Finding Critical Users in Social Communities: The Collapsed Core and Truss Problems

Fan Zhang, Conggai Li, Ying Zhang, Lu Qin, and Wenjie Zhang

Abstract—In social networks, the leave of critical users may significantly break network engagement, i.e., lead a large number of other users to drop out. A popular model to measure social network engagement is k-core, the maximal subgraph in which every vertex has at least k neighbors. To identify critical users, we propose the collapsed k-core problem: given a graph G, a positive integer k and a budget b, we aim to find b vertices in G such that the deletion of the b vertices leads to the smallest k-core. We prove the problem is NP-hard and inapproximate. An efficient algorithm is proposed, which significantly reduces the number of candidate vertices. We also study the user leave towards the model of k-truss which further considers tie strength by conducting additional computation w.r.t. k-core. We prove the corresponding collapsed k-truss problem is also NP-hard and inapproximate. An efficient algorithm is proposed to solve the problem. The advantages and disadvantages of the two proposed models are experimentally compared. Comprehensive experiments on nine real-life social networks demonstrate the effectiveness and efficiency of our proposed methods.

Index Terms—Cohesive subgraph, user engagement, tie strength, k-core, k-truss

1 Introduction

User engagement on social network has attracted significant interests over recent years [1], [2], [3]. k-core is a simple and popular model based on degree constraint, which has been widely used to measure the network engagement [4], [5], [6]. Assuming all users in a community/group are initially engaged, each individual has two strategies, to remain engaged or drop out. Particularly, a user will remain engaged if and only if at least k of his/her friends are engaged (i.e., degree constraint). A user with less than k friends engaged will drop out, and his/her leave may be contagious and forms a cascade of the departure (i.e., collapse) in the network. When the collapse stops, the remaining engaged users corresponds to the well-known concept k-core, the maximal induced subgraph in which every vertex has at least k neighbors. The large number of engaged friends for a user can ensure active engagement of the user [3]. The size of k-core can be used to measure the overall engagement of the social network [1].

A natural question is that, given a limited budget b, how to find b vertices (i.e., users) in a network so that we can get the smallest k-core by removing these b vertices. This problem is named the collapsed k-core problem (CCP) in this paper, which aims to break user engagement with the greatest extent for a given budget b. By developing an efficient and scalable solution for this problem, we can quickly identify critical users whose leave will destroy the k-core communities and collapse the network most severely. These users are critical for the overall engagement of k-core communities and the network. For instance, we can find most valuable users, to sustain or destroy the engagement. We can also evaluate the robustness of network engagement or k-core communities against the vertex attack.

Example 1. Suppose there is a study group, and the number of friends in the group reflects the willingness of engagement for each member (i.e., user). If one drops out, he/she will weaken the willingness of his/her friends to remain engaged, which may incur the collapse of the group. As illustrated in Fig. 1, we model 17 members in a study group and their relationship as a network. According to the above engagement model with k=3, i.e., a person will drop out if there are less than 3 friends, 15 members will remain engaged; that is, 3-core of the network is the whole network excluding u_1 and u_{12}. Clearly, if users in 3-core drop out regardless the number of friends, e.g., attracted by another group, the network will further collapse. The extent of the collapse varies among different users. For instance, although u_9 has 6 friends in 3-core, the departure of u_9 will not further lead to the leave of other users because each of his/her neighbors still has 3 friends engaged. On the contrary, the leave of u_{11} will lead to the leave of 7 members in the group including u_2, u_5, u_6, u_7, u_{13}, u_{16}, and u_{17}. In this sense, it is more cost-effective to give u_{11} the incentive (e.g., bonus) to ensure his/her engagement or persuade him/her to leave the group.

The leave of users not only brings down user engagement level, but also weakens the strength of user ties inside the communities. Tie strength is often considered in social network study, as a fundamental and important network characteristic [7], [8], [9]. The model of k-truss is proposed as essentially an enhanced version of k-core by further

- *F. Zhang and W. Zhang are with the School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia. E-mail: fan.zhang3@unsw.edu.au, zhangw@cse.unsw.edu.au.*
- *C. Li, Y. Zhang, and L. Qin are with the Centre for Artificial Intelligence, University of Technology Sydney, Sydney, NSW 2007, Australia. E-mail: conggai.li@student.uts.edu.au, {ying.zhang, lu.qin}@uts.edu.au.*

Manuscript received 25 Apr. 2018; revised 3 Oct. 2018; accepted 1 Nov. 2018. Date of publication 12 Nov. 2018; date of current version 5 Dec. 2019. (Corresponding author: Fan Zhang.) Recommended for acceptance by J. Xu. Digital Object Identifier no. 10.1109/TKDE.2018.2880976

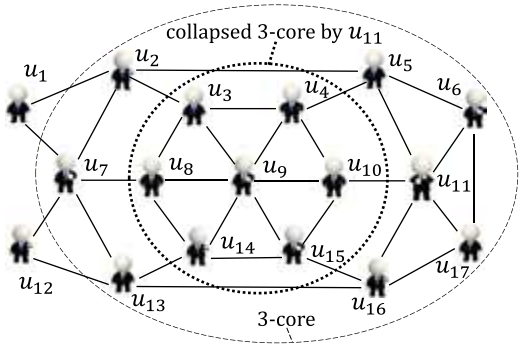


Fig. 1. Motivating example.

computing and requiring the strength of each tie (i.e., edge) [10], [11]. Many recent studies show that k-truss captures users with high engagement and strong interactions [12], [13]. The k-truss requires each edge inside to be contained in at least $k-2$ triangles in the k-truss. Thus each k-truss edge is a relatively strong tie. The requirement also deduces that each k-truss vertex has at least $k-1$ neighbors in the k-truss. It is interesting and promising to study the user leave in k-truss, which examines whether the computation cost regarding k-truss is worthwhile and affordable compared with k-core. Consequently, we also study the collapsed k-truss problem (CTP), which is to find b vertices whose deletion can lead to the smallest k-truss in the network. This problem helps to discover critical k-truss users whose leave can extremely break the k-truss communities.

In addition to social networks, cohesive subgraph models can discover some tight-knit elements in many other complex networks including the Internet, the World Wide Web, cellular networks and brain networks [14], [15], [16], [17]. For instance, in protein interaction networks, k-core subgraphs are shown to represent cohesive groups of proteins with same functionalities [18], [19]. Thus, the proposed problems may also help us find some important elements in these networks.

*Challenges.*To the best of our knowledge, we are the first to propose and investigate the CCP and CTP. We prove both of the problems are NP-hard for any k value. We also prove the CCP and CTP are inapproximate within a factor of $1-1/e$ for $k \geq 3$ and $k \geq 4$, respectively.

A basic exact solution requires to enumerate all possible answer sets with size b. Towards a possible answer set A, we have to conduct the complete k-core computation with the deletion of A to find the size of collapsed k-core. Due to the cascade nature in k-core computation, it is unpromising to estimate the size of collapsed k-core without the complete k-core computation. The above observations also hold for k-truss in the CTP. Although the k-core and k-truss computations can be done in polynomial time with efficient algorithms, the large number of candidate answer sets makes the exact solutions unaffordable. Then we aim to optimize the heuristical solutions for both of the problems, where reducing the candidate number is critical and challenging. Although the two proposed problems share the idea of vertex remove, the computations of k-core and k-truss are inherently different where the former is based on vertex deletions and the latter is based on edge deletions with update of triangles. The capture of triangles in k-truss leads to a larger computation cost while also a more cohesive structure in k-truss than k-core.

Contributions. Our principal contributions are as follows.

- We propose and investigate the CCP and CTP to find critical users according to the well-studied models k-core and k-truss, respectively.
- We prove both of the problems are NP-hard for any k value. We also prove the problems are inapproximate within a factor of $1-1/e$.
- We develop efficient heuristic algorithms, named CKC and CKT, to solve the problems. The proposed pruning techniques significantly eliminate the unpromising candidate vertices.
- We experimentally compare the CKC and CKT models. Our comprehensive experiments on 9 real-life networks demonstrate the efficiency of our algorithms and effectiveness of our models.

Outline. Section 2 reviews the related work. Section 3 presents the preliminaries for CCP. Section 4 analyzes the complexity of CCP. Section 5 proposes the CKC algorithm. Section 6 presents the preliminaries for CTP. Section 7 analyzes the complexity of CTP. Section 8 proposes the CKT algorithm. Section 9 evaluates the models and the algorithms. Section 10 concludes the paper.

2 Related Work

There are various cohesive subgraph models to accommodate specific scenarios in the literature, such as clique [20], k-plex [21], k-core [22] and k-truss [23], to name a few. Among these subgraph models, k-core and k-truss are the widely studied models with polynomial computation time.

Seidman [22] introduces the k-core model which benefits many important problems in recent years with a wide spectrum of applications such as social contagion [24], network analysis [25], network visualization [26], event detection [27], protein function prediction [18] and so on. There are multiple studies for core number computation under different settings including a linear-time in-memory algorithm [28], I/O efficient algorithms [29], [30], locally computing and estimating [31] and core number maintenance on dynamic graphs [32], [33].

The engagement dynamic in social networks has attracted significant studies, e.g., [1], [3], [34]. The k-core becomes popular in social studies, because its degeneration property can be used to quantify engagement dynamics in real social networks [1]. Luo et al. [35] investigate the parameterized complexity of collapsed k-core problem. Bhawalkar et al. [3] propose the problem of anchored k-core to prevent unraveling of social networks. This problem is to find b vertices outside k-core such that the existence (anchor) of the b vertices leads to the largest k-core. Zhang et al. [36] propose an efficient heuristic algorithm to solve this problem on general graphs. In the anchored k-core problem, we need to consider the vertices *not* in k-core because it is useless to anchor vertices already in k-core. This is different from the collapsed k-core problem, where the deletion of a vertex *not* in k-core will not affect the resulting k-core.

Tie strength, introduced by [7], is a fundamental and important social network characteristic. However, the k-core treats each tie equally, which cannot guarantee the strength of ties inside to be strong. Besides, the k-core is considered as "seedbeds, within which cohesive subsets can precipitate out" [22]. Further considering the tie strength and its

dynamic in communities, Cohen [23] introduces the model of k-truss where each edge is contained in at least $k-2$ triangles in the k-truss. Rotabi et al. [8] shows the existing methods for strong tie detection are usually based on structural information, especially on triangles. The definition of k-truss also deduces that each k-truss vertex has at least $k-1$ neighbors in the k-truss. For the fact that the k-truss model not only ensures the strong tie strength among users but also captures users with high engagement inside the community, the k-truss is considered as an enhanced version of k-core.

There are many studies on the model of k-truss while none of them utilizes k-truss to find critical users. Wang and Cheng [37] show the time complexity of truss decomposition is $O(m^{1.5})$ and propose an I/O efficient algorithm. Some works study the truss-based community search on large and dynamic graphs [12], [38]. Huang and Lakshmanan [13] search the attributed k-truss community with the largest attribute relevance score.

3 Preliminaries of CCP

We consider an unweighted and undirected graph $G=(V,E)$, where V (resp. E) represents the set of vertices (resp. edges) in G. We use n (resp. m) to denote the number of vertices (resp. edges) in the graph G and we assume $m > n$. Given a subgraph S of G, we denote the adjacent vertex set (i.e., neighbor set) of u in S by $N(u,S)$. We use $G \setminus S$ to represent the subgraph which removes S from G. When the context is clear, we may eliminate the second parameter in notations, such as using $deg(u)$ instead of $deg(u,G)$. The notations are summarized in Table 1.

Definition 1. ***k-core****. Given a graph G and a positive integer k, a subgraph S is the k-core of G, denoted by $C_k(G)$, if (i) S satisfies* degree constraint*, i.e., $deg(u,S) \geq k$ for every $u \in S$; and (ii) S is maximal, i.e., any subgraph $S' \supset S$ cannot be a k-core.*[1]

Algorithm 1. ComputeCore(G,k)

Input: G : a social network, k : degree constraint
Output: $C_k(G)$: the k-core of G
1 **while** exists $u \in G$ with $deg(u,G) < k$ **do**
2 $G \leftarrow G \setminus \{u \cup E(u,G)\}$;
3 **return** G

As shown in Algorithm 1, the k-core of a graph G can be obtained by recursively removing the vertices whose degrees are less than k, with the time complexity of $\mathcal{O}(m)$ [28].

In this paper, once a vertex u in G is *collapsed*, u and its incident edges are always removed from k-core regardless of the degree constraint. We may use *collapsers* to represent the collapsed vertices.

Definition 2. ***collapsed k-core****. Given a graph G and a set $A \subseteq G$ of vertices, the collapsed k-core, denoted by $C_k(G_A)$, is the corresponding k-core of G with vertices in A removed.*

In addition to the deletion of the collapsed vertices in A, more vertices in $C_k(G)$ might be deleted as well due to the contagious nature of the k-core computation. These vertices are called *followers* of the collapsed vertices A, denoted by $\mathcal{F}(A,G)$, because they will remain in k-core if the vertices in A are not deleted. Formally, $\mathcal{F}(A,G)$ = the vertices in $C_k(G) \setminus \{C_k(G_A) \cup A\}$. The size of the followers reflects the effectiveness of the collapsed vertices.

1. In real-life applications, the value of k is determined by users based on their requirement for cohesiveness, or learned according to ground-truth communities in the network.

TABLE 1
Summary of Notations

Notation	Definition
G	an unweighted and undirected graph
S	a subgraph of G
u, v, x; e	a vertex in G; an edge in G
(u,v), $e(u,v)$	the edge between u and v
n, m	the number of vertices and edges in G
$deg(u,S)$	the number of adjacent vertices of u in S
$N(u,S)$	the adjacent vertices of u in S
$V(S)$	the vertex set of S
$E(u,G)$	the edge set where each edge is incident to u and each edge is in G
$E(S,G)$	the union set of $E(u,G)$ for each $u \in S$
k	a positive integer
b	the budget for the number of collapsers
A	a set of collapsers vertices
G_A	the graph G with A removed
$C_k(G)$	the k-core of G
$\|C_k\|$	the number of vertices in $C_k(G)$
$C_k(G_A)$	the k-core of the graph G_A
$\mathcal{F}(A)$	the followers of A in the collapsed k-core, i.e., the vertices in $C_k(G) \setminus \{C_k(G_A) \cup A\}$

Problem Statement.Given a graph G, a degree constraint k and a budget b, the *collapsed k-core problem (CCP)* aims to find a set A of b collapsed vertices in G so that the number of followers, $\mathcal{F}(A,G)$, is maximized.

Example 2. In Fig. 1, if we set $k=3$ and $b=1$, the result of the collapsed k-core problem can be $A=\{u_{11}\}$ with $C_k(G_A)=\{u_3,u_4,u_8,u_9,u_{10},u_{14},u_{15}\}$ and $\mathcal{F}(A,G)=\{u_2, u_5,u_6,u_7,u_{13},u_{16},u_{17}\}$.

4 Complexity of CCP

In this section, we present the complexity results of CCP.

Theorem 1. *The collapsed k-core problem is NP-hard for any k.*

Proof. (1) When $k=1$, we reduce the collapsed k-core problem from the maximum independent set problem [39]. To delete a vertex from 1-core during the collapsed 1-core computation, we have to remove all its adjacent vertices, i.e., make the vertex independent. Consequently, the problem of finding the maximum independent set S in a graph G is equivalent to finding the set of vertices $G \setminus S$ such that $G \setminus S$ is minimum and collapsing them can lead to an empty 1-core. Note that we need to try at most $n-1$ times ($1 \leq b < n$) to find the minimum $G \setminus S$. Thus, we have the collapsed k-core problem is NP-hard when $k=1$.

(2) When $k=2$, we reduce the collapsed 2-core problem from the case of $k=1$, which has been proved to be NP-hard. Given any graph G_1 with n vertices and m edges, we construct another graph G_2 with $n+2m$ vertices and $4m$ edges as follows. For each edge $e(v_1,v_2)$ in G_1, we add two virtual vertices w and w' and construct the following four edges in G_2: $e(v_1,w)$, $e(w,v_2)$, $e(v_1,w')$ and $e(w',v_2)$, as shown in Fig. 2a. An example of graph

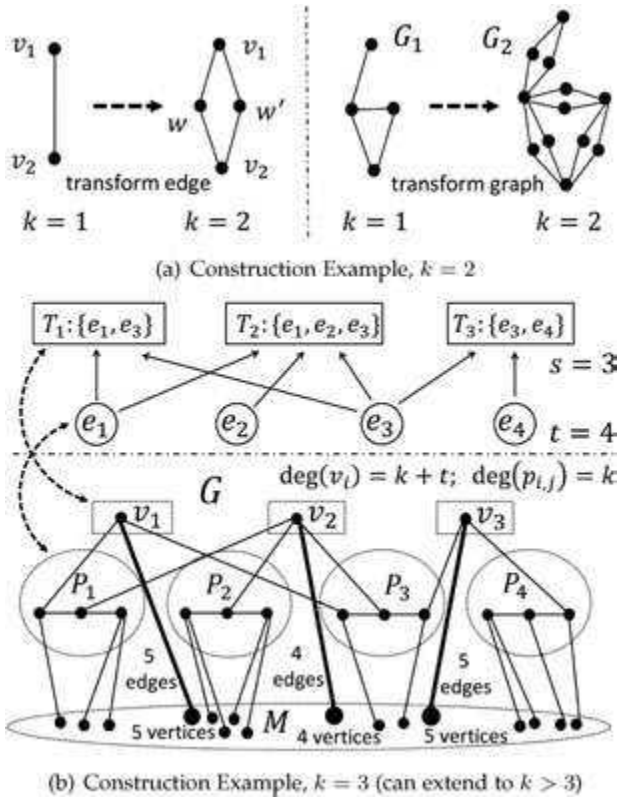


Fig. 2. Examples for proving NP-hardness and inapproximability.

construction is also illustrated in Fig. 2a. We do not need to include any virtual vertices in the optimal solution of collapsed 2-core because the influence of deleting a virtual vertex can always be covered by deleting one of its two neighbor vertices (non-virtual vertices). Therefore, the deletion of each edge in G_1 during the computation is always mapped to the deletion of four corresponding edges in G_2. Then the optimal solution of collapsed 2-core on G_2 is also that of collapsed 1-core on G_1. As a result, the collapsed k-core problem is NP-hard when $k = 2$.

(3) When $k \geq 3$, we reduce the collapsed k-core problem from the maximum coverage problem [40]; that is finding at most b sets to cover the largest number of elements, where b is a given budget. First, we consider an arbitrary instance of maximum coverage problem with s sets $T_1, .., T_s$ and t elements $\{e_1, .., e_t\} = \cup_{1 \leq i \leq s} T_i$. Then we construct a corresponding instance of the collapsed k-core problem in a graph G as follows.

The set of vertices in G consists of three parts: M, V, and P. M consists of $(t+s)^4$ vertices in which every pair of vertices in M are adjacent. V consists of s vertices, $v_1, v_2, \ldots, v_s$, where vertex v_i corresponds to the set T_i for any $1 \leq i \leq s$. For each vertex v_i $(1 \leq i \leq s)$, we add $k + t - |T_i|$ edges from v_i to $k + t - |T_i|$ unique vertices in M. Here, by unique, we mean that each vertex in M can be used at most once when adding edges to vertices outside M. P consists of t parts $P_1, P_2, \ldots, P_t$, where each part P_i $(1 \leq i \leq t)$ corresponds to the element e_i and P_i consists of s vertices $p_{i,1}$, $p_{i,2}, \ldots, p_{i,s}$. For each P_i $(1 \leq i \leq t)$ we first add $s - 1$ edges, that is, for each $1 \leq j < s$, we add an edge from $p_{i,j}$ to $p_{i,j+1}$. For each set T_i $(1 \leq i \leq s)$ and each element e_j $(1 \leq j \leq t)$, if $e_j \in T_i$, we add an edge $e(v_i, p_{j,i})$ in G. At this stage, the degree of each vertex in P is at most 3. Next, we add edges from vertices in P to unique vertices in M to guarantee that the degree of each vertex in P is exactly k. This can be done since $k \geq 3$. Then the construction of G is completed. Clearly, G is a k-core. Fig. 2b shows an example of the graph G with $k = 3$ constructed from 3 sets and 4 elements.

The key idea is that we ensure that: (i) only vertices in V need to be considered as collapsed vertices, since any vertex in M or P cannot have more followers than a vertex in V; (ii) none of the vertices in M will be deleted during the computation; (iii) all P_i have the same size for $1 \leq i \leq t$; and (iv) when a vertex v_i $(1 \leq i \leq s)$ is removed, for each part P_j $(1 \leq j \leq t)$ connected with v_i (i.e., $e_j \in T_i$), all vertices in P_j will be deleted due to degree constraint. By doing this, the optimal solution of the collapsed k-core problem corresponds to optimal solution of the maximum coverage problem. Since the maximum coverage problem is NP-hard, we prove that the collapsed k-core problem is NP-hard for any $k \geq 3$. □

We also prove the inapproximability of the collapsed k-core problem.

Theorem 2. *For $k \geq 3$ and any $\epsilon > 0$, the collapsed k-core problem cannot be approximated in polynomial time within a ratio of $(1 - 1/e + \epsilon)$, unless $P = NP$.*

Proof. We have reduced the collapsed k-core problem from the maximum coverage (MC) problem in the proof of Theorem 1. Here we show this reduction can also prove the inapproximability of CCP. For any $\epsilon > 0$, the MC problem cannot be approximated in polynomial time within a ratio of $(1 - 1/e + \epsilon)$, unless $P = NP$ [41]. According to the previous reduction, every solution of the collapsed k-core problem in the instance graph G corresponds to a solution of the MC problem, where the follower number for CCP is s times the element number for the MC Problem, where s is the number of sets. Let $\gamma > 1 - 1/e$, if there is a solution with γ-approximation on optimal follower number for CCP, there will be a γ-approximate solution on element number for the MC problem. So it is NP-hard to approximate collapsed k-core problem within a ratio of $(1 - 1/e + \epsilon)$ when $k \geq 3$. □

If the follower function in CCP is submodular, a greedy algorithm can still have $(1 - 1/e)$-approximation on follower number. Unfortunately, Theorem 3 gives a negative result.

Theorem 3. *Let $f(A) = |\mathcal{F}(A) \cup \{A\}|$. We have f is monotone but not submodular for any k.*

Proof. Suppose there is a set $A' \supseteq A$. For every vertex u in $\mathcal{F}(A)$, u will still be deleted in the collapsed k-core with the collapsers set A', because removing vertices in $A' \setminus A$ cannot increase the degree of u. Thus $f(A') \geq f(A)$ and f is monotone. For two arbitrary collapsers sets A and B, if f is submodular, it must hold that $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. We show that the inequality does not hold using counterexamples. When $k = 1$, we use the example shown in Fig. 3a. Suppose $k = 1$, $A = \{v_1\}$ and $B = \{v_2\}$, we have $\mathcal{F}(A) = \{v_1\}$, $\mathcal{F}(B) = \{v_2\}$, $\mathcal{F}(A \cup B) = \{v_1, v_2, v_3, v_4\}$ and $\mathcal{F}(A \cap B) = \emptyset$. So the inequation does not hold. When $k = 2$, we use the example shown in Fig. 3b. Here, M is a complete graph with $4 \times k$ vertices. When $k = 2$, if $A = \{v_1\}$ and $B = \{v_2\}$, we have $\mathcal{F}(A) = \{v_1\}$, $\mathcal{F}(B) = \{v_2\}$, $\mathcal{F}(A \cup B) = \{v_1, v_2, v_3, v_4\}$ and $\mathcal{F}(A \cap B) = \emptyset$. So the

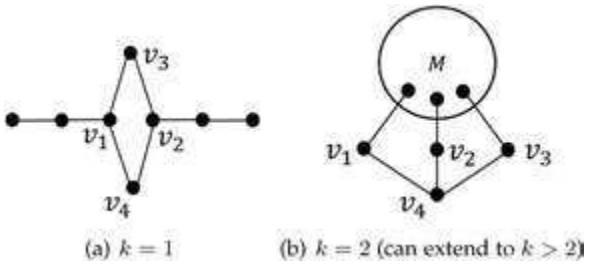


Fig. 3. Examples for non-submodularity.

inequation does not hold. When $k > 2$, we add $k-2$ edges between v_i and M, for each $1 \leq i \leq 4$. We can prove that for $A = \{v_1\}$ and $B = \{v_2\}$, the inequation is still violated. □

5 Solution of CCP

A straightforward solution of the collapsed k-core problem is to exhaustively enumerate all possible set A with size b, and compute the resulting collapsed k-core for each possible A. The time complexity of $\mathcal{O}(\binom{n}{b}m)$ is cost-prohibitive. Considering the NP-hardness and inapproximability of the problem, we resort to the greedy heuristic which iteratively finds the best collapser, i.e., the vertex with the largest number of followers. Clearly, we only need to consider the vertices in $C_k(G_A)$ since all other vertices will be deleted by degree constraint during k-core computation. Thus, a greedy algorithm is shown in Algorithm 2. In each iteration from Line 2, it finds a best collapser with most followers (Line 5) by computing the collapsed k-core of every candidate vertex at Line 3-4. The time complexity of Algorithm 2 is $\mathcal{O}(bnm)$, where n and m correspond to the number of candidate collapsers in each iteration (Line 3) and the cost of follower computation (Line 4), i.e., k-core computation.

The number of vertices in $C_k(G_A)$ at Line 3 is still considerably large, which motivates us to develop two effective pruning rules to further reduce the candidate vertices in each iteration of the greedy algorithm.

5.1 Reducing Candidate Collapsers

For presentation simplicity, in this subsection, we introduce two pruning rules to find the vertex with the largest number of followers in the first iteration of the greedy algorithm (i.e., $A = \emptyset$). They can be immediately extended to the following iterations of the greedy algorithm by using the updated $C_k(G_A)$ to replace $C_k(G)$.

Theorem 4 indicates that only the in-k-core neighbors of the vertices with degree k in k-core can have followers. Particularly, P represents the vertices with degree k in k-core, and T represents the neighbors of P in k-core.

Theorem 4. *Given a graph G and $P = \{u \mid deg(u, C_k(G)) = k\}$, if a collapsed vertex x has at least one follower, x is from T where $T = \{u \mid u \in C_k(G)$ and $N(u, G) \cap P \neq \emptyset\}$; that is $|\mathcal{F}(x, G)| > 0$ implies $x \in T$.*

Proof. We prove that a vertex $x \in G \setminus T$ cannot have any follower. (1) If $x \in G \setminus C_k(G)$, x will be deleted in k-core computation and hence $|\mathcal{F}(x)| = 0$. (2) If $x \in C_k(G) \setminus T$, x survived in k-core computation and for each x's neighbor u within $C_k(G)$, we have $deg(u, C_k(G)) > k$ since $x \notin T$. Consequently, if x is deleted, we have $deg(u, C_k(G)) \geq k$; that is, the removal of x cannot be propagated to any of its neighbors regarding degree constraint and hence other vertices. It means x does not have any follower. Since $(G \setminus C_k(G)) \cup (C_k(G) \setminus T) \cup T = G$, we have $|\mathcal{F}(x, G)| > 0$ implies $x \in T$. □

Algorithm 2. GreedyCKC(G, k, b)

Input: G : a social network, k : degree constraint, b : number of collapsers
Output: A : the set of collapsers

```
1 A ← ∅; i ← 0;
2 while i < b do
3   for each u ∈ C_k(G_A) do
4     Compute F(A ∪ u, G);
5   u* ← the best collapser in this iteration;
6   A ← A ∪ u*; i ← i + 1; update C_k(G_A);
7 return A
```

Example 3. Fig. 1 shows a graph G. When $k = 3$, the k-core $C_k(G)$ is induced by the vertices in $V(G) \setminus \{u_1, u_{12}\}$. According to Theorem 4, the set $P = \{u_2, u_6, u_7, u_{13}, u_{17}\}$ and the set $T = V(C_k(G)) \setminus \{u_4, u_9, u_{10}, u_{15}\}$. For a vertex $x \in V(G) \setminus T$, there is no follower of x in the collapsed k-core by x because it is always held that $C_k(G_x) = C_k(G) \setminus \{x \cup E(x, G)\}$. To find a best collapser, we only have to compute the collapsed k-core for each vertex in T and choose the one with most followers.

In the following theorem, we further reduce the candidate vertices by excluding vertices which have been identified as followers of other vertices.

Theorem 5. *Given two vertices x and u in graph G, we have $\{\mathcal{F}(u) \cup u\} \subseteq \{\mathcal{F}(x) \cup x\}$ if $u \in \mathcal{F}(x)$.*

Proof. $u \in \mathcal{F}(x)$ implies that u will be deleted if x is collapsed. For every vertex in $\mathcal{F}(u)$, if x is collapsed, it will also be deleted since u will be deleted and collapsing x cannot increase degrees for vertices. Thus $\mathcal{F}(u) \subset \{\mathcal{F}(x) \cup x\}$. Since $u \in \mathcal{F}(x)$ and $u \notin \mathcal{F}(u)$, we have $\{\mathcal{F}(u) \cup u\} \subseteq \{\mathcal{F}(x) \cup x\}$. □

Example 4. Fig. 1 shows a graph G. When $k = 3$, to find a best collapser in k-core, we compute the collapsed k-core for each vertex in $T = V(C_k(G)) \setminus \{u_4, u_9, u_{10}, u_{15}\}$. Suppose we compute the collapsed k-core of u_{11} first and get the follower set $\mathcal{F}(u_{11}, G) = \{u_2, u_5, u_6, u_7, u_{13}, u_{16}, u_{17}\}$, we do not need to compute the collapsed k-core of any vertex $v \in \mathcal{F}(u_{11}, G)$ to find its followers, because the follower number of v cannot be larger than $|\mathcal{F}(u_{11}, G)|$, according to Theorem 5.

By Theorem 5, in the procedure of finding a best collapser, every vertex which is a follower of a vertex can be excluded from candidate collapsers. Consequently, checking promising collapsers first, which may have large number of followers, can skip more vertices in the computation. Naturally, a vertex with more neighbors in the set P is more promising because all of its neighbors in P will follow the vertex to be deleted. Thus, to further reduce the number of candidate collapsers, we try collapsing vertices in decreasing order of their degrees in P.

5.2 CKC Algorithm

By taking advantage of two pruning rules in Theorems 4 and 5, Algorithm 3 illustrates the details of CKC algorithm which

finds the best collapser for a given graph G (i.e., $b = 1$). Particularly, we first compute the k-core of graph G (Line 1) and find the set P of vertices with degree k in $C_k(G)$ (Line 2). According to Theorem 4, we find the set T by the vertices which are inside of $C_k(G)$ and are neighbors of at least one vertex in P (Line 3). To compute $\mathcal{F}(u, G)$, we can continue the k-core computation in Line 1 with vertex u deleted (Line 5). We have the best collapser when the algorithm terminates.

In Algorithm 3, the k-core computation takes $\mathcal{O}(m)$ time (Line 1), finding the set P takes $\mathcal{O}(n)$ by scanning the degrees of k-core vertices once (Line 2) and finding the set T takes $\mathcal{O}(m)$ by visiting the neighbors of the vertices in P once (Line 3). The collapsed k-core computation takes $\mathcal{O}(m)$ for each vertex in T. Thus, the time complexity of Algorithm 3 is $\mathcal{O}(nm)$.

Algorithm 3. CKC(G, k)

Input: G : a social network, k : degree constraint,
Output: x : the best collapser
1 $C_k(G) \leftarrow$ **computeCore**(G, k);
2 $P \leftarrow \{u \mid deg(u, C_k(G)) = k\}$;
3 $T \leftarrow \{u \mid u \in C_k(G)$ and $N(u, G) \cap P \neq \emptyset\}$;
4 **for each** $u \in T$ (Theorem 4) **do**
5 Compute $\mathcal{F}(u, G)$;
6 $T \leftarrow T \setminus \mathcal{F}(u, G)$ (Theorem 5);
7 **return** the best collapser

To handle the general case with $b > 1$, our CKC algorithm can be easily fit to Algorithm 2 (replacing Line 3 and 4) to find the best collapser in each iteration. If we find a vertex $u \in \mathcal{F}(x)$ in one iteration of the greedy algorithm, x is always an equal or better candidate collapser than u in following iterations, because deleting other vertices cannot change the fact that x has equal or more followers than u (Theorem 5). Actually, we do not need to consider u as a candidate in following iterations because u will be excluded from k-core whenever x is removed. In our implementation, we order the candidates by their number of neighbors in P in each iteration to prune more candidate collapsers.

The above optimizations do not affect the time complexity of Algorithm 3. Thus, our final CKC algorithm (Algorithm 2 equipped with Algorithm 3) has a time complexity of $\mathcal{O}(bnm)$ where b is the number of resulting collapsers.

6 Preliminaries of CTP

To study the collapsed k-truss problem, we first define additional notations. A triangle is a cycle of length 3 in the graph. A e-containing triangle is a triangle which contains the edge e. We use $sup(e, S)$, the *support* of e in S, to represent the number of e-containing triangles in S. The new notations are summarized in Table 2.

Definition 3. *k-truss. Given a graph G, a subgraph S is the k-truss of G, denoted by $T_k(G)$, if (i) S satisfies* support *constraint, i.e., $sup(e, S) \geq k - 2$ for every edge $e \in S$; (ii) S is maximal, i.e., any subgraph $S' \supset S$ is not a k-truss; and (iii) S is non-trivial, i.e., no isolated vertex in S.*[2]

This definition also deduces that each k-truss vertex has at least $k - 1$ neighbors in the k-truss, because a vertex involves in at least $k - 2$ triangles in the k-truss. As shown in Algorithm 4, to find the k-truss, we first compute the $(k - 1)$-core as current graph. Then we recursively remove every edge whose support is less than $k - 2$ in current graph. We get the k-truss after removing isolated vertices. The time complexity is $\mathcal{O}(m^{1.5})$ [37].

2. The value of k is determined by user requirement for cohesiveness, or learned according to ground-truth communities.

TABLE 2
Summary of Additional Notations

Notation	Definition
(u, v, w)	a triangle contains vertices u, v and w
$sup(e, G)$	the number of e-containing triangles in G
$T_k(G)$	the k-truss of G
$T_k(G_A)$	the k-truss of $G \setminus \{A \cup E(A, G)\}$
$\mathcal{F}_t(A, G)$	the followers of A in collapsed k-truss, i.e., the vertices in $T_k(G) \setminus \{T_k(G_A) \cup A\}$
$V_\triangle(e, G)$	the set of vertices where each vertex is from a e-containing triangle in G, and each vertex is not incident to e
$V_\triangle(S, G)$	the union set of $V_\triangle(e, G)$ for every $e \in S$
$E_\triangle(u, G)$	the set of edges where each edge (v, w) belongs to a triangle (u, v, w) in G

In the collapsed k-truss model, once a vertex u in G is *collapsed*, u and its incident edges are removed from k-truss even if u is not isolated. We use *collapsers* to represent the collapsed vertices.

Definition 4. ***collapsed k-truss.*** *Given a graph G and a set $A \subseteq G$ of vertices, the collapsed k-truss, denoted by $T_k(G_A)$, is the k-truss of the subgraph $G \setminus \{A \cup E(A, G)\}$.*

Besides the deletion of the collapsers in A, there may be some other vertices in $T_k(G)$ which follow the collapsers to leave the k-truss due to the cascade of edge deletions. These vertices are called *followers* of the collapsers in A, denoted by $\mathcal{F}_t(A, G)$. Formally, $\mathcal{F}_t(A, G)$ = the vertices in $T_k(G) \setminus \{T_k(G_A) \cup A\}$. The size of the followers reflects the importance of the collapsed vertices.

Problem Statement. Given a graph G, a support constraint k and a budget b, the *collapsed k-truss problem* is to find a set A of b vertices in G such that the number of followers, $\mathcal{F}_t(A, G)$, is maximized.

Algorithm 4. ComputeTruss(G, k)

Input: G : a social network, k : support constraint
Output: $T_k(G)$: the k-truss of G
1 $G \leftarrow$ **computeCore**$(G, k - 1)$;
2 **while** exists an edge $e \in G$ with $sup(e, G) < k - 2$ **do**
3 $G \leftarrow G \setminus \{e\}$;
4 Delete isolated vertices in G;
5 **return** G

7 Complexity of CTP

Theorem 6. *The collapsed k-truss problem is NP-hard for any k.*

Proof. We reduce the collapsed k-truss problem from the minimum vertex cover (MVC) problem [42]. Given a graph G, the MVC problem is to find a minimum vertex set such that each edge in G is incident to at least one vertex of the set. When $k \geq 2$, we construct a graph G' by (1) adding a vertex set U of $k - 2$ vertices for every edge (u, v) in G; and (2) adding edges to make every $U \cup \{u, v\}$ a clique where

every two vertices are adjacent. Since every edge in the induced graph of a k-clique (a clique of k vertices) is contained in $k-2$ triangles, a k-clique forms a k-truss.

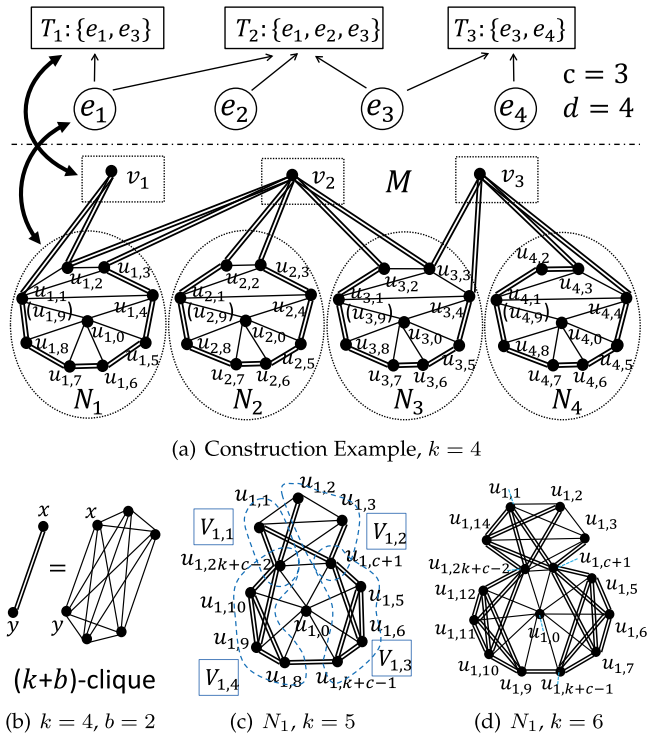
Then we prove the MVC problem on G is equivalent to finding a minimum vertex set W in G' such that the k-truss of $G' \setminus (W \cup E(W))$ is empty. Let C denote a k-clique in G' which corresponds to an edge in G. To make the k-truss empty, we have to delete at least one vertex in every C. Then we have that the minimum set W comes from G, because deleting a vertex in $G' \setminus G$ cannot destroy more C than deleting a vertex in $C \cap G$. Then every edge in G will be covered by an incident vertex in W. So the above two problems are equivalent. To find the minimum vertex set W, we can try solving the collapsed k-truss problem by at most $n-1$ times ($1 \leq b < n$). So the collapsed k-truss problem is NP-hard when $k \geq 2$. Note that when $k \leq 1$, the problem is same to the collapsed 2-truss problem according to the definitions. Then the collapsed k-truss problem is NP-hard for any k. □

Theorem 7. *For $k \geq 4$ and any $\epsilon > 0$, the collapsed k-truss problem cannot be approximated in polynomial time within a ratio of $(1-1/e+\epsilon)$, unless $P = NP$.*

Proof. We reduce the collapsed k-truss problem from the maximum coverage (MC) problem [41] to prove the inapproximability. The MC problem is to find at most b sets to cover the largest number of elements, where b is a given budget. For any $\epsilon > 0$, the MC problem cannot be approximated in polynomial time within a ratio of $(1-1/e+\epsilon)$, unless $P = NP$ [41]. Let $\gamma > 1-1/e$, we prove that if there is a solution with γ-approximation on optimal follower number for our problem, there will be a γ-approximate solution on optimal element number for MC.

We consider an arbitrary instance of MC with c sets $T_1, .., T_c$ and d elements $\{e_1, .., e_d\} = \cup_{1 \leq i \leq c} T_i$. Then we construct a corresponding instance of the collapsed k-truss problem on a graph G. Fig. 4a shows a construction example from 3 sets and 4 elements when $k = 4$. The set of vertices in G consists of two parts: M and N. M contains c vertices, i.e., $M = \cup_{1 \leq i \leq c} v_i$. N contains d sets of vertices, i.e., $N = \cup_{1 \leq j \leq d} N_j$. For every $j \in [1, d]$, N_j contains $3k+c-6$ vertices, i.e., $N_j = \cup_{0 \leq p \leq 3k+c-7} u_{j,p}$. To show the construction clearly, as the example in Fig. 4c, we divide N_j into 5 sets: $V_{j,1}$ ($k-3$ vertices), $V_{j,2}$ (c vertices), $V_{j,3}$ ($k-1$ vertices), $V_{j,4}$ ($k-1$ vertices) and $\{u_{j,0}\}$. Specifically, $V_{j,1} = u_{j,1} \cup \{u_{j,p} \mid 2k+c-2 \leq p \leq 3k+c-7\}$, $V_{j,2} = \{u_{j,p} \mid 2 \leq p \leq c+1\}$, $V_{j,3} = \{u_{j,p} \mid c+1 \leq p \leq k+c-1\}$ and $V_{j,4} = \{u_{j,p} \mid k+c \leq p \leq 2k+c-2\}$ (Note that $u_{j,1} = u_{j,2k+c-2}$ when $k = 4$).

The edge construction is as follows: (1) we define the *super edge* as a (k+b)-clique, denoted by C, where only two vertices in C have neighbors in $G \setminus C$, as the example in Figs. 4a and 4b; (2) for every set T_i ($i \in [1, c]$) and every element e_j ($j \in [1, d]$), if $e_j \in T_i$, we add two super edges $(v_i, u_{j,i})$, $(v_i, u_{j,i+1})$ and an edge (non-super) $(u_{j,i}, u_{j,i+1})$; if $e_j \notin T_i$, we add a super edge $(u_{j,i}, u_{j,i+1})$. (3) there is a super edge between every vertex pair in $V_{j,1}$; (4) for every $p \in [3, c]$, there is an edge between $u_{j,p}$ and each vertex in $V_{j,1}$; (5) we add a super edge between $u_{j,2}$ and each vertex in $V_{j,1} \setminus \{u_{j,1}\}$; (6) we add an edge $(u_{j,c+1}, u_{j,2k+c-2})$, and add a super edge between $u_{j,c+1}$ and each vertex in $V_{j,1} \setminus \{u_{j,2k+c-2}\}$; (7) we add a super edge between every vertex pair in $V_{j,3}$ except the pair $(u_{j,c+1}, u_{j,k+c-1})$; (8) we add a super edge between every vertex pair in $V_{j,4}$ except the pair $(u_{j,k+c}, u_{j,2k+c-2})$; and (9) we add an edge between $u_{j,0}$ and each vertex in $V_{j,3} \cup V_{j,4}$, and add a super edge between $u_{j,k+c-1}$ and $u_{j,k+c}$. The construction of G is completed. Figs. 4c and 4d show the construction of N_1 ($c = 3$) when $k = 5$ and $k = 6$, respectively.



(a) Construction Example, $k = 4$

(b) $k = 4, b = 2$ (c) N_1, $k = 5$ (d) N_1, $k = 6$

Fig. 4. Examples for proving inapproximability.

Here we show the support of each non-super edge in G is exactly $k-2$: (1) for a non-super edge $(u_{j,1}, u_{j,2})$, it can only form a triangle with each vertex in $V_{j,1} \setminus \{u_{j,1}\}$ ($k-4$ vertices), the vertex $u_{j,3}$ and one vertex in M; (2) for a non-super edge $(u_{j,p}, u_{j,p+1})$ ($2 \leq p \leq c$), it can only form a triangle with each vertex in $V_{j,1}$ ($k-3$ vertices) and one vertex in M; (3) for a non-super edge $(u_{j,c+1}, u_{j,2k+c-2})$, it can only form a triangle with each vertex in $V_{j,1} \setminus \{u_{j,2k+c-2}\}$ ($k-4$ vertices), the vertex $u_{j,c}$ and the vertex $u_{j,0}$; (4) for a non-super edge $(u_{j,p}, u_{j,q})$ between $V_{j,1}$ and $V_{j,2}$, it can form a triangle with each vertex in $V_{j,1} \setminus \{u_{j,p}\}$ ($k-4$ vertices), the vertex $u_{j,q-1}$ and the vertex $u_{j,q+1}$; (5) for a non-super edge $(u_{j,0}, u_{j,c+1})$, it can only form a triangle with each vertex in $V_{j,3} \setminus \{u_{j,c+1}, u_{j,k+c-1}\}$ ($k-3$ vertices) and the vertex $u_{j,2k+c-2}$. Note that it is similar for edges $(u_{j,0}, u_{j,k+c-1})$, $(u_{j,0}, u_{j,k+c})$, and $(u_{j,0}, u_{j,2k+c-2})$; and (6) for a non-super edge $(u_{j,0}, u_{j,p})(c+2 \leq p \leq k+c-2)$, it can only form a triangle with each vertex in $V_{j,3} \setminus \{u_{j,p}\}$ ($k-2$ vertices). Note than it is similar for a non-super edge in $(u_{j,0}, u_{j,p})(k+c+1 \leq p \leq 2k+c-3)$. Now the support of each non-super edge has been verified to be $k-2$. Obviously G is a k-truss.

The key idea is we ensure that: (1) only the vertices $u_{j,0}$ can become a follower of a vertex, since we need to delete at least $b+1$ vertices in a (k+b)-clique to produce followers; (2) only vertices in M need to be considered as collapsed vertices, since any vertex in N cannot have more followers than a vertex in M; (3) reducing the support for any non-super edge in N_j ($1 \leq j \leq d$) will lead to the deletion of $u_{j,0}$ due to the support constraint; and (4)

every N_j ($1 \leq j \leq d$) can only produce one follower $u_{j,0}$. Then, every solution of the collapsed k-truss problem in G corresponds to a solution of the MC problem, where the follower number for our problem equals the element number for the MC Problem. So it is NP-hard to approximate collapsed k-truss problem within a ratio of $(1 - 1/e + \epsilon)$ when $k \geq 4$. □

If the follower function in CTP is submodular, a greedy algorithm can still have $(1 - 1/e)$-approximation on follower number. Unfortunately, Theorem 8 gives a negative result.

Theorem 8. *Let $f(A) = |\mathcal{F}_t(A) \cup \{A\}|$. We have f is monotone but not submodular for any k.*

Proof. Suppose there is a set $A' \supseteq A$. For every vertex u in $\mathcal{F}_t(A)$, u will still be deleted in the collapsed k-truss with the collapser set A', because removing vertices in $A' \setminus A$ cannot increase the supports of edges. Thus $f(A') \geq f(A)$ and f is monotone. For two arbitrary collapsers sets A and B, if f is submodular, it must hold that $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. We show that the inequality does not hold using counterexamples. When $k = 1$, we consider a graph which consists of v_1, v_2, v_3, $e(v_1, v_2)$ and $e(v_2, v_3)$. Suppose $k = 1$, $A = \{v_1\}$ and $B = \{v_3\}$, we have $\mathcal{F}_t(A) = \{v_1\}$, $\mathcal{F}_t(B) = \{v_3\}$, $\mathcal{F}_t(A \cup B) = \{v_1, v_2, v_3\}$ and $\mathcal{F}_t(A \cap B) = \emptyset$. So the inequation does not hold. When $k \geq 2$, we consider a graph which consists of $2k$ vertices, i.e., $\{v_i \mid 1 \leq i \leq 2k\}$. There is an edge between every vertex pair in $\{v_i \mid 1 \leq i \leq k+1\}$, i.e., it forms a $(k+1)$-clique. The vertex set $\{v_i \mid k \leq i \leq 2k\}$ also forms a $(k+1)$-clique. Let $V_1 = \{v_i \mid 1 \leq i \leq k-1\}$ and $V_2 = \{v_i \mid k+2 \leq i \leq 2k\}$. When $k = 2$, if $A = V_1$ and $B = V_2$, we have $\mathcal{F}_t(A) = V_1$, $\mathcal{F}_t(B) = V_2$, $\mathcal{F}_t(A \cup B) = V_1 \cup V_2 \cup \{v_k, v_{k+1}\}$ and $\mathcal{F}_t(A \cap B) = \emptyset$. So the inequation is still violated. □

8 Solution of CTP

Due to the NP-hardness and inapproximability of the problem, we adopt a greedy heuristic which iteratively finds the best collapser, i.e., the vertex with the largest number of followers. We only need to consider the vertices in $T_k(G_A)$ as candidate collapsers, because all other vertices will be deleted during k-truss computation. A greedy algorithm is shown in Algorithm 5. In each iteration from Line 2, it finds a best collapser with most followers (Line 5) by computing the collapsed k-truss of every candidate vertex at Line 3-4. Because the k-truss computation at Line 4 takes $\mathcal{O}(m^{1.5})$ time, the time complexity of Algorithm 5 is $\mathcal{O}(bnm^{1.5})$, where n is the number of candidate collapsers in each iteration (Line 3) and m is the number of edges in follower computation (Line 4), i.e., k-truss computation.

Algorithm 5. GreedyCKT(G, k, b)

Input: G : a social network, k : support constraint, b : the budget for collapser number
Output: A : the set of collapsers

```
1 A ← ∅; i ← 0;
2 while i < b do
3   for each u ∈ T_k(G_A) do
4     Compute F_t(A ∪ u, G);
5   u* ← the best collapser in this iteration;
6   A ← A ∪ u*; i ← i + 1; update T_k(G_A);
7 return A
```

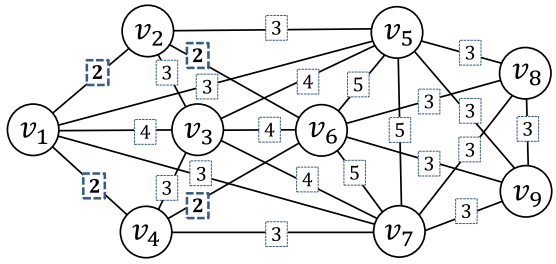


Fig. 5. Candidate reduction, $k = 4$.

The follower computation at Line 4 is efficient by conducting k-truss computation. However, the number of candidate vertices in $T_k(G_A)$ at Line 3 is still too large to afford, which motivates us to reduce the candidate vertices in the heuristic algorithm by developing effective pruning rules.

8.1 Reducing Candidates

In this section, we introduce the pruning rules on the first iteration in the greedy algorithm (i.e., $A = \emptyset$ and $i = 0$). The pruning rules can be immediately applied to the following iterations by using $T_k(G_A)$ to replace $T_k(G)$. The following theorem finds the candidate set of collapsers.

Theorem 9. *Given a graph G, let D denote the support $k-2$ edge set in $T_k(G)$, i.e., $D = \{e \mid sup(e, T_k(G)) = k-2\}$, if a collapsed vertex x has at least one follower, x is from $V_\triangle(D, T_k(G))$; that is $|\mathcal{F}_t(x, G)| > 0$ implies $x \in V_\triangle(D, T_k(G))$.*

Proof. We prove that a vertex $x \in G \setminus V_\triangle(D, T_k(G))$ cannot have any followers. (1) If $x \in G \setminus T_k(G)$, x will be deleted in k-truss computation and hence $|\mathcal{F}_t(x)| = 0$. (2) If $x \in T_k(G) \setminus V_\triangle(D, T_k(G))$, x survived in k-truss computation. Let $E_\triangle$ denote the edge set where each edge is from a triangle containing x in $T_k(G)$, and let E_x denote the edge set where each edge is incident to x in $T_k(G)$. We have that each edge in $E_\triangle \setminus E_x$ has a support of at least $k-1$, otherwise x will be in $V_\triangle(D, T_k(G))$. If we delete x in $T_k(G)$, the support of every edge in $E_\triangle \setminus E_x$ can only decrease by one, i.e., every edge in $E_\triangle \setminus E_x$ still has a support of at least $k-2$. Consequently, all the edges in $E_\triangle \setminus E_x$ will exist in $T_k(G_x)$, i.e., no cascade of deletion will incur for removing x. So only edges in E_x will be deleted by removing x, and x does not have any followers. Consequently, if x has at least one follower, $x \in V_\triangle(D, T_k(G))$. □

Example 5. In Fig. 5, when $k = 4$, the graph is already a k-truss. On each edge, we label its support, i.e., the number of triangles containing the edge. According to Theorem 9, the set $D = \{(v_1, v_2), (v_2, v_6), (v_1, v_4), (v_4, v_6)\}$. Thus the set $V_\triangle(D, T_k(G)) = \{v_3, v_5, v_7\}$ and every other vertex cannot have any followers. The best collapser is v_3 since $\mathcal{F}_t(v_3) = \{v_1, v_2, v_4\}$, $\mathcal{F}_t(v_5) = \{v_2\}$ and $\mathcal{F}_t(v_7) = \{v_4\}$.

The following theorem further reduces the candidate collapsers by excluding the vertices which have been identified as followers of other vertices.

Theorem 10. *Given a graph G, if a vertex u is a follower of x, i.e., $u \in \mathcal{F}_t(x)$, we have $\{\mathcal{F}_t(u) \cup u\} \subseteq \{\mathcal{F}_t(x) \cup x\}$.*

Proof. Since $u \in \mathcal{F}_t(x)$, u will be deleted if x is collapsed. For every vertex v in $\mathcal{F}_t(u)$, v will be deleted if x is collapsed, because u will be deleted and collapsing x cannot increase

the supports for edges. So $\mathcal{F}_t(u) \subseteq \{\mathcal{F}_t(x) \cup x\}$. Since $u \in \mathcal{F}_t(x) \setminus \mathcal{F}_t(u)$, we have $\{\mathcal{F}_t(u) \cup u\} \subseteq \{\mathcal{F}_t(x) \cup x\}$. □

Example 6. In Fig. 5, when $k = 4$, the graph is a k-truss with each edge labeled by its support. To find a best collapser, we compute the collapsed k-truss of each candidate vertex and choose the one with most followers. Suppose we compute for v_3 first and find $\mathcal{F}_t(v_3) = \{v_1, v_2, v_4\}$, we do not need to compute the collapsed k-truss of each vertex $u \in \mathcal{F}_t(v_3)$ because the follower number of u cannot be larger than $|\mathcal{F}_t(v_3)|$, according to Theorem 10.

Based on Theorem 10, every vertex which is a follower of another vertex should be excluded from candidate collapser set in the computation of a best collapser. Thus we can reduce even more vertices in the computation by checking promising collapsers first, which may have large number of followers. We say a vertex u *corresponds* to an edge e if $e \in E_{\triangle}(u, T_k(G))$. A vertex corresponding to more edges in the set D is more promising, because all these edges will follow the vertex to be deleted. The numbers of these edges in D can be accumulatively computed by visiting the triangles of each edge in D. Thus, to further prune unpromising candidate, we try collapsing vertices in decreasing order of $E_{\triangle}(u, T_k(G)) \cap D$ for each candidate u.

8.2 CKT Algorithm

Algorithm 6 shows the details of CKT algorithm which finds the best collapser for a given graph G (i.e., $b = 1$). Specifically, we first compute the k-truss of graph G (Line 1) and find the set D of edges with support $k-2$ in $T_k(G)$ (Line 2). According to Theorem 9, we find the candidate set $V_{\triangle}(D, T_k(G))$ where each vertex corresponds to at least one edge in D (Line 3-4). To compute $\mathcal{F}_t(u, G)$, we continue the k-truss computation in Line 1 with vertex u deleted (Line 5). The best collapser is produced by trying every candidate.

In Algorithm 6, the k-truss computation takes $\mathcal{O}(m^{1.5})$ time (Line 1), finding the set D takes $\mathcal{O}(m)$ by scanning the supports of k-truss edges once (Line 2) and finding the set V takes $\mathcal{O}(m^{1.5})$ by visiting the triangles containing the edges in D once (Line 3). The collapsed k-truss computation takes $\mathcal{O}(m^{1.5})$ for each vertex in V. Thus, the time complexity of Algorithm 6 is $\mathcal{O}(nm^{1.5})$.

Algorithm 6. CKT(G, k)

Input: G : a social network, k : support constraint,
Output: x : the best collapser
1 $T \leftarrow$ **ComputeTruss**(G, k);
2 $D \leftarrow \{e \mid sup(u, T) = k - 2\}$;
3 $V \leftarrow V_{\triangle}(D, T)$;
4 **for** each $u \in V$ (Theorem 9) **do**
5 Compute $\mathcal{F}_t(u, G)$;
6 $V \leftarrow V \setminus \mathcal{F}_t(u, G)$ (Theorem 10);
7 **return** the best collapser

For a general case with $b > 1$, our CKT algorithm can be simply embedded in the greedy algorithm (replacing Line 3 and 4) to find the best collapser in each iteration. If we find a vertex $u \in \mathcal{F}_t(x)$ in one iteration of the greedy algorithm, we do not need to consider u as a candidate in all following iterations because the followers of x is always not less than the followers of u, and u will be excluded from k-truss whenever x is removed (Theorem 10). Furthermore, in one iteration, if there is a connected k-truss subgraph which does not contain the produced best collapser, the result on this subgraph can be reused. The reason is that the collapsed k-truss of each vertex in the subgraph will keep same. We can share the computation by only recording the largest number of followers for a vertex in this subgraph.

The above optimizations do not affect the time complexity of Algorithm 6. Thus, our final CKT algorithm (Algorithm 5 equipped with Algorithm 6) has a time complexity of $\mathcal{O}(bnm^{1.5})$ where b is the number of resulting collapsers.

TABLE 3
Statistics of Datasets

Dataset	Nodes	Edges	d_{avg}	k_{max}^{core}	k_{max}^{truss}
Facebook	4,039	88,234	43.7	115	97
Brightkite	58,228	194,090	6.7	52	42
Gowalla	196,591	456,830	4.7	43	23
Yelp	552,339	1,781,908	6.5	105	73
YouTube	1,134,890	2,987,624	5.3	51	19
DBLP	1,566,919	6,461,300	8.3	118	119
Pokec	1,632,803	8,320,605	10.2	27	20
LiveJournal	3,997,962	34,681,189	17.4	360	352
Orkut	3,072,441	117,185,083	76.3	253	78

9 Evaluation

This section evaluates the effectiveness and efficiency of the proposed techniques through comprehensive experiments.

9.1 Experimental Setting

9.1.1 Algorithms

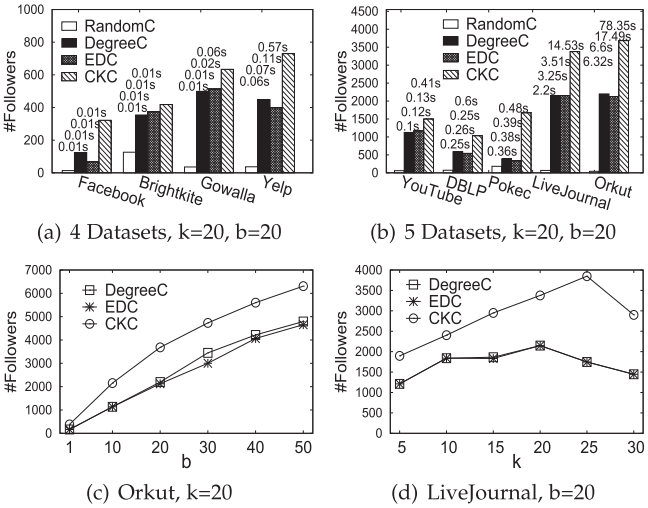
To the best of our knowledge, there is no existing work investigating the CCP and CTP. In this paper, we implement and evaluate the following algorithms.

- *BaselineC.* The baseline algorithm (Algorithm 2). In each iteration, it finds a best collapser by computing the collapsed k-core for every candidate vertex in current k-core.
- *CKC.* The advanced algorithm equipped with Algorithm 3 in each iteration, where Theorem 4 and Theorem 5 are applied.
- *BaselineT.* The baseline algorithm (Algorithm 5). In each iteration, it finds a best collapser by computing the collapsed k-truss for every candidate vertex in current k-truss.
- *CKT.* The advanced algorithm equipped with Algorithm 6 in each iteration, where Theorem 9 and Theorem 10 are applied.

The other algorithms will be introduced and marked in bold when appear in the experiments for the first time.

9.1.2 Datasets

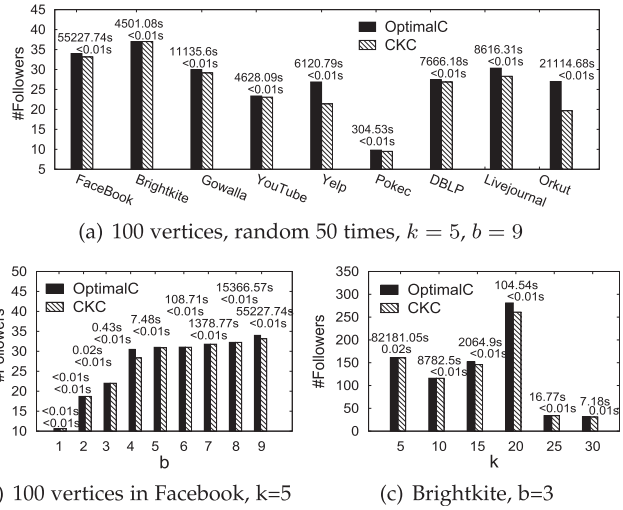
We deploy 9 real-life social networks in our experiments and we assume all vertices in each network are initially engaged. The original data of Yelp is from [43], DBLP is from [44] and the others are from [45]. In DBLP, each vertex represents an author and each edge between two authors represents there is at least one co-authored paper of the two authors. The other datasets have existing vertices representing users and edges representing relationships. Table 3 shows the statistics of the datasets in the experiments, in increasing order of their edge numbers.



(a) 4 Datasets, k=20, b=20 (b) 5 Datasets, k=20, b=20

(c) Orkut, k=20 (d) LiveJournal, b=20

Fig. 6. Number of the followers in collapsed k-core.



(a) 100 vertices, random 50 times, $k = 5$, $b = 9$

(b) 100 vertices in Facebook, k=5 (c) Brightkite, b=3

Fig. 7. Greedy versus optimal in collapsed k-core.

9.1.3 Parameters

We conduct experiments under different settings by varying degree (support) constraint k and the budget of collapsers b. The default value of k for collapsed k-core algorithms is 20. The default value of k for collapsed k-truss algorithms is 15. The default value of b is 20. In the experiments, the value of k varies from 5 to 50 and the value of b varies from 1 to 100.

All programs are implemented in standard C++ and compiled with G++ in Linux. All experiments are performed on a machine with Intel Xeon 2.8GHz CPU and Redhat Linux system.

9.2 Effectiveness

We first report the numbers of followers produced by CKC and CKT, compared with the results of other approaches. Then we show some statistical results and case studies of the two models.

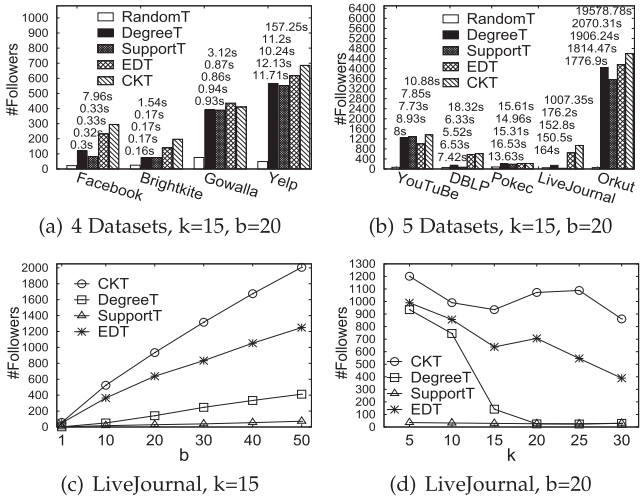
9.2.1 Effectiveness of CKC

Fig. 6 compares the number of followers w.r.t b collapsers identified by CKC algorithm with that of 3 other approaches. *RandomC* randomly chooses b collapsers from vertices in k-core. *DegreeC* chooses b collapsers in the candidate set T (Theorem 4) with the largest degrees in k-core. In each iteration of the b iterations, *EDC* greedily chooses a collapser in T (Theorem 4) with the largest number of neighbors which have degree k in k-core, where the k-core and T are updated in each iteration. For RandomC, we report the average number of the followers for 100 independent testings. Figs. 6a and 6b show that DegreeC and EDC significantly improve the performance, but it is outperformed by CKC with a large margin. This implies that it is not effective to find collapsers simply based on degree information. EDC and DegreeC often have the similar follower numbers, because they usually find the similar vertices (high degrees in some extent) as the collapsers. Figs. 6c and 6d report the impact of b and k on the number of followers. The number of the followers clearly grows with the increase of budget b. The number becomes relatively small when k is small or large, which is affected by the size and the structure of k-core. For convenience, we also report the time cost of the evaluated approaches. The 3 baseline approaches are more efficient than CKC because they only need to compute the collapsed k-core at most b times. The margins are not very large, because the collapsed k-core computation costs more time for a collapser with more followers.

To further justify the effectiveness of CKC, we compare its performance with that of optimal algorithm (*OptimalC*), which exhaustively computes the follower number for every combination of b collapsers on small networks. To compare the results with a largest possible b, we extract a small subgraph from each dataset by randomly selecting a vertex and choose its neighbors within x-hop until the vertex number reaches 100. For each dataset, we extract 50 such subgraphs. Then we compute the average follower number per subgraph on the same 50 subgraphs for OptimalC and CKC, respectively. Figs. 7a and 7b show that CKC achieves similar performance with that of the OptimalC. We can see OptimalC is very time-consuming and cannot return the result within 1 month for a large b. In Fig. 7c, CKC has almost the same number of followers with OptimalC on Brightkite when $b = 3$ and k is a reported value. We observe that the optimal follower number does not certainly increase or decrease for an increasing k. Thus the potential of collapsing power changes for different k on a dataset.

9.2.2 Effectiveness of CKT

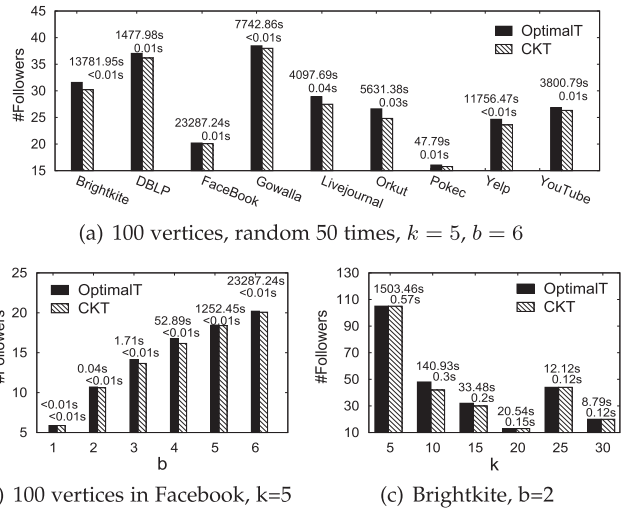
Fig. 8 compares the number of followers w.r.t b collapsers identified by CKT algorithm with that of 4 other approaches. Specifically, *RandomT* randomly chooses b collapsers in k-truss. *SupportT* chooses b collapsers from the vertices with the largest numbers of triangles containing a vertex in k-truss. *DegreeT* chooses b collapsers from the vertices with the largest degrees in k-truss. We define the effective degree of a vertex u in k-truss (T_k) by the number of edges in $E_{\triangle}(u, T_k)$ with support $k-2$ in T_k, where $E_{\triangle}(u, G)$ denotes the set of edges where each edge (v, w) belongs to a triangle (u, v, w) in G. In each iteration of the b iterations, *EDT* greedily chooses a collapser in the candidate set (Theorem 9) with the largest effective degree in k-truss, where the k-truss and effective degrees are updated in each iteration. For RandomT, we show the average number of the followers for 100 independent testings. Note that the input value of k for YouTube is 10 because its collapsed 15-truss is always empty when $b = 20$. Fig. 8 shows that CKT produces significantly more followers than the others, except one



(a) 4 Datasets, k=15, b=20 (b) 5 Datasets, k=15, b=20

(c) LiveJournal, k=15 (d) LiveJournal, b=20

Fig. 8. Number of the followers in collapsed k-truss.



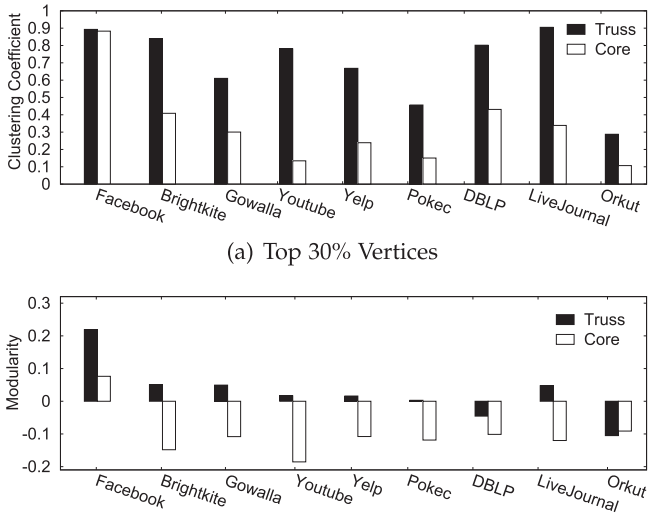
(a) 100 vertices, random 50 times, $k = 5$, $b = 6$

(b) 100 vertices in Facebook, k=5 (c) Brightkite, b=2

Fig. 9. Greedy versus optimal in collapsed k-truss.

setting where our greedy strategy is outperformed by EDT. The margins change on different k because the support distribution and the structure of k-truss become different. In Fig. 8, the CKT can always find the effective vertices for collapsing while the other approaches fail in some settings. The 4 baseline approaches are more efficient than CKT because they only need to compute the collapsed k-truss at most b times. The margins are not very large, because the collapsed k-truss computation costs more time for a collapser with more followers.

We also compare the performance of CKT with that of the optimal algorithm (*OptimalT*), which exhaustively computes the followers for every combination of b collapsers. In Figs. 9a and 9b, for each dataset, we compute the average follower number per subgraph on 50 subgraphs where each is extracted by selecting a vertex randomly and choosing its neighbors within x-hop until the vertex number reaches 100. We can see that OptimalT is very time-consuming. Fig. 9 shows that CKT achieves similar results with that of the OptimalT. The optimal follower number does not certainly increase or decrease for an increasing k. Thus the collapsing potential changes for different k on a dataset.



(a) Top 30% Vertices

(b) Top 30% Vertices

Fig. 10. Comparing core and truss.

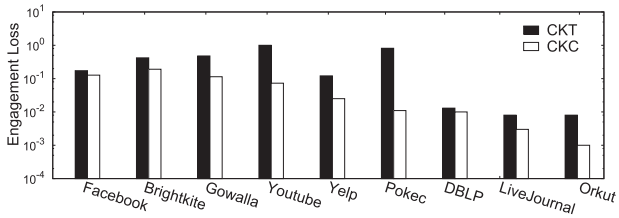


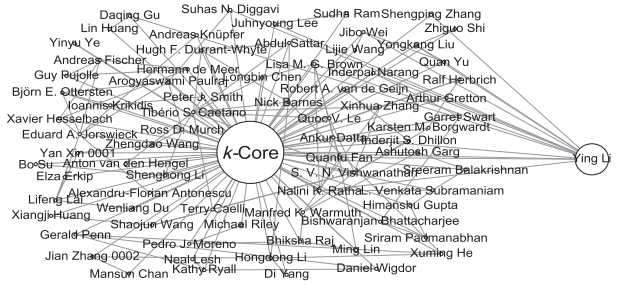
Fig. 11. Engagement loss of collapsing.

9.2.3 Comparison between Core and Truss

For a fair comparison on a dataset, we find a representative set of vertices which are the 30 percent vertices in the dataset with the largest core numbers for k-core model (resp. truss numbers for k-truss model). In Fig. 10, for each dataset, we report the global clustering coefficient and modularity values on the induced subgraphs of the representative set of vertices for k-core and k-truss, respectively. Note that the representative set of vertices is the user group with highest user engagement according to k-core and k-truss, respectively. Fig. 10a shows the k-truss vertices possess significantly higher clustering coefficients on all datasets than the k-core vertices. Fig. 10b shows the k-truss vertices also have better modularity values than the k-core vertices. The experiments indicate that utilizing k-truss may be more promising than k-core when the additional computation cost is affordable.

9.2.4 Comparison of Engagement Loss

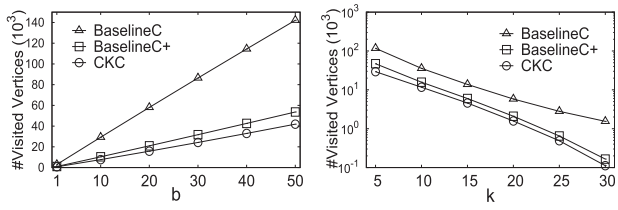
In Fig. 11, we report the engagement loss in the result of CKC and the result of CKT. For a given budget b, we consider the reduction percentage of the original non-collapsed subgraph as the engagement loss, i.e., the number of reduced vertices (i.e., followers and collapsers) divided by the number of vertices in the k-core or k-truss. Because the (k-1)-core is always a supergraph of k-truss, the values of k in Fig. 11 are 14 for CKC and 15 for CKT, respectively. The input value of b is 20 for both algorithms. In the figure, the collapsers of CKT reduce the original subgraph by a larger extent than CKC under the same collapsing budget. Particularly, the collapsed 15-truss by CKT on Orkut has 4598 followers while the collapsed 14-core only has 2907 followers for a



(a) CKC, k=20, b=1

(b) CKT, k=21, b=1

Fig. 12. Case studies on DBLP.



(a) DBLP, k=20 (b) DBLP, b=20

Fig. 13. Effectiveness of reducing candidate collapsers.

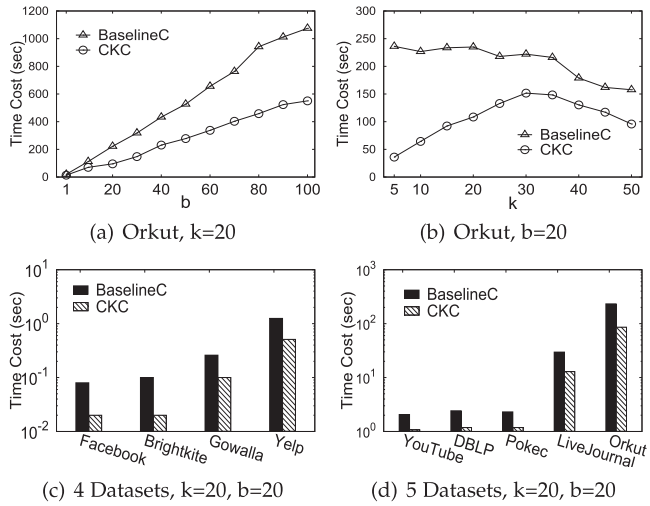
same budget b of 20. It shows utilizing collapsed k-truss model may be more effective on destroying the engagement of social communities.

9.2.5 Case Studies of CKC and CKT

Fig. 12 depicts the best collapser and the corresponding followers on DBLP identified by CKC and CKT, respectively, where $k = 20$ for CKC and $k = 21$ for CKT. For a clear presentation, the edges between each author and authors in k-core (resp. k-truss) are integrated as one edge. In Fig. 12a, it is interesting that the author "Ying Li" (the collapser) alone has 74 followers, and only 12 of them are co-authors of Li. This shows the user engagement can be severely damaged by the leave of a few individuals and the effect of leave cascade. In Fig. 12b, the author "Reinhold Haux" (the collapser) has 57 followers and 56 of them are co-authors of Haux. We can see the connection among followers and the collapser is significantly stronger in the collapsed k-truss than that of the collapsed k-core. The power of collapse by Haux is stronger in the sense that his leave breaks a tighter group than the leave of Li. This shows the CKT may better model the collapse of tight-knit user groups. The user leave caused by "Ying Li" may be affected deeper by the butterfly effect in leave cascade.

9.3 Efficiency

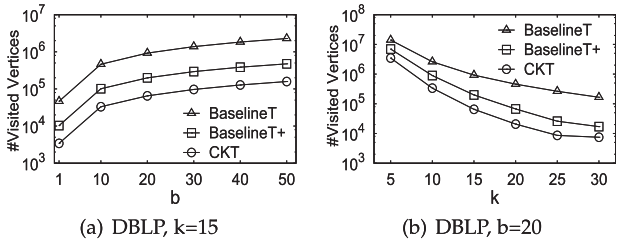
In this section, we evaluate the efficiency on individual techniques, the baselines, CKC and CKT.



(a) Orkut, k=20 (b) Orkut, b=20

(c) 4 Datasets, k=20, b=20 (d) 5 Datasets, k=20, b=20

Fig. 14. Performance of the CKC algorithms.



(a) DBLP, k=15 (b) DBLP, b=20

Fig. 15. Effectiveness of reducing candidate collapsers.

9.3.1 Evaluation of Individual Techniques in CKC

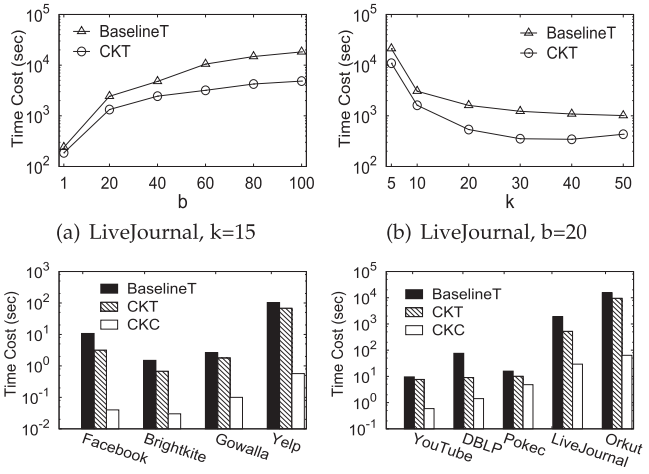
Fig. 13 reports the number of visited vertices, i.e., the size of candidate collapsers, in three algorithms. Algorithm *BaselineC+* represents BaselineC algorithm equipped with candidate collapser reduction technique (Theorem 4). We can see the number of visited vertices significantly drops on DBLP with the use of Theorem 4 for different k and b. It is reported that Theorem 5 further reduces the number of candidate collapsers, which is applied in CKC.

9.3.2 Performance Evaluation of CKC

Figs. 14a and 14b study the impact of k and b on two algorithms against Orkut, with b varying from 1 to 100 and k ranging from 5 to 50. We can see CKC is scalable with the growth of b. It is also efficient for different k, especially for a small or large value of k. With the increasing of k, the candidate number decreases while the k-core computation costs more time. These two factors together affect the trends of BaselineC and CKC on different k. Figs. 14c and 14d report the performance of two algorithms on 9 networks with $k = 20$ and $b = 20$. We can see CKC runs several times faster than BaselineC on all datasets. It is shown that CKC is also scalable to the growth of the network size, which identifies a set of 20 collapsers in 110 seconds on Orkut. The figures show that CKC significantly outperforms BaselineC under all settings.

9.3.3 Evaluation of Individual Techniques in CKT

Fig. 15 shows the number of the visited vertices, i.e., the size of candidate collapsers, in three algorithms, where *BaselineT+* is BaselineT equipped with Theorem 9. Note that CKT is further equipped with Theorem 10 on Baseline+. We can see the number of visited vertices significantly decreases by both of the theorems on DBLP for different k and b.



(a) LiveJournal, k=15 (b) LiveJournal, b=20

(c) 4 Datasets, k=15, b=20 (d) 4 Datasets, k=15, b=20

Fig. 16. Performance of the CKT algorithms.

9.3.4 Performance Evaluation of CKT

Fig. 16 shows performance of three algorithms on all datasets. In Fig. 16a, the runtime of CKT increases when we vary b from 1 to 100. In Fig. 16b, the runtime for CKT is relatively large for extremely small k and does not change much for other values of k. With the increasing of k, the candidate number usually decreases and the k-truss computation becomes slightly faster because we compute k-truss based on (k-1)-core. Note that k-core computation is faster than k-truss computation. From Figs. 16c and 16d, we can see that CKT outperforms BaselineT on all datasets by several times, where $k = 15$ and $b = 20$. We can see CKT outperforms BaselineT under all settings. The speedup on YouTube is not as significant as on the other datasets, because we uniformly set k as 15 such that the number of candidates on YouTube relatively is small (where $k_{max} = 19$) and the computation of collapsed k-truss on some effective collapsers dominates the runtime. The collapsed k-truss computation costs more time for a collapser with more followers. Although CKT costs more time than CKC ($k = 14$, $b = 20$) by computing the tie strength (i.e., edge support and triangle update), CKT shows better potential on effectiveness evaluation. It is consistent with the nature that k-truss is an enhanced version of k-core.

10 Conclusion

In this paper, we propose and study the problem of collapsed k-core (resp. k-truss), which intends to find a set of vertices whose deletion can lead to the smallest k-core (resp. k-truss) of the network. We prove both of the problems are NP-hard and inapproximate within a factor of $1 - 1/e$. Efficient algorithms are proposed, which significantly reduces the candidate number to speed up the computation. Extensive experiments on 9 real-life networks demonstrate our model is effective and our algorithm is efficient.

Acknowledgments

Fan Zhang is supported by Huawei YBN2017100007. Ying Zhang is supported by ARC DE140100679 and DP170103710. Lu Qin is supported by ARC DE140100999 and DP160101513. Wenjie Zhang is supported by ARC DP150103071 and DP150102728.

References

[1] F. D. Malliaros and M. Vazirgiannis, "To stay or not to stay: Modeling engagement dynamics in social graphs," in *Proc. 22nd ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 469–478.
[2] S. Wu, A. D. Sarma, A. Fabrikant, S. Lattanzi, and A. Tomkins, "Arrival and departure dynamics in social networks," in *Proc. 6th ACM Int. Conf. Web Search Data Mining*, 2013, pp. 233–242.
[3] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, "Preventing unraveling in social networks: The anchored k-core problem," *SIAM J. Discrete Math.*, vol. 29, no. 3, pp. 1452–1475, 2015.
[4] R. Chitnis, F. V. Fomin, and P. A. Golovach, "Parameterized complexity of the anchored k-core problem for directed graphs," *Inf. Comput.*, vol. 247, pp. 11–22, 2016.
[5] J. Abello and F. Queyroi, "Fixed points of graph peeling," in *Proc. IEEE/ACM Int. Conf. Advances Social Netw. Anal. Mining*, 2013, pp. 256–263.
[6] D. Garcia, P. Mavrodiev, and F. Schweitzer, "Social resilience in online communities: The autopsy of friendster," in *Proc. 1st ACM Conf. Online Social Netw.*, 2013, pp. 39–50.
[7] M. S. Granovetter, "The strength of weak ties," *Amer. J. Sociology*, vol. 78, no. 6, pp. 1360–1380, 1973.
[8] R. Rotabi, K. Kamath, J. M. Kleinberg, and A. Sharma, "Detecting strong ties using network motifs," in *Proc. 26th Int. Conf. World Wide Web Companion*, 2017, pp. 983–992.
[9] S. Aral and D. Walker, "Tie strength, embeddedness, and social influence: A large-scale networked experiment," *Manag. Sci.*, vol. 60, no. 6, pp. 1352–1370, 2014.
[10] F. Zhao and A. K. H. Tung, "Large scale cohesive subgraphs discovery for social network visual analysis," *Proc. VLDB Endowment*, vol. 6, no. 2, pp. 85–96, 2012.
[11] X. Huang, W. Lu, and L. V. S. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 77–90.
[12] E. Akbas and P. Zhao, "Truss-based community search: A truss-equivalence based indexing approach," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1298–1309, 2017.
[13] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," *Proc. VLDB Endowment*, vol. 10, no. 9, pp. 949–960, 2017.
[14] S. Wuchty and E. Almaas, "Evolutionary cores of domain co-occurrence networks," *BMC Evol. Biol.*, vol. 5, no. 1, 2005, Art. no. 24.
[15] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes, "K-core organization of complex networks," *Phys. Rev. Lett.*, vol. 96, no. 4, 2006, Art. no. 040601.
[16] P. Hagmann, L. Cammoun, X. Gigandet, R. Meuli, C. J. Honey, V. J. Wedeen, and O. Sporns, "Mapping the structural core of human cerebral cortex," *PLoS Biol.*, vol. 6, no. 7, 2008, Art. no. e159.
[17] M. Daianu, N. Jahanshad, T. M. Nir, A. W. Toga, C. R. J. Jr, M. W. Weiner, and P. M. Thompson, "Breakdown of brain connectivity between normal aging and alzheimer's disease: A structural k-core network analysis," *Brain Connectivity*, vol. 3, no. 4, pp. 407–422, 2013.
[18] M. Altaf-Ul-Amine, K. Nishikata, T. Korna, T. Miyasato, Y. Shinbo, M. Arifuzzaman, C. Wada, M. Maeda, T. Oshima, H. Mori, et al., "Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences," *Genome Inform.*, vol. 14, pp. 498–499, 2003.
[19] G. D. Bader and C. W. V. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC Bioinf.*, vol. 4, 2003, Art. no. 2.
[20] R. D. Luce and A. D. Perry, "A method of matrix analysis of group structure," *Psychometrika*, vol. 14, no. 2, pp. 95–116, 1949.
[21] S. B. Seidman and B. L. Foster, "A graph-theoretic generalization of the clique concept," *J. Math. Sociology*, vol. 6, no. 1, pp. 139–154, 1978.
[22] S. B. Seidman, "Network structure and minimum degree," *Social Netw.*, vol. 5, no. 3, pp. 269–287, 1983.
[23] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," *Nat. Security Agency Tech. Rep.*, Citeseer, vol. 16, 2008.
[24] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg, "Structural diversity in social contagion," *Proc. Nat. Academy Sci. United States America*, vol. 109, no. 16, pp. 5962–5966, 2012.