu.edu.cn

<sup>1</sup> Guangzhou University, Guangzhou, China

<sup>2</sup> East China Normal University, Shanghai, China

<sup>3</sup> University of New South Wales, Sydney, Australia

## 1 Introduction

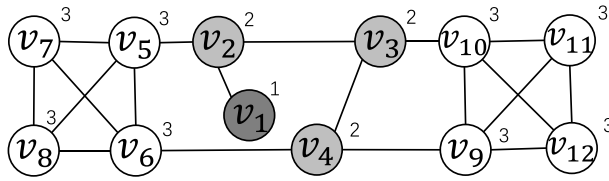
Complex networks are often modeled as graphs to process the entities/users and their relations. The structural stability of a graph is critical for maintaining the functionality of the graph, e.g., the running of a social network. In the dynamic of a network, some users may leave the network due to particular defects and/or attacks (e.g., user attraction strategies) from the competitor networks. The leave of key users will affect the engagement of their neighbors and the others s.t. the whole network becomes unstable. For instance, Friendster was a popular social network with more than 115 million users while it is collapsed due to the contagious user leave [20, 38].

As the departure of users may greatly break overall user engagement, i.e., the network stability, it is critical to discover these key users and conduct wise actions accordingly. We can motivate the key users to continuously engage in the network by proper actions, e.g., providing incentives for user engagement. The model of  $k$ -core is often utilized in the study of network stability for its effectiveness, e.g., [36]. It is defined as a maximal subgraph where each vertex has at least  $k$  neighbors in the subgraph [34, 37]. As the size of  $k$ -core can be used to estimate the (structural) stability of a network, the collapsed  $k$ -core problem is proposed to find  $b$  vertices (users) s.t. the deletion of these vertices leads to the smallest  $k$ -core. Nevertheless, this problem only extracts the key users from a partial group of users, i.e., the users related to the  $k$ -core with the input  $k$ . The result of the problem becomes different when a different  $k$  is given, and the best  $k$  is hard to be determined.

To address the above defects, we propose and study the collapsed coreness problem: finding  $b$  vertices s.t. the deletion of these vertices leads to the largest coreness loss (the total decrease of coreness from every vertex. The coreness of a vertex is validated as the “best practice” for measuring the engagement of the vertex [33]. Clearly, there is no input of  $k$  required for the collapsed coreness problem, and the only input is the budget  $b$  for the number of collapsers. The collapsed coreness problem essentially considers the  $k$ -core of every input  $k$  to find the key users, while its result is not a trivial aggregation. Since the collapsed coreness problem considers the coreness dynamic of each vertex, it is more promising in finding the key users regarding the stability of a network. The following example illustrates the difference between the collapsed  $k$ -core and collapsed coreness problems.

*Example 1* Figure 1 shows a graph  $G$  with 12 vertices and their connections. The coreness of each vertex is marked near the vertex, e.g., the coreness of  $v_4$  is 2. The  $k$ -core of  $G$  is induced by all the vertices with coreness of at least  $k$ , e.g., the 2-core is induced by  $v_2, v_3$  and  $v_4$ .

Table 1 shows the results of the collapsed  $k$ -core problem (CKP) and the collapsed coreness problem (CCP) when  $b = 1$ . If  $k = 2$ , the result of CKP may be  $v_3$ . The deletion of  $v_3$  will decrease the coreness of  $v_2$  which is named the follower of  $v_3$ . If  $k = 3$ , the result of CKP may be  $v_6$  and the followers are  $v_5, v_7$  and  $v_8$ . As  $k$  is not an input for CCP, its result is just  $v_5$  and the followers are  $v_2, v_6, v_7$  and  $v_8$ . It is more important to protect  $v_5$  compared with  $v_3$  or  $v_6$ , because the leave of  $v_5$  may affect more users.



**Figure 1** A Toy Graph (The coreness of each vertex is marked near the vertex)

From theoretical perspective, we prove the collapsed coreness problem is NP-hard and hard to approximate. It is essentially more challenging than the collapsed  $k$ -core problem, because there are more candidates and the computation of coreness loss is more costly (core decomposition v.s.  $k$ -core computation). We first analyze the problem in the case of one collapsed vertex, and prove several effective theorems. The lower and upper bounds are proposed to fast estimate the coreness loss of deleting a vertex, based on the core forest structure. An efficient way to compute the exact coreness loss for a collapser is introduced. With the integration of all the proposed techniques, a novel greedy algorithm is proposed to efficiently address the collapsed coreness problem. The algorithm is adapted to fast find the collapsers within a given time limit.

**Contributions** Our major contributions in this paper are as follows:

- Considering the coreness loss of each vertex, we propose the collapsed coreness problem to support the structural stability of a network.
- We prove the collapsed coreness problem is NP-hard, and APX-hard unless P=NP. The function of coreness loss is shown to be monotone but not submodular.
- An efficient greedy algorithm GCC is proposed with novel pruning techniques based on the theorems of one collapsed vertex, the upper/lower bounds of coreness loss, and the efficient computation of coreness loss.
- Our experiments on 12 real datasets show that the greedy GCC is more effective than other heuristics, and GCC outperforms the baseline algorithm on runtime by up to 3 orders of magnitude.

**Table 1** Collapsed  $k$ -core v.s. collapsed coreness in Figure 1

Problem	Input	Collapser	Followers	Coreness
Collapsed $k$ -Core	$k = 2, b = 1$	$v_3$	$v_2$	from 2 to 1
	$k = 3, b = 1$	$v_6$	$v_5, v_7, v_8$	from 3 to 2
Collapsed Coreness	$b = 1$	$v_5$	$v_2$	from 2 to 1
			$v_6, v_7, v_8$	from 3 to 2

**Roadmap** The rest of the paper is organized as follows. Section 2 formally defines the problem. Section 3 analyzes the hardness of the problem. Section 4 proposes the techniques and the GCC algorithm. The experiments are conducted in Section 5. More related works are surveyed in Section 6. Finally, Section 7 concludes the paper.

## 2 Preliminaries

We consider an undirected and unweighted simple graph  $G = (V, E)$ , with  $n = |V|$  vertices and  $m = |E|$  edges (assume  $m > n$ ). Given a vertex  $v$  in a subgraph  $S$  of  $G$ ,  $N(v, S)$  denotes the neighbor set of  $v$  in  $S$ , i.e.,  $N(v, S) = \{u \mid (u, v) \in E(S)\}$ . The degree of  $v$  in subgraph  $S$ , i.e.,  $|N(v, S)|$ , is denoted by  $d(v, S)$ . Table 2 summarizes the notations.

The model of  $k$ -core [34, 37] is defined as follows.

**Definition 1** ( $k$ -core  $C_k(\cdot)$ ) Given a graph  $G$  and an integer  $k$ , a subgraph  $S$  is a  $k$ -core of  $G$ , if (i) each vertex  $v \in S$  has at least  $k$  neighbors in  $S$ , i.e.,  $d(v, S) \geq k$ ; (ii)  $S$  is connected; and (iii)  $S$  is maximal, i.e., any supergraph of  $S$  is not a  $k$ -core except  $S$  itself.

The computation of  $k$ -core is to recursively delete each vertex with degree less than  $k$  in the graph. According to the definition of  $k$ -core, every vertex has a coreness value.

**Table 2** Summary of Notations (We may omit the target graph in notations when the context is clear, e.g., we may abbreviate  $N(v, S)$  to  $N(v)$ )

Notation	Definition
$G = (V, E)$	an undirected and unweighted simple graph
$n$	the number of vertices in $G$
$m$	the number of edges in $G$ ( $m > n$ )
$S$	a subgraph of $G$
$V(S)$	the set of vertices in $S$
$E(S)$	the set of edges in $S$
$E(v)$	the set of edges incident to vertex $v$ in $G$
$E(D)$	the set of edges incident to a vertex of $D$ in $G$
$N(v, S)$	the set of neighbors of $v$ in $S$
$d(v, S)$	degree of $v$ in $S$
$C_k(G)$	the $k$ -core of $G$
$c(v, S)$	coreness of $v$ in $S$
$D$	the set of collapsed vertices (i.e., collapsers)
$c^D(c, S)$	the coreness of $v$ in $S$ with the collapser set $D$
$b$	the budget for the number of collapsers
$l(D, G)$	the coreness loss of deleting $D$ in $G$
$F(x)$	the follower set of a collapser $x$
$F(D)$	the follower set of the collapsed set $D$
$T(v)$	the tree node containing $v$ in the core forest
$PS$	pivot set, i.e., $\{u \mid d(u, C_{c(u)}(G)) = c(u, G) \wedge u \in V(G)\}$

**Definition 2** (coreness  $c(\cdot)$ ) Given a graph  $G$ , the coreness of a vertex  $u \in V(G)$ , denoted by  $c(u, G)$ , is the largest  $k$  such that  $C_k(G)$  contains  $u$ , i.e.,  $c(u, G) = \arg \max_k u \in C_k(G)$ .

A graph can be decomposed in a hierarchy where the vertices are distinguished by their coreness values.

**Definition 3** (core decomposition) Given a graph  $G$ , core decomposition of  $G$  is to compute the coreness of every vertex in  $V(G)$ .

Core decomposition can be computed in  $O(m)$  time by recursively removing each vertex with the smallest degree in the graph, and update the degrees of its neighbors by bin sort [4].

Though the coreness of a user (vertex) in a network well estimates the engagement of the user [28, 33], the users with any coreness values may leave the network by outside forces, e.g, the user attraction strategy of a competing network. In this paper, such a leaved user (vertex) is named as a collapsar, and we say *they are collapsed* or *we collapse them*.

**Definition 4** (collapsars  $D$ ) Given a graph  $G$ , the set  $D$  of collapsars (i.e., collapsed vertices), are removed from the  $G$  regardless of their coreness values.

The collapsed (removed) vertices may change the corenesses of other vertices in  $G$ , i.e., the coreness values of some vertices will decrease. These vertices are called the followers of the collapsars in this paper. For each follower  $v$ , the coreness of  $v$  is changed (certainly decreased) by deleting  $D$  in  $G$ .

**Definition 5** (followers  $F(\cdot)$ ) Given a graph  $G$  and the collapsar set  $D$ , the follower set of  $D$  in  $G$  is denoted by  $F(D, G)$ . As the followers do not contain the collapsars, we have  $F(D, G) = \{v \mid v \in V(G - D) \wedge c(v, G) \neq c(v, G - \{D + E(D)\})\}$ .

We use  $c^D(u, G)$  (resp.  $c^x(u, G)$ ) to denote the coreness of  $u$  in  $G$  with  $D$  collapsed (resp. the vertex  $x$  collapsed). The loss of coreness by the collapsars is the sum of coreness decrease from each follower.

**Definition 6** (coreness loss  $l(\cdot)$ ) Given a graph  $G$  and the collapsar set  $D$ , the coreness loss of  $G$  regarding  $D$ , denoted by  $l(D, G)$ , is the total decrease of coreness for every vertex in  $V(G) \setminus D$ , i.e.,  $l(D, G) = \sum_{v \in V(G) \setminus D} (c(v) - c^D(v))$ .

For a graph with collapsars, the computation of core decomposition is to first delete the collapsars and then recursively remove each vertex with the smallest degree in the graph. The time complexity is still  $O(m)$ , because the only difference to core decomposition without collapsars is that we delete the collapsars first. The pseudo-code is given in Algorithm 1.

As the coreness loss is promising for evaluating the leave effect of the key users (e.g., the collapsars), we propose and study the collapsed coreness problem to find these important users according to the coreness loss.

**Problem definition** Given a graph  $G$  and a budget  $b$ , the *collapsed coreness problem* aims to find a set  $D$  of  $b$  vertices in  $G$  such that the coreness loss regarding  $D$  is maximized, i.e.,  $l(D, G)$  is maximized.

**Algorithm 1** CoreDecomp( $G, D$ ).

---

**Input** : a graph  $G$ , a set  $D$  of collapsed vertices  
**Output**:  $c^D(u, G)$  for each  $u \in V(G) \setminus D$

- 1 **foreach**  $u \in D$  **do**
- 2     **foreach**  $v \in N(u, G)$  **do**
- 3          $d(v) \leftarrow d(v) - 1$ ;
- 4     remove  $u$  and its incident edges from  $G$ ;
- 5  $k \leftarrow 1$ ;
- 6 **while** exist vertices in  $V(G) \setminus D$  **do**
- 7     **while**  $\exists u \in V(G) \setminus D$  with  $d(u) < k$  **do**
- 8         **foreach**  $v \in N(u, G)$  **do**
- 9              $d(v) \leftarrow d(v) - 1$ ;
- 10         remove  $u$  and its incident edges from  $G$ ;
- 11          $c^D(u, G) \leftarrow k - 1$ ;
- 12      $k \leftarrow k + 1$ ;
- 13 **return**  $c^D(u, G)$  for each  $u \in V(G) \setminus D$

---

### 3 Problem analysis

In this section, we analyze the hardness of the collapsed coreness problem.

**Theorem 1** *Given a graph  $G$ , the collapsed coreness problem is NP-hard.*

*Proof* We reduce the maximum coverage (MC) problem, which is NP-hard, to the collapsed coreness problem. Given an integer  $b$  and a collection of sets where each set contains some elements, the MC problem is to find at most  $b$  sets to cover the largest number of elements.

Consider an arbitrary instance  $H$  of MC with  $c$  sets  $T_1, \dots, T_c$  and  $d$  elements  $\{e_1, \dots, e_d\} = \cup_{1 \leq i \leq c} T_i$ , we construct a corresponding instance of the collapsed coreness problem on a graph  $G$ . W.l.o.g., we assume  $b < c < d$ , Figure 2 shows an example of the construction from 3 sets and 4 elements.

The graph  $G$  contains three parts:  $W$ ,  $P$ , and some 4-cliques. The part  $W$  contains  $c$  cliques of size 4, i.e.,  $W = \cup_{1 \leq i \leq c} W_i$  where each  $W_i$  corresponds to  $T_i$  of instance  $H$ . The part  $P$  contains  $d$  cliques of size 3, i.e.,  $P = \cup_{1 \leq i \leq d} P_i$  where each  $P_i$  corresponds to  $e_i$  of instance  $H$ .

For each set  $T_i$  ( $1 \leq i \leq c$ ) and each elements  $e_j$  ( $1 \leq j \leq d$ ), if  $e_j \in T_i$ , we add an edge  $(w_i, p_j^i)$  in  $G$  where  $w_i \in W_i$  and  $p_j^i \in P_j$ . Then, we attach a 4-clique for each vertex with degree 2 in current  $G$ . The construction of  $G$  is completed.

The degree of each vertex in  $G$  is at least 3. Recall that core decomposition iteratively deletes the vertices with degree less than  $k$  and assigns them the coreness of  $k - 1$  in current iteration, from  $k = 1, 2, \dots$  to  $k = k_{\max}$ . Thus, the coreness of each vertex in  $G$  is 3.

For each  $w_i \in W$ , if  $w_i$  is collapsed (deleted), the 4-clique containing  $w_i$  becomes a 3-clique, and the coreness of each vertex in  $P_j$  decreases by 1 if  $e_j \in T_i$ . Similarly, the coreness loss of collapsing any vertex  $v \in W_i \setminus \{w_i\}$  is same as the coreness loss of collapsing  $w_i$ . Note that it is not worthwhile to collapse more than 1 vertex in the 4-clique, because the coreness loss is less than collapsing the vertices in different 4-cliques.

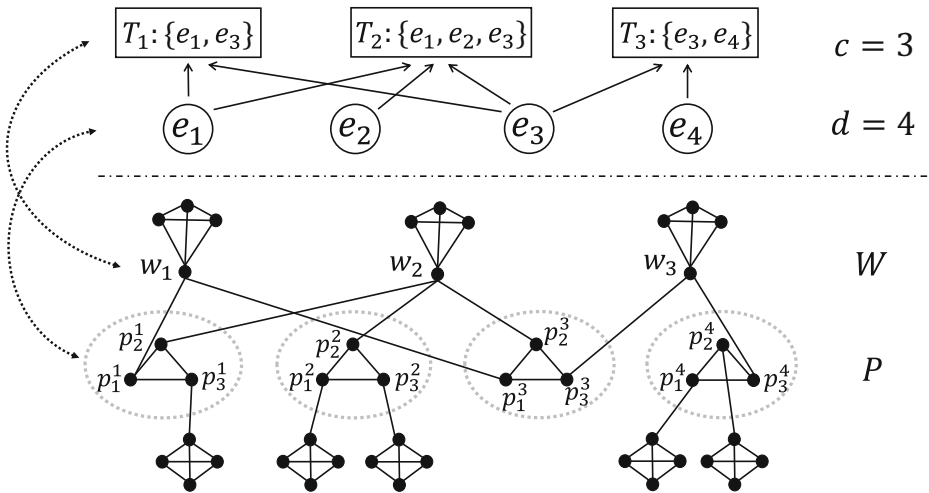


Figure 2 Construction example for hardness proofs

Besides, the coreness loss of collapsing the vertices in the 4-cliques attached to  $P$  is less than collapsing the same number of vertices in  $W$  where each 4-clique has one collapsed vertex.

Consequently, the optimal collapser set  $D$  for the collapsed coreness problem corresponds to the optimal collection  $I$  for the MC problem, where each vertex  $w_i \in D$  corresponds to the set  $T_i \in I$ . Since the MC problem is NP-hard, we have the collapsed coreness problem is NP-hard.  $\square$

Then, we prove that the collapsed coreness problem is hard to approximate.

**Theorem 2** Given a graph  $G$ , the collapsed coreness problem is APX-hard, unless  $P=NP$ .

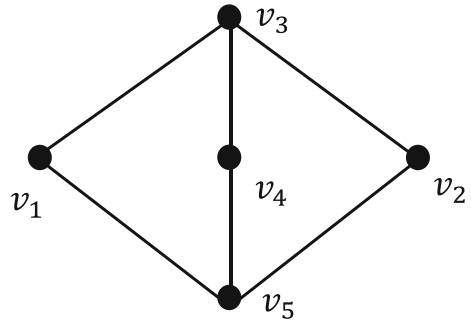
*Proof* We use the reduction from the MC problem same to the proof of Theorem 1. For any  $\epsilon > 0$ , the MC problem cannot be approximated in polynomial time within a ratio of  $(1 - 1/e + \epsilon)$ , unless  $P=NP$ . If there is a solution with  $\gamma$ -approximation on the coreness loss for the collapsed coreness problem, there will be a  $\lambda$ -approximate solution on optimal element number for the MC problem. Thus, there is no PTAS for the collapsed coreness problem and it is APX-hard unless  $P=NP$ .  $\square$

Besides, the function of coreness loss is monotone but not submodular.

**Theorem 3** The coreness loss function  $l(\cdot)$  is monotone but not submodular.

*Proof* As vertex deletion will not increase the coreness of any vertex,  $l(\cdot)$  is monotone. For two arbitrary collapser sets  $A$  and  $B$ , if  $l(\cdot)$  is submodular, it must hold that  $l(A) + l(B) \geq l(A \cup B) + l(A \cap B)$ . We consider a graph  $G$  where the vertex set  $V = \cup_{1 \leq i \leq 5} v_i$ , as shown in Figure 3. Suppose  $A = \{v_1\}$  and  $B = \{v_2\}$ , we have  $l(A \cup B) = 3$ ,  $l(A \cap B) = l(A) = l(B) = 0$ , and  $l(A) + l(B) = 0 < l(A \cup B) + l(A \cap B) = 3$ .  $\square$

**Figure 3** Example for non-submodularity



## 4 Our approach

As the collapsed coreness problem is NP-hard, we aim to design an effective heuristic algorithm that can return high-quality results efficiently. Although it is not promising to have an approximation guarantee on the heuristics, we find that the computation cost is relatively restricted when the target is to find one best collapsar. Besides, our preliminary experiments find that the result quality of a greedy heuristic is close to that of the optimal solution.

In this section, we first present the theoretical properties for one collapsed vertex, and the bounds to restrict the computation cost. Then, we introduce the algorithm for computing the followers, and the efficient GCC algorithm to greedily find promising collapsars.

### 4.1 Theorems for one collapsar

For the deletion of one vertex, the coreness change of the affected vertices is limited as shown by the following theorem.

**Theorem 4** *If a vertex  $x$  is collapsed (i.e., deleted) in  $G$ , any vertex  $u \in V(G) \setminus \{x\}$  may reduce its coreness by at most 1.*

*Proof* Suppose there is a vertex  $u \in V(G) \setminus \{x\}$  with coreness decreasing from  $k$  to  $k'$  after deleting  $x$  and  $k > k' + 1$ . Let  $M$  be the  $k$ -core before  $x$  is deleted, we have  $u \in M$  and  $d(v, M) \geq k$  for every vertex  $v \in M$ . If we delete  $x$  and its corresponding edges from  $M$ , we have  $d(v, M \setminus \{x \cup E(x)\}) \geq k - 1$  for every vertex  $v \in M$  due to at most one edge is removed for each vertex  $v \in M$ . Thus,  $M \setminus \{x \cup E(x)\} \subseteq C_{k-1}(G)$ . As  $u \in M$  and  $u \neq x$ , we have  $u \in C_{k-1}(G)$  and thus  $k' \geq k - 1$  which causes a contradiction.  $\square$

According to the theorem, the coreness of each follower of  $x$  decreases by exactly 1 for collapsing (only)  $x$ . Thus, for the collapse of one vertex, the number of followers is equal to the coreness loss.

Then, we show that the followers of a collapsar always have a smaller coreness than the coreness of the collapsar.

**Theorem 5** *If a vertex  $x$  is collapsed in  $G$  and the vertex  $u$  is a follower of vertex  $x$  (i.e.,  $u \in F(x, G)$ ), we have  $c(u) \leq c(x)$ .*



*Proof* For any integer  $k > c(x)$ , the  $k$ -core of  $G$  is same to the  $k$ -core of  $G - \{x \cup E(x)\}$  according to the computation of  $k$ -core, i.e., recursively deleting each vertex with degree less than  $k$ . Thus, any vertex  $v$  with  $c(v) > c(x)$  is not a follower of  $x$ .  $\square$

Note that every vertex  $u$  with the same coreness to  $x$  may have its coreness decreased by collapsing  $x$ , despite the deletion sequence of  $u$  and  $x$  in core decomposition.

For the collapse of one vertex, to find out the effective candidate collapsers, we define the pivot set as follows.

**Definition 7** (pivot set  $PS$ ) For each  $k$ -core, the pivots are the vertices with degree equal to coreness in the  $k$ -core. The pivot set of the graph  $G$ , denoted by  $PS$ , is the union of all the pivots for each  $k$ -core and each integer  $k$ , i.e.,  $PS = \{u \mid d(u, C_{c(u)}(G)) = c(u, G) \wedge u \in V(G)\}$ .

The coreness of a pivot  $u$  is relatively unstable because it will decrease if any neighbor of  $u$  in  $c(u)$ -core is collapsed. Thus, we can find the set of candidate collapsers based on the pivot set for the collapse of one vertex.

**Theorem 6** Given a graph  $G$ , if a vertex  $x$  is collapsed and it has at least one followers,  $x$  is from  $X$  where  $X = \{u \mid u \in N(v, G) \wedge v \in PS \wedge c(u) \geq c(v)\}$ ; that is  $|F(x, G)| > 0$  implies  $x \in X$ .

*Proof* For any vertex  $u \in V(G) \setminus X$  and any vertex  $v \in N(u, G)$ , we have  $v$  is not a pivot, and  $c(v, G) > d(v, C)$  where  $C$  is any  $k$ -core containing  $v$ . Thus, the collapse of  $u$  will not decrease the coreness of any neighbor of  $u$ , i.e., there is no follower for collapsing  $u$ .  $\square$

*Example 2* In Figure 1, the coreness of each vertex is attached to the vertex. For 1-core, the pivot is  $v_1$  as  $c(v_1) = d(v_1, C_1(G)) = 1$ . For 2-core, the pivot is  $v_2$  as  $c(v_2) = d(v_2, C_2(G)) = 2$ . For 3-core, the pivot set is  $\cup_{5 \leq i \leq 12} \{v_i\}$ . Thus, we have  $X = V(G) \setminus \{v_1, v_4\}$ , i.e., each of  $\{v_1, v_4\}$  has no follower while any other vertex has at least one follower.

The theorems in this section indicate that computing the collapse of one vertex can be largely optimized to reduce the time cost, because the cost of follower computation is limited (Theorem 4 and 5) and the candidate collapsers are restricted (Theorem 6). Naturally, a greedy algorithm to find one good collapser in each iteration may be promising.

## 4.2 Core forest structure

Here we introduce the core forest structure of core decomposition which can be used to design effective optimizations (e.g., the bounding techniques in next subsection).

For every integer  $k$ , we have (i) a  $k$ -core is contained in exactly one  $(k - 1)$ -core, and (ii) each  $k$ -core is disjoint from each other. Thus, all the  $k$ -cores in the graph (for every possible  $k$ ) can be organized into a forest structure, i.e., the core forest, where each connected component of the graph corresponds to a tree in the core forest.

**Tree nodes in core forest** For every  $k$ -core  $C$  in  $G$ , if there is at least one vertex in  $C$  with coreness  $k$ , the  $k$ -core  $C$  is associated with a tree node  $N$ . The node  $N$  stores all the the vertices in  $C$  with coreness  $k$  (i.e.,  $\{u \mid u \in C \wedge c(u, G) = k\}$ ). A tree node can only have

one associated  $k$ -core. Let  $T$  denote the core forest. We use  $T[v]$  to denote the tree node containing  $v$ , and  $T_i.V$  to denote the vertex set in node  $T_i$ .

**Tree edges in core forest** Given a  $k_1$ -core  $C_1$  associated with tree node  $N_1$ , and a  $k_2$ -core  $C_2$  associated with tree node  $N_2$ , the node  $N_1$  is the parent node of  $N_2$ , if (i)  $k_1 < k_2$ ; (ii)  $C_1 \supset C_2$ ; and (iii) any tree node associated with a  $k'$ -core is not the parent of  $N_2$ , where  $k_1 < k' < k_2$ .

*Example 3* Figure 4 depicts the core forest of the graph  $G$  in Figure 1, where  $G$  is shown at the left and the corresponding  $T$  is at the right. As there is only one 1-core (resp. 2-core), we have  $T[v_1] = v_1$  and  $T[v_2] = \{v_2, v_3, v_4\}$ . As there are two (connected) 3-cores, we have  $T[v_5] = \{v_5, v_6, v_7, v_8\}$  and  $T[v_9] = \{v_9, v_{10}, v_{11}, v_{12}\}$ .

**Constructing the core forest** The state-of-the art algorithm for constructing the forest of  $k$ -core is LCPS (Level Component Priority Search) [34] with time complexity of  $O(m)$ . The algorithm pushes every vertex and its unvisited neighbors into queues (according to a priority function) s.t. the subtree containing the vertex is built.

### 4.3 Bounds of follower number

In a greedy algorithm, we need to compute the coreness loss for each vertex s.t. the best collapser can be found for the collapse of one vertex, in each iteration. However, it is costly to compute such coreness loss. In this subsection, we aim to fast estimate the lower/upper bounds of coreness loss for each vertex (if it is collapsed). Note that the number of followers is equal to the coreness loss for one collapsed vertex.

**Lower bound of follower number** For a vertex  $x$ , we get the lower bound of followers from its neighbors which are pivots.

**Theorem 7** Given a graph  $G$ , if a vertex  $x$  is collapsed, we have  $|F(x, G)| \geq lb(x)$ , where

$$lb(x) = |\{u \mid u \in N(x, G) \cap PS \wedge c(u) \leq c(x)\}|$$

*Proof* For each vertex  $u \in N(x, G) \cap PS$  with  $c(u) \leq c(x)$ , we have  $d(u, C_{c(u)}(G)) = c(u, G)$ , and its degree in  $c(u)$ -core will decrease by 1 after deleting  $x$ , i.e.,  $d(u, C_{c(u)}(G \setminus$

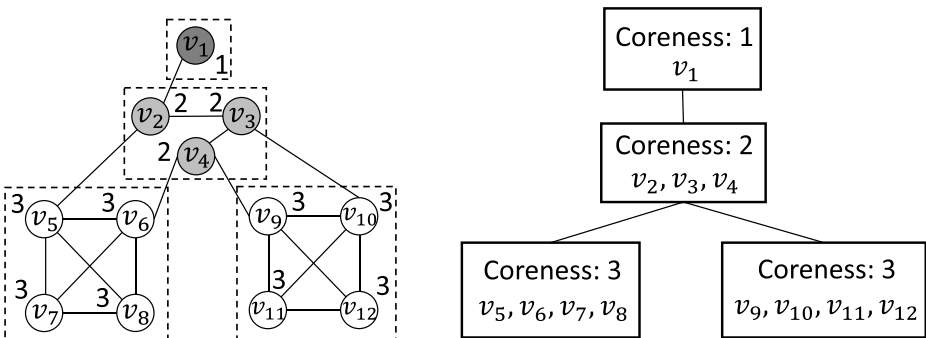


Figure 4 Core component tree

$\{x \cup E(x)\}) \geq c(u, G) - 1$ . Thus, the coreness of  $u$  decreases by 1 after deleting  $x$ , and  $u$  is a follower of  $x$ .  $\square$

The lower bound can be more accurate if we consider  $h$ -hop neighbors of  $x$ , i.e., the vertices affected by the neighbors of  $x$  with coreness decreased. The most accurate version becomes the exact computation of the follower number. Our preliminary experiments find that the additional cost of bound computation is not worthwhile, and the proposed lower bound well balances the computation cost and the bound accuracy.

**Upper bound of follower number** For a vertex  $x$ , we can get the upper bound of follower number from its own tree node  $T[x]$  and the tree nodes containing its neighbors which are pivots. Specifically, the upper bound of follower number for a collapsed  $x$ , denoted by  $ub(x)$ , is the number of vertices in the tree nodes where each node has at least one pivot which is a neighbor of  $x$ .

**Theorem 8** *Given a graph  $G$ , if a vertex  $x$  is collapsed, we have  $|F(x, G)| \leq ub(x)$ , where*

$$ub(x) = |\mathcal{T}|, \text{ and } \mathcal{T} = \bigcup_{v \in N(x, G) \cap PS \wedge c(v) \leq c(x)} T[v].V \setminus \{x\}$$

*Proof* Let  $T'$  be any tree node NOT in  $\mathcal{T}$ , and  $u$  be any vertex in  $T'.V$ . (i) If  $c(u) > c(x)$ , the coreness of  $u$  keeps same on  $G$  and  $G - \{x \cup E(x)\}$  by Theorem 5. (ii) If  $c(u) \leq c(x)$ , every vertex  $v \in T'.V$  is a non-pivot neighbor of  $x$  or is not a neighbor of  $x$ , i.e.,  $v \notin N(x, G) \cap PS$ . So, we have  $C_{c(v)}(G - \{x \cup E(x)\}) = C_{c(v)}(G) - \{x \cup E(x)\}$  by Theorem 4: [ii.a] for  $v \in N(x, G)$  and  $v \notin PS$ , we have  $d(v, C_{c(v)}(G)) > c(v, G)$  and the degree of  $v$  in  $c(v)$ -core decreases by 1 after the deletion of  $x$ . So, the coreness of  $v$  is same in  $G$  and  $G - \{x \cup E(x)\}$ ; [ii.b] for  $v \notin N(x, G)$ , the degree of  $v$  is same in  $c(v)$ -core of  $G$  and  $G - \{x \cup E(x)\}$ . Thus,  $c(v, G) = c(v, G - \{x \cup E(x)\})$  for every vertex  $v \in T'.V$ , and there is no follower of  $x$  in  $T'$ . Since each vertex in  $V(G)$  exists in one and only one tree node of  $\mathcal{T}$ , and  $x$  is not a follower of itself,  $ub(x)$  is a correct upper bound of  $|F(x, G)|$ .  $\square$

---

**Algorithm 2** CalcBound( $G, PS$ ).

---

```

Input : a graph  $G$ , the pivot set  $PS$  of  $G$ 
Output: the upper bound  $ub(x)$  and the lower bound  $lb(x)$  of each vertex  $x \in V(G)$ 
1 foreach  $x \in V(G)$  do
2    $ub(x) \leftarrow 0, lb(x) \leftarrow 0;$ 
3   each tree node of  $T \leftarrow$  unvisited;
4   foreach  $u \in N(x, G) \cap PS$  and  $c(u) \leq c(x)$  do
5      $lb(x) \leftarrow lb(x) + 1;$ 
6     if  $T(u)$  is unvisited then
7        $ub(x) \leftarrow ub(x) + |T(u).V \setminus \{x\};$ 
8        $T(u) \leftarrow$  visited;
9 return  $ub(x)$  and  $lb(x)$  of each  $x \in V(G)$ 

```

---

*Example 4* For the graph in Figure 4, we have  $PS = V(G) \setminus \{v_3, v_4\}$ . If  $v_5$  is collapsed (the others are not collapsed), the set  $\{u \mid u \in N(v_5, G) \cap PS \wedge c(u) \leq c(v_5)\} =$

$\{v_2, v_6, v_7, v_8\}$ . Therefore, we have  $lb(v_5) = 4$ ,  $ub(v_5) = |T[v_2].V \cup T[v_6].V \setminus \{v_5\}| = |\{v_2, v_3, v_4, v_6, v_7, v_8\}| = 6$ , and  $4 \leq |F(v_5, G)| \leq 6$ . If  $v_6$  is collapsed (the others are not collapsed), we have  $\{u \mid u \in N(v_6, G) \wedge u \in PS \wedge c(u) \leq c(v_6)\} = \{v_5, v_7, v_8\}$ . So,  $lb(v_5) = 3$ ,  $ub(v_6) = |T[v_5].V \setminus \{v_6\}| = 3$ , and  $|F(v_6, G)| = 3$ . Similarly, we have  $lb(v_1) = ub(v_1) = lb(v_4) = ub(v_4) = 0$ ,  $lb(v_2) = ub(v_2) = 1$ ,  $lb(v_3) = 1 < ub(v_3) = 2$ , and  $ub(u) = lb(u) = 3$  for any other vertex  $u$ .

Algorithm 2 shows the pseudo-code for computing the upper bound and lower bound of follower number for each vertex. For each  $x \in V(G)$  (Line 1), we initialize the upper bound  $ub(x)$  and lower bound  $lb(x)$  at Line 2. The tree nodes are set to unvisited or updated to unvisited (for latter iterations). Then for each non-pivot neighbor  $u$  of  $x$  with  $c(u) \leq c(x)$  (Line 4), we count the lower bound (Line 5) and upper bound (Line 6-8) of  $x$ . At Line 9, the algorithm returns the bounds for each vertex.

The time complexity of Algorithm 2 is  $O(m)$ , because each edge is visited by at most twice for neighbor query (Line 4), each vertex is visited by at most twice (Line 1 and 4) for enumeration, and the tree nodes are visited along with the edge visit within  $O(m)$  times.

#### 4.4 Computation of followers

As there are still many candidates collapsers for a greedy solution, it is cost-prohibitive to compute the followers by core decomposition from scratch. The algorithm for core maintenance can be used to compute (update) the follower set efficiently, which is to update the coreness of each vertex against edge insertion or deletion. The state-of-the-art algorithm for core maintenance is proposed in [49]. The deletion of a vertex can be regarded as the deletion of its incident edges.

---

#### Algorithm 3 CalcFollower( $G, x$ ).

---

**Input** : a graph  $G$ , a collider  $x$   
**Output**: the follower set  $F(x)$   
**1** **foreach**  $u \in N(x, G)$  and  $c(u) \leq c(x)$  **do**  
**2**    $\lfloor$  delete edge  $(u, x)$  from  $G$  and run core maintenance [49];  
**3**    $F \leftarrow$  the set of vertices with coreness changed;  
**4** **return**  $F$

---

Algorithm 3 shows the pseudo-code for computing the follower set if  $x$  is collapsed. For each  $x$ 's neighbor  $u$  with  $c(u) \leq c(x)$  (Line 1), we delete the edge  $(u, x)$  from  $G$  and update the coreness of each vertex by core maintenance [49] (Line 2). Note that a neighbor  $u$  with  $c(u) > c(x)$  is not visited at Line 1, because it cannot be a follower of  $x$  by Theorem 5. Since the coreness of each vertex may decrease by at most 1 for the collapse of  $x$  (Theorem 4), we simply record the vertices with coreness changed after running Line 2 and accumulate it to the set  $F$  (Line 3 and 4).

According to the time cost of core maintenance [49], the time complexity of Algorithm 3 is  $O(\sum_{u \in N(x, G)} \sum_{w \in F_{(x, u)}} d(w))$  where  $F_{(x, u)}$  is the set of vertices with coreness decreased after the deletion of  $(x, u)$ .

Although the average time cost of Algorithm 3 is not large for one collapsed vertex, considering the large number of candidate collapsers to compute, it is necessary to apply the lower/upper bounds in Section 4.3 s.t. unpromising candidates can be immediately pruned (i.e., unnecessary follower computations are skipped).

## 4.5 The greedy algorithm (GCC)

In this subsection, we introduce the final GCC algorithm which integrates all the proposed techniques. The aforementioned theorems and techniques naturally suggest a greedy algorithm as the computation for one collapsed vertex can be significantly optimized by them. In each of the  $b$  iteration, GCC greedily finds a best collapser by computing/estimating the follower number of each candidate collapser.

Algorithm 4 shows the detail of our GCC algorithm. We use  $c_{min}$  to record the smallest core-ness of the vertices with core-ness decreased in last iteration, and use  $D$  to denote the set of collapsers (Line 1). We firstly apply core decomposition (Algorithm 1) to get the initial core-ness of each vertex (Line 2). The pivot set  $PS$  can be retrieved along with the core decomposition when the degree threshold  $k$  increases (from the computation of  $k$ -core to  $(k+1)$ -core). Then, we apply LCPS [34] to build the core forest  $T$  (Line 3). The upper bound  $ub(\cdot)$  and lower bound  $lb(\cdot)$  of follower number are computed by Algorithm 2 (Line 4).

---

### Algorithm 4 GCC( $G, b$ )

---

**Input** : a graph  $G$ , the budget  $b$  for the number of collapsers

**Output** : the set  $D$  of collapsed vertices

```

1  $D \leftarrow \emptyset$ ;
2 pivot set  $PS$ , core-ness  $c(u)$  of each  $u \in V(G) \leftarrow \text{CoreDecomp}(G, \emptyset)$ ;
3  $T \leftarrow \text{LCPS}(G, c(\cdot))$ ;
4  $ub(x), lb(x)$  of each  $x \in V(G) \leftarrow \text{CalcBound}(G)$ ;
5 for  $i$  from 1 to  $b$  do
6    $l_{max} \leftarrow 0$ ;
7   foreach  $u \in \{w \mid w \in N(v, G) \wedge v \in PS \wedge c(w) \geq c(v)\} \setminus D$  with decreasing
   order of  $ub(u)$ /*Theorem 6*/ do
8     if  $ub(u) > l_{max}$  /*Theorem 8*/ then
9       if  $lb(u) = ub(u)$ /*Theorem 7, 8*/ then
10         $l(u) = lb(u)$ ;
11       else
12         $F(u) \leftarrow \text{CalcFollow}(u, G)$ ;  $l(u) = |F(u)|$ ;
13        if  $l(u) > l_{max}$  then
14           $l_{max} = l(u)$ ;  $x = u$ ;
15    $D \leftarrow D \cup \{x\}$ ;
16   delete  $x$  and its incident edges from  $G$ ;
17   update  $c(\cdot)$ ,  $PS$  and  $T$  by (hierarchical) core maintenance;
18 return  $D$ 

```

---

In each of the  $b$  iterations (Line 5), we use  $x$  to record the best collapser found so far, and use  $l_{max}$  to record the follower number (i.e. core-ness loss) of the best collapser (Line 6). Then, we enumerate each vertex with non-zero follower and not in  $D$  in decreasing order of their upper bounds based on Theorem 6 (Line 7). A vertex with larger upper bound often has more followers in our experiments, and the computation of such vertex at an early stage may speed up the algorithm because it fast produces a large  $l_{max}$ .

Only when the upper bound of a vertex  $u$  is larger than  $l_{max}$  (Line 8), it is worthwhile to compute the follower set of  $x$ , i.e.,  $l_{max}$  and  $x$  may be updated. If the lower bound and upper bound of  $u$  are the same, then the coreness loss of  $u$  is simply  $lb(u)$  (Line 9-10); Otherwise, we compute the follower set of  $u$  and retrieve its coreness loss by Algorithm 3 (Line 11-12). If  $l(u)$  is larger than  $l_{max}$ ,  $u$  becomes the current best collapse (Line 13-14).

After we determine the best collapse  $x$  in current iteration, we will update the set  $D$  (Line 15) and delete  $x$  with its incident edges from  $G$  (Line 16). The coreness  $c(\cdot)$  of each vertex is updated by core maintenance [49] where  $PS$  is updated together, and the core forest  $T$  is updated by hierarchical core maintenance [27] (Line 17). After  $b$  iterations, Algorithm 4 returns the set  $D$  of  $b$  collapsed vertices (Line 18).

The time complexity of Algorithm 4 is  $O(\sum_{i=1}^b \sum_{x \in CV_i} Time_f(x))$  where  $CV_i$  is the candidate vertices in  $i^{th}$  iteration for finding a best collapse and  $Time_f(x)$  is the time cost of Algorithm 3 with the input of  $x$ . Note that the overall time cost of core decomposition, core forest construction, and hierarchical core maintenance is not dominated for Algorithm 4.

To find high-quality collapses within a time limit, we may first choose  $b$  collapses with the largest vertex degrees, and then replace each collapse (with the smallest coreness loss for one collapsed vertex) by the collapses produced by Algorithm 4 if the total coreness loss can be larger.

## 5 Experiments

In this section, we conduct extensive experiments to verify the effectiveness of the collapsed coreness model and the efficiency of our proposed algorithms.

### 5.1 Experimental setting

**Datasets** The experiments are conducted on 12 real-life datasets. Table 3 shows the statistics of the datasets, ordered by the number of edges in each dataset, where  $d_{avg}$  is the average

**Table 3** Statistics of datasets

Dataset	$n$	$m$	$d_{avg}$	$d_{max}$	$k_{max}$
Facebook	4,039	88,234	43.7	1045	75
Brightkite	58,228	194,090	6.7	1098	52
Socfb-Northeastern19	13,882	381,934	77.1	1886	43
Arxiv	34,546	421,578	24.4	846	30
Gowalla	196,591	456,830	9.2	10721	51
Socfb-Syracuse56	13,653	543,982	79.7	1340	75
NotreDame	325,729	1,497,134	6.5	3812	155
Stanford	281,903	2,312,497	16.4	38626	71
Youtube	1,134,890	2,987,624	5.3	28754	51
Soc-Catster	149,700	5,448,197	72.8	80636	419
DBLP	1,566,919	6,461,300	8.3	2023	118
LiveJournal	3,997,962	34,681,189	17.4	14815	360

**Table 4** Summary of algorithms

Algorithm	Description
Exact	identifies the optimal solution by searching all possible combinations of $b$ collapsers
Rand	randomly chooses $b$ collapsers from $V(G)$
Deg	chooses $b$ collapsers from $V(G)$ with the highest degree
Deg-C	chooses $b$ collapsers with the highest value of $d(u, G) - c(u)$ for each $u \in V(G)$
Core	chooses $b$ collapsers from $V(G)$ with the highest coreness
CKC	the state-of-the-art algorithm for collapsed $k$ -core problem [47]
GCC	our greedy algorithm (Algorithm 4)
GCC-B	GCC without bound pruning (Section 4.3)
GCC-P-B	GCC-B without applying Theorem 6
GCC-O-P-B	GCC-P-B without applying core maintenance in Section 4.4 (which is replaced by core decomposition)
GCC-T	within a given time limit, first use Deg to choose $b$ collapsers, and then replace each collapser (with the smallest coreness loss for one collapsed vertex) by the collapsers produced by GCC if the total coreness loss can be larger

vertex degree,  $d_{max}$  is the largest vertex degree and  $k_{max}$  is the largest vertex coreness. The datasets are obtained from SNAP<sup>1</sup> and Network Repository.<sup>2</sup>

**Algorithms** For effectiveness, we mainly compare 7 algorithms with our GCC algorithm, including 4 heuristics, the exact solution, the algorithm for collapsed  $k$ -core problem, and the GCC-T algorithm. For efficiency, we incrementally equip the baseline with our proposed techniques to evaluate the performance. Table 4 lists all the evaluated algorithms.

**Parameters** We perform our experiments on a CentOS Linux server (Release 7.5.1804) with Quad-Core Intel Xeon CPU (E5-2640 v4 @ 2.20GHz) and 128G memory. All the algorithms are implemented in C++. The source code is compiled by GCC(7.3.0) under -O3 optimization.

## 5.2 Performance evaluation

**Comparison with exact solution** We first compare the result of GCC with the Exact algorithm which identifies the optimal  $b$  collapses by enumerating all possible combinations of  $b$  vertices. Due to the huge time cost of Exact, we build small datasets by iteratively extracting a vertex and all its neighbours from Brightkite or Arxiv, until the number of extracted vertices reaches 20. We randomly extract 100 subgraphs, and then calculate the average coreness loss of them. Figure 5 shows that the coreness loss of GCC is at least 94.7% (resp. 96.2%) of the optimal value in Brightkite (resp. Arxiv). In addition, GCC is faster than Exact by up to 6 orders of magnitude, as shown by the time in the figures.

<sup>1</sup><http://snap.stanford.edu>

<sup>2</sup><http://networkrepository.com>

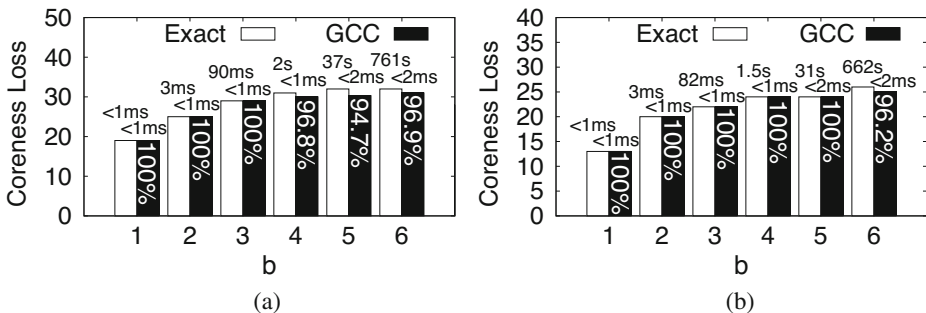
**Table 5** Coreness loss from CKC and GCC

Dataset	F.	A.	B.	G.	S.	N.	Y.	D.	L.	SS.	SN.	SC.
$max_{CKC}$	76%	77%	84%	93%	57%	70%	97%	55%	36%	80%	75%	98%
$avg_{CKC}$	54%	53%	47%	60%	23%	26%	73%	27%	19%	52%	55%	82%

**Comparison with CKC** Table 5 shows that the largest coreness loss (denoted by  $max_{CKC}$ ) that CKC can achieve on all the datasets, which is computed by running CKC with every possible input of  $k$ . Specifically, for the collapser set  $C$  computed by CKC, we compute the total sum of coreness loss for every coreness value if  $C$  is collapsed. It is reported that the largest coreness loss by CKC is only 36% – 98% of the coreness loss by GCC). Table 5 also shows that the average coreness loss of CKC (denoted by  $avg_{CKC}$ ) for an input of  $k$ , which is 19% – 82% of the coreness loss by GCC. Besides, Figure 6 shows the coreness loss of different  $k$  for CKC on different datasets, which indicates that CKC is infeasible to solve the collapsed coreness problem.

We also investigate the distribution of collapsers and followers regarding their coreness values, from GCC and CKC, respectively. Let  $CKC_{num}$  denote CKC with  $k = num$ , e.g.,  $CKC_{10}$  is CKC with  $k = 10$ . We observe that the distribution of CKC highly depends on the choice of  $k$  value. In Figure 7a, the collapsers from  $CKC_{10}$  have small coreness values while the collapsers from GCC have a relatively uniform distribution on coreness. In Figure 7b, for the distribution of followers, the margin is smaller while GCC has more followers than CKC on almost every coreness value.

**Comparison with other heuristics** In Figure 8, we compare the coreness loss from GCC with other heuristics (Rand, Deg, Deg-c, and Core). The details of these heuristics are introduced in Table 4. In Figure 8a, the performance of Rand is the worst as it chooses random vertices to collapse, where the results are omitted if the values are too small. The performance of Core is better than Rand as they choose vertices with large coreness to collapse, which may affect many vertices. Deg often has a larger coreness loss than Core. It indicates that the degree of a vertex is more relevant to the power of collapse for the collapsed coreness problem, compared with the coreness of the vertex. Deg-C represents the redundancy of a vertex on its coreness value towards its degree, which performs almost the same to Deg (better on Facebook). Compared with the above heuristics, GCC always



**Figure 5** Exact v.s. GCC on small graphs



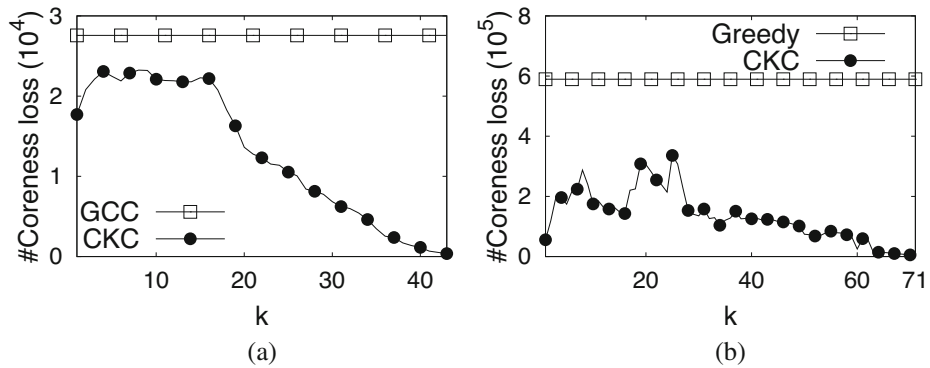


Figure 6 Coreness Loss on Different  $k$

has the largest coreness loss. In Figure 8b and c, we can see the result of Deg is similar to GCC when  $b$  is very small, but the margin becomes much larger with the increase of  $b$ .

**Case study on DBLP** We evaluate the effectiveness of our model on a graph which is built on DBLP, where each vertex is an author and two authors are linked if they have more than 5 co-authored papers in top Database conferences including SIGMOD, VLDB, and ICDE. Figure 9 depicts the collapser identified by GCC with  $b = 1$  as well as the corresponding followers. The collapser is the author “Hector Garcia-Molina” who has 66 followers. Here 24 of the followers are not the 1-hop neighbors of “Hector” which means a large amount of followers does not have direct relations to the collapser. It implies that Deg is not a good choice for the problem. Besides, we observe that the followers include many computer scientists, and many of them are ACM fellows, e.g., “Magdalena Balazinska”, “Stanley B. Zdonik”, and so on. The followers are also from different universities and institutions in different countries. The above finding indicates that the engagement of many users may be affected if the collapser left a network.

**Performance of time-dependent GCC** As shown in Figure 8, the algorithm Deg performs well when  $b$  is small. This motivates us to design a time-dependent algorithm GCC-T which first uses Deg to choose  $b$  collapsers, and then applies GCC to replace the collapsers s.t. the coreness loss can be larger, within a given time limit. Figure 10 shows the performance

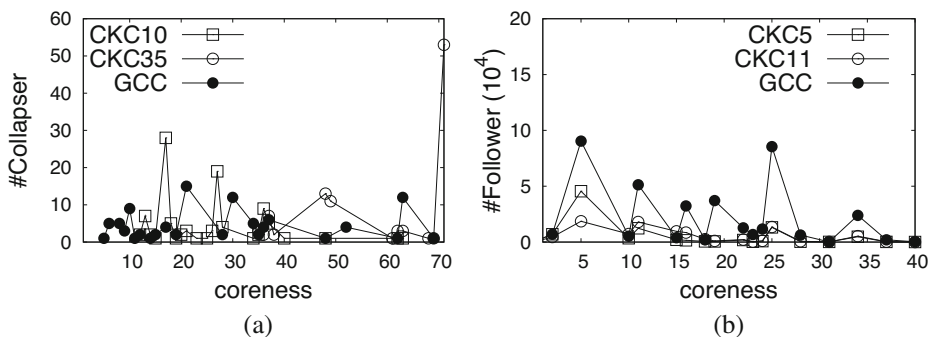


Figure 7 Distributions of collapsers and followers on Stanford

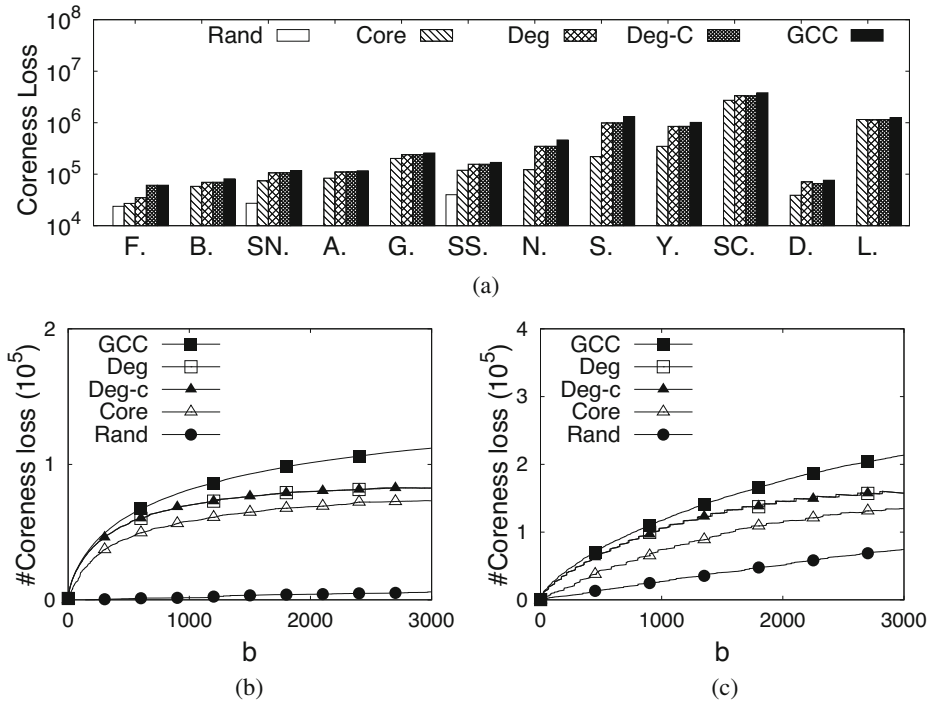


Figure 8 Coreness loss from different heuristics

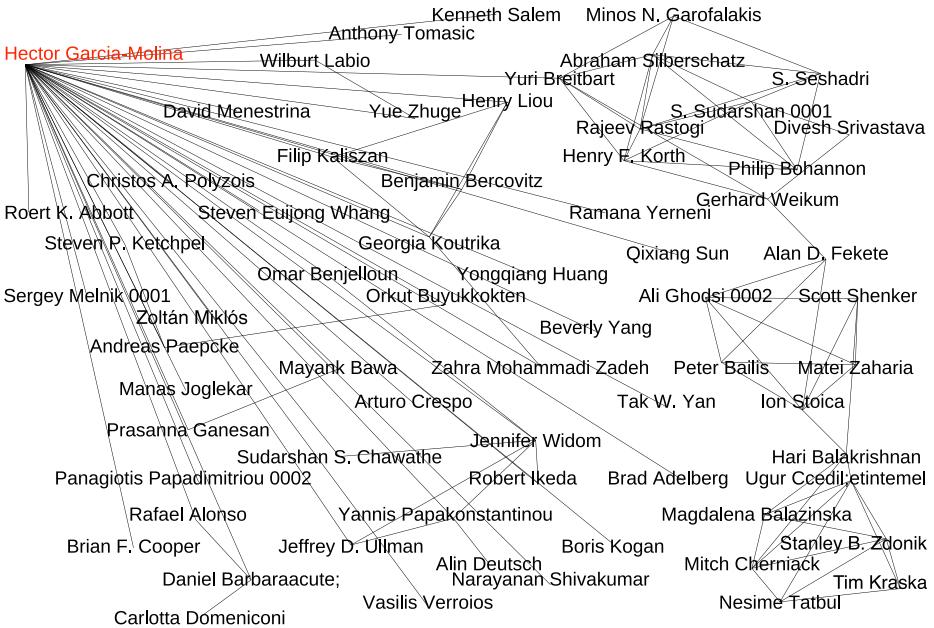


Figure 9 Case study on DBLP,  $b = 1$

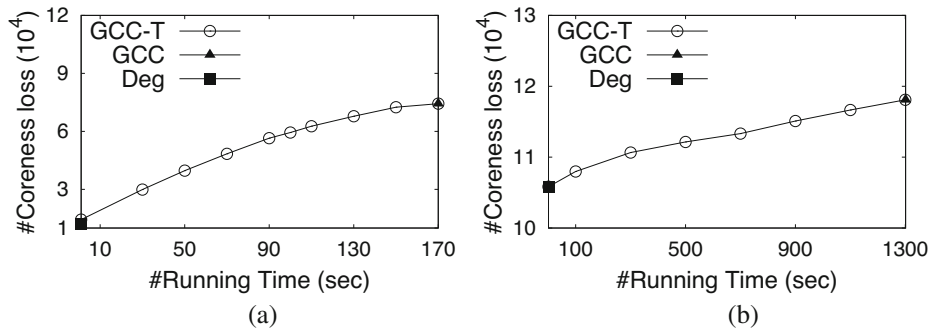


Figure 10 Performance of time-dependent GCC algorithm

of GCC-T, GCC, and Deg on facebook ( $b = 3000$ ) and Socfb-Northeastern19 ( $b = 1000$ ), we can see that the coreness loss of GCC-T fast approaches that of GCC when the algorithm costs more time. Regarding a given time limit, a relatively effective result can be retrieved by GCC-T.

**Evaluating optimization techniques** Figure 11a shows the total running time of GCC-O-P-B, GCC-P-B, GCC-B, and GCC on all the 12 datasets when  $b = 100$ . The results show that GCC-P-B is much faster than GCC-O-P-B by applying core maintenance as introduced in Section 4.4, and GCC is faster than GCC-P-B and GCC-B by at least 2 times because of the optimization by Theorem 5 and Theorem 6. Figure 11b and c study the impact  $b$  on four algorithms DBLP and Stanford. We can see that GCC is scalable with the growth of  $b$  and it is always faster than the other algorithms.

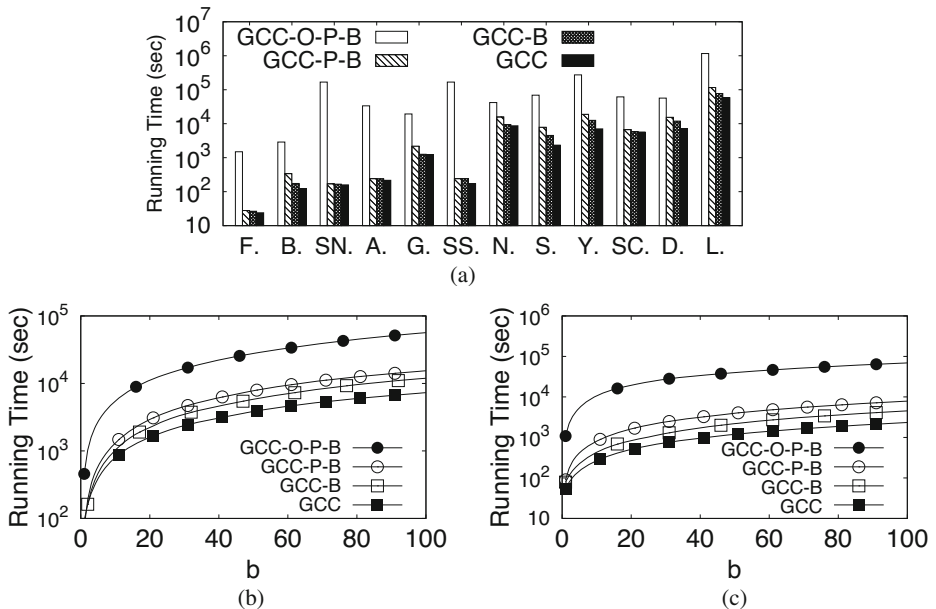


Figure 11 Time cost of different algorithms

## 6 Related work

Identifying cohesive subgraphs is an important problem in graph analytics. In the literature, many cohesive subgraph models have been proposed, e.g.,  $k$ -core [4, 7, 22, 30, 34, 37, 45],  $k$ -truss [15, 23, 31, 39, 41], clique [1, 8, 12],  $k$ -plex [43] and  $k$ -ECC [10, 51]. Among them,  $k$ -core is a well-studied model, due to its effectiveness in various applications including community discovery [11, 17, 19, 25], influential spreader identification [18, 24, 26, 32], network structure analysis [3, 6, 9, 13, 16], predicting structural collapse [36], and graph visualization [2, 50].

The concept of  $k$ -core and the computing algorithm are first introduced in [34, 37]. An  $O(m)$ -time in-memory algorithm for core decomposition is proposed in [4]. The core forest can also be computed in  $O(m)$  time as studied in [34]. Additionally, the core decomposition algorithms under different environments are also studied including I/O efficient core decomposition [44] and distributed core decomposition [35]. Besides, various variants of  $k$ -core are explored, e.g.,  $(\alpha, \beta)$ -core [29, 42],  $(k, r)$ -core [48],  $(k, s)$ -core [21, 46], radius-bounded  $k$ -core [40] and skyline  $k$ -core [25].

The engagement dynamics in social networks has attracted significant focus, e.g., [5, 14, 28, 33, 47]. In these works, the  $k$ -core model is widely adopted, due to its degeneration property that can be used to quantify engagement dynamics in real social networks [34]. In the view of enhancing the engagement of some users, the anchored  $k$ -core problem [5] and the anchored coreness problem [28] are studied, where the degrees of the anchored (enhanced) users are regarded as positive infinity. The collapsed  $k$ -core problem [47] considers a different view where some key users should be protected against the attacks from the competing networks. Thus, it investigates the effect of user departure while the anchor problems focus on user enhancement. However, the collapsed  $k$ -core problem only considers the key users regarding a given  $k$ -core. The collapsed coreness problem studied in this paper finds the key users regarding the stability of the whole network.

## 7 Conclusion and future work

The coreness sum of all the vertices in a network is regarded as a major indicator for the stability of the network. In this paper, we propose and study the collapsed coreness problem to find the key users for network structural stability. Given a graph  $G$  and a budget  $b$ , the problem aims to find  $b$  vertices in  $G$  such that the deletion of the  $b$  vertices leads to the largest coreness loss on all the vertices. We prove the problem is NP-hard and APX-hard unless  $P=NP$ . An efficient greedy heuristic is proposed with effective optimization techniques. The algorithm is also extended to return a set of key users within a given time limit. Extensive experiments on 12 real-life graphs verify that the collapsed coreness model is effective and the proposed algorithm is efficient.

The proposed GCC algorithm may inspire efficient parallel and distributed solutions, as the data locality and independence derived from the heuristic can ensure a sound strategy for task allocation. We may assign the candidate vertices according to the sub-trees containing the vertices in the forest, restrict the computation in each computing node by carefully partitioning the graph with tree nodes, and thus reduce the communication cost between different computing nodes. Besides, it is also interesting to explore a faster variant of the greedy heuristic with acceptable sacrifice on result quality.

**Acknowledgements** This work is supported by National Key Research and Development Plan (Grant No. 2018YFB1800701), National Natural Science Foundation of China (Grant No. 62002073), Guangzhou Basic and Applied Basic Research Foundation (Grant No. 202102020675), Guangdong Higher Education Innovation Group 2020KCXTD007, Guangdong Province Key Research and Development Plan (Grant No. 2019B010137004), and Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2019).

**Funding** The grants declared in Acknowledgements were received to assist with the preparation of this manuscript.

**Availability of data and material** All the datasets used in this paper is public and every one can access to it.

**Code Availability** All the source codes will be shared online after the manuscript is accepted.

## Declarations

**Conflict of Interests** The authors have no relevant financial or non-financial interests to disclose.

## References

1. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection. In: LATIN, pp. 598–612 (2002)
2. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: Large scale networks fingerprinting and visualization using the k-core decomposition. In: NeurIPS, pp. 41–50 (2005)
3. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *Networks Heterog. Media* **3**(2), 371–393 (2008). <https://doi.org/10.3934/nhm.2008.3.371>
4. Batagelj, V., Zaversnik, M.: An  $o(m)$  algorithm for cores decomposition of networks CoRR cs.DS/0310049 (2003)
5. Bhawalkar, K., Kleinberg, J.M., Lewi, K., Roughgarden, T., Sharma, A.: Preventing unraveling in social networks: The anchored k-core problem. *SIAM J. Discrete Math.* **29**(3), 1452–1475 (2015)
6. Bola, M., Sabel, B.A.: Dynamic reorganization of brain functional networks during cognition. *Neuroimage* **114**, 398–413 (2015)
7. Bonchi, F., Khan, A., Severini, L.: Distance-generalized core decomposition. In: SIGMOD, pp. 1006–1023 (2019)
8. Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM* **16**(9), 575–576 (1973)
9. Carmi, S., Havlin, S., Kirkpatrick, S., Shavitt, Y., Shir, E.: A model of internet topology using k-shell decomposition. *Proc. Natl. Acad. Sci.* **104**(27), 11150–11154 (2007)
10. Chang, L., Yu, J.X., Qin, L., Lin, X., Liu, C., Liang, W.: Efficiently computing k-edge connected components via graph decomposition. In: SIGMOD, pp. 205–216 (2013)
11. Chen, L., Liu, C., Liao, K., Li, J., Zhou, R.: Contextual community search over large social networks. In: ICDE, pp. 88–99. IEEE (2019). <https://doi.org/10.1109/ICDE.2019.00017>
12. Cheng, J., Ke, Y., Fu, A.W., Yu, J.X., Zhu, L.: Finding maximal cliques in massive networks by  $h^*$ -graph. In: SIGMOD, pp. 447–458 (2010)
13. Chu, D., Zhang, F., Lin, X., Zhang, W., Zhang, Y., Xia, Y., Zhang, C.: Finding the best k in core decomposition: A time and space optimal solution. In: ICDE, pp. 685–696 (2020). <https://doi.org/10.1109/ICDE48307.2020.00065>
14. Chwe, M.S.Y.: Communication and coordination in social networks. *Rev. Econ. Stud.* **67**(1), 1–16 (2000)
15. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. National Security Agency Technical Report, p. 16 (2008)
16. Daianu, M., Jahanshad, N., Nir, T.M., Toga, A.W., Jack Jr., C.R., Weiner, M.W., Thompson, P.M.: Breakdown of brain connectivity between normal aging and alzheimer's disease: A structural k-core network analysis. *Brain Connect.* **3**(4), 407–422 (2013). <https://doi.org/10.1089/brain.2012.0137>

17. Dourisboure, Y., Geraci, F., Pellegrini, M.: Extraction and classification of dense implicit communities in the web graph. *TWEB* **3**(2), 7:1–7:36 (2009)
18. Elsharkawy, S., Hassan, G., Nabhan, T., Roushdy, M.: Effectiveness of the k-core nodes as seeds for influence maximisation in dynamic cascades. *Int. J. Comput.*, p. 2 (2017)
19. Fang, Y., Cheng, R., Li, X., Luo, S., Hu, J.: Effective community search over large spatial graphs. *PVLDB* **10**(6), 709–720 (2017)
20. García, D., Mavrodiev, P., Schweitzer, F.: Social resilience in online communities: the autopsy of friendster. In: *Conference on Online Social Networks*, pp. 39–50 (2013)
21. Ghafouri, M., Wang, K., Zhang, F., Zhang, Y., Lin, X.: Efficient graph hierarchical decomposition with user engagement and tie strength. In: *DASFAA*, pp. 448–465 (2020)
22. Giatsidis, C., Malliaros, F.D., Thilikos, D.M., Vazirgiannis, M.: Corecluster: A degeneracy based graph clustering framework. In: *AAAI*, pp. 44–50 (2014)
23. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: *SIGMOD*, pp. 1311–1322 (2014)
24. Kitsak, M., Gallos, L.K., Havlin, S., Liljeros, F., Muchnik, L., Stanley, H.E., Makse, H.A.: Identification of influential spreaders in complex networks. *Nat. Phys.* **6**(11), 888 (2010)
25. Li, R., Qin, L., Ye, F., Yu, J.X., Xiao, X., Xiao, N., Zheng, Z.: Skyline community search in multi-valued networks. In: *SIGMOD*, pp. 457–472 (2018)
26. Lin, J.H., Guo, Q., Dong, W.Z., Tang, L.Y., Liu, J.G.: Identifying the node spreading influence with largest k-core values. *Phys. Lett. A* **378**(45), 3279–3284 (2014)
27. Lin, Z., Zhang, F., Lin, X., Zhang, W., Tian, Z.: Hierarchical core maintenance on large dynamic graphs, vol. 14, pp. 757–770 (2021). <https://doi.org/10.14778/3446095.3446099>
28. Linghu, Q., Zhang, F., Lin, X., Zhang, W., Zhang, Y.: Global reinforcement of social networks: The anchored coreness problem. In: Maier, D., Pottinger, R., Doan, A., Tan, W., Alawini, A., Ngo, H.Q. (eds.) *SIGMOD*, pp. 2211–2226. ACM (2020). <https://doi.org/10.1145/3318464.3389744>
29. Liu, B., Yuan, L., Lin, X., Qin, L., Zhang, W., Zhou, J.: Efficient ( $\alpha, \beta$ )-core computation: an index-based approach. In: Liu, L., White, R.W., Mantrach, A., Silvestri, F., McAuley, J.J., Baeza-Yates, R., Zia, L. (eds.) *WWW*, pp. 1130–1141. ACM (2019). <https://doi.org/10.1145/3308558.3313522>
30. Liu, B., Zhang, F., Zhang, C., Zhang, W., Lin, X.: Corecube: Core decomposition in multilayer graphs. In: *WISE*, pp. 694–710 (2019). [https://doi.org/10.1007/978-3-030-34223-4\\_44](https://doi.org/10.1007/978-3-030-34223-4_44)
31. Liu, B., Zhang, F., Zhang, W., Lin, X., Zhang, Y.: Efficient community search with size constraint. In: *ICDE* (2021)
32. Malliaros, F.D., Rossi, M.E.G., Vazirgiannis, M.: Locating influential nodes in complex networks. *Scientific Reports* **6**, 19307 (2016)
33. Malliaros, F.D., Vazirgiannis, M.: To stay or not to stay: modeling engagement dynamics in social graphs. In: *CIKM*, pp. 469–478 (2013)
34. Matula, D.W., Beck, L.L.: Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM* **30**(3), 417–427 (1983)
35. Montresor, A., Pellegrini, F.D., Miorandi, D.: Distributed k-core decomposition. *IEEE Trans. Parallel Distrib. Syst.* **24**(2), 288–300 (2013)
36. Morone, F., Del Ferraro, G., Makse, H.A.: The k-core as a predictor of structural collapse in mutualistic ecosystems. *Nat. Phys.* **15**(1), 95 (2019)
37. Seidman, S.B.: Network structure and minimum degree. *Social Networks* **5**(3), 269–287 (1983)
38. Seki, K., Nakamura, M.: The mechanism of collapse of the friendster network: What can we learn from the core structure of friendster? *Social Netw. Analys. Mining* **7**(1), 10:1–10:21 (2017)
39. Wang, J., Cheng, J.: Truss decomposition in massive networks. *PVLDB* **5**(9), 812–823 (2012)
40. Wang, K., Cao, X., Lin, X., Zhang, W., Qin, L.: Efficient computing of radius-bounded k-cores. In: *ICDE*, pp. 233–244 (2018)
41. Wang, K., Lin, X., Qin, L., Zhang, W., Zhang, Y.: Efficient bitruss decomposition for large-scale bipartite graphs. In: *ICDE*, pp. 661–672 (2020)
42. Wang, K., Zhang, W., Lin, X., Zhang, Y., Qin, L., Zhang, Y.: Efficient and effective community search on large-scale bipartite graphs. In: *ICDE*. IEEE (2021)
43. Wang, Y., Jian, X., Yang, Z., Li, J.: Query optimal k-plex based community in graphs. *Data Sci. Eng.* **2**(4), 257–273 (2017). <https://doi.org/10.1007/s41019-017-0051-3>
44. Wen, D., Qin, L., Zhang, Y., Lin, X., Yu, J.X.: I/O efficient core graph decomposition at web scale. In: *ICDE*, pp. 133–144 (2016)
45. Zhang, C., Zhang, F., Zhang, W., Liu, B., Zhang, Y., Qin, L., Lin, X.: Exploring finer granularity within the cores: Efficient (k, p)-core computation. In: *ICDE*, pp. 181–192 (2020). <https://doi.org/10.1109/ICDE48307.2020.00023>

46. Zhang, F., Yuan, L., Zhang, Y., Qin, L., Lin, X., Zhou, A.: Discovering strong communities with user engagement and tie strength. In: DASFAA, pp. 425–441. Springer (2018)
47. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: Finding critical users for social network engagement: The collapsed k-core problem. In: AAAI, pp. 245–251 (2017)
48. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: When engagement meets similarity: Efficient (k, r)-core computation on social networks. *PVLDB* **10**(10), 998–1009 (2017)
49. Zhang, Y., Yu, J.X., Zhang, Y., Qin, L.: A fast order-based approach for core maintenance. In: ICDE, pp. 337–348 (2017). <https://doi.org/10.1109/ICDE.2017.93>
50. Zhao, F., Tung, A.K.H.: Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB* **6**(2), 85–96 (2012)
51. Zhou, R., Liu, C., Yu, J.X., Liang, W., Chen, B., Li, J.: Finding maximal k-edge-connected subgraphs from a large graph. In: EDBT, pp. 480–491 (2012)

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