2018 IEEE 34th International Conference on Data Engineering

# Efficiently Reinforcing Social Networks over User Engagement and Tie Strength

Fan Zhang[§], Ying Zhang[†], Lu Qin[†], Wenjie Zhang[§], Xuemin Lin[§]

[§]*University of New South Wales,* [†]*CAI, University of Technology Sydney*
fan.zhang3@unsw.edu.au, {ying.zhang, lu.qin}@uts.edu.au, {zhangw, lxue}@cse.unsw.edu.au

***Abstract*—User engagement and tie strength are fundamental and important components in social networks. The model of $k$-truss not only captures actively engaged users, but also ensures strong tie strength among these users. It motivates us to utilize the model of $k$-truss in preventing network unraveling, which simultaneously considers both of the basic components. In this paper, we propose and investigate the anchored $k$-truss problem to reinforce a network by anchoring critical users who can significantly stop the unraveling. We prove the problem is NP-hard for $k \geq 4$. A fast edge deletion order based algorithm, named AKT, is proposed with efficient candidate exploration and pruning techniques based on the order. Comprehensive experiments on 10 real-life graphs demonstrate the effectiveness of our model and the efficiency of our methods.**

## I. Introduction

The leave of users can be serious for a social network, which not only brings down user engagement level, but also weakens the strength of user ties. Furthermore, the cascade of user departure may even lead to the death of a social network, such as Friendster which had over 115 million users at early 21st century [8], [22]. For the stability and activity of a social network, it is worthwhile to give the users some incentives to reinforce the network. Particularly, we need to consider both user engagement and tie strength in finding the target users, since the two elements are defining in social networks [4], [17], [19], [26], [28], [29].

Seidman [20] introduces the $k$-core model which is defined as a maximal subgraph where each vertex has at least $k$ neighbors in the subgraph. A lot of studies show that the large number of friends in a community for a user can ensure active engagement of this user [4], [5], [17]. However, the simple definition of $k$-core corresponds to promiscuous subgraphs, and thus the $k$-core is considered as "seedbeds, within which cohesive subsets can precipitate out" [20]. Consider the fact that the social ties (i.e., edges) have different strength in real social networks, Cohen [6] introduces the $k$-truss model which is a maximal subgraph where each edge is contained in at least $k-2$ triangles in the subgraph. It is effective to use the number of triangles containing an edge to estimate the strength of the edge [12], [19], [23], [33]. The $k$-truss is essentially an enhanced subgraph of ($k$-1)-core, since a user in the $k$-truss community also has at least $k-1$ friends within the community. By considering both user engagement and tie strength, the $k$-truss model not only ensures strong tie strength among users, but also captures users with high engagement. A social network tends to be a $k$-truss community with the removing of weak ties and the leaving of weakly engaged users. Consequently, it is promising to utilize the model of $k$-truss in modeling network unraveling.

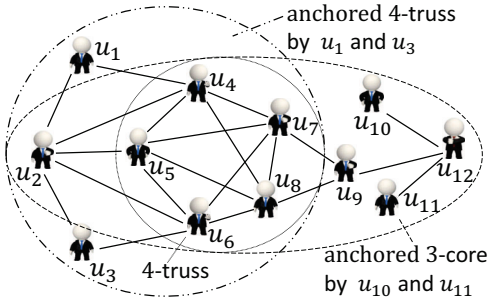



Fig. 1. Motivation Example

The objective of this paper is to reinforce the social networks by giving incentives to some users so that they will keep engaged together with their relatively strong ties. Since the budget of social networking sites is limited, we need to find the users who have the greatest power to enlarge the finally engaged communities. Specifically, we propose and investigate the problem of anchored $k$-truss, which is to retain (anchor) a number $b$ of users (anchors) in the network such that the resulting $k$-truss community has the largest number of users.

**Example 1.** *Suppose there is a project group where each member is modeled as a vertex and a friendship between two members is modeled as an edge. The number of common friends in the group between two friends represents the strength of their relationship. In Figure 1, we model 12 initial members and their relationship as a network. In $k$-truss computation, the weak friendships, i.e., the edges involving in fewer than $k-2$ triangles, are removed. The 4-truss of the network contains $u_4$, $u_5$,...,$u_8$. Note that the deletion of edges reduces triangles which may incur the deletion of other edges. Correspondingly, the 3-core of the network contains $u_2$, $u_4$, $u_5$,..., $u_8$. To reinforce the network, we may persuade (anchor) the members $u_1$ and $u_3$ to keep engaged through additional incentives, such as salary bonus or rewards. Thus the edges incident to $u_1$ and $u_3$ can exist in the $k$-truss as long as involving in at least one triangle in the $k$-truss. As a result, $u_2$ will also remain engaged since the corresponding edges now involve in enough triangles in the $k$-truss. That is, by anchoring $u_1$ and $u_3$, the anchored 4-truss contains $u_1$, $u_2$, ..., $u_8$. Note that this anchored 4-truss has 8 vertices which is the maximum one among any two anchored vertices.*

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00057

*Towards the $k$-core model, the maximum anchored 3-core with two anchored vertices (here the anchors always exist in the anchored $k$-core) has 10 vertices which contains $u_2$, $u_4$, $u_5$, ..., $u_{12}$. The maximum anchored 4-truss and 3-core both contain unique (unshared) vertices inside them, which demonstrates that the anchored $k$-truss and anchored $k$-core models are inherently different. The maximum anchored 4-truss is tighter than the maximum anchored 3-core in the example.*

**Challenges.** To the best of our knowledge, we are the first to study the anchored $k$-truss problem to reinforce social networks. We prove the problem is NP-hard when $k \geq 4$. Later in the paper, we will show that the study of anchored $k$-truss problem is meaningless when $k \leq 3$ since in this case, anchoring any vertices will not bring any extra vertices into the anchored $k$-truss. Thus, the anchored $k$-truss problem is essentially NP-hard for each $k$ value. Although computing the $k$-truss is PTIME, all the possible combinations of the $b$ anchors make the problem NP-hard. To avoid enumerating all possible anchor sets with size $b$, we adopt a greedy strategy to find a best anchor in each iteration. As demonstrated in our empirical study, a straightforward implementation of the greedy algorithm for our problem is very time consuming, due to the large number of candidate anchors. Since anchoring a vertex in $k$-truss may also enlarge the $k$-truss, the initial candidate anchors are all the vertices in the graph.

Bhawalkar, Kleinberg *e*t al. [4] introduce the problem of anchored $k$-core to prevent network unraveling. Zhang *e*t al. [30] propose an efficient algorithm to solve the problem. Nevertheless, the anchored $k$-core model only considers the perspective of user engagement and thus cannot solve our anchored $k$-truss problem, as shown in the following two aspects. (1) From the technical perspective, these two problems are inherently different. The techniques for anchored $k$-core problem [30] are based on vertex deletion order, while the anchored $k$-truss problem is defined against edges and triangles. Consequently, the pruning rules, early termination and other techniques developed in [30] cannot be applied in the anchored $k$-truss problem. (2) From the semantic perspective, the model of $k$-truss can estimate the strength [19] of an edge by counting the number of triangles; nevertheless, the model of $k$-core treats each edge equally, which ignores the fact that the edges in social communities possess unequal strength and produce different influence [7], [9], [24].

**Our Solution.** A straightforward consideration is whether we can enforce the *vertex* deletion order in a structure to facilitate the anchored $k$-truss computation, i.e., order the vertices by the time they become isolated or have insufficient degrees in the $k$-truss computation. Unfortunately, it is useless because the vertex deletion order and the inherent support relation among vertices in $k$-truss computation are based on the edge deletion order. Consequently, we design an auxiliary structure $\mathcal{L}$ based on edges, named *edge layers*, which divides a small set of *edges* by layers. Based on the well-designed structure, we develop corresponding efficient techniques.

The existence of anchor vertices may retain more triangles in the $k$-truss subgraph. Then some new vertices may survive in the $k$-truss, which are named *followers* in this paper. The number of followers is regarded as the *gain* of the anchoring action. In our greedy method, we focus on finding the best anchor in each iteration, i.e., the vertex with the largest number of followers on current graph. We have an observation that when considering only one anchor, all followers come from the ($k$-1)-hull, i.e., the vertices in ($k$-1)-truss but not in $k$-truss. We put the vertices and edges in the triangles, which contain an edge in ($k$-1)-hull, into $\mathcal{L}$ and order the edges by layers.

Then, we only need to consider the vertices in $\mathcal{L}$ as the candidate anchors, which significantly reduces the number of candidate anchors for the computation of followers. Besides, the follower computation can be restricted to a small part in $\mathcal{L}$ according to our strict search rules, which significantly reduces the search space. Furthermore, we also develop early termination and candidate anchor pruning techniques based on the *edge layer* structure to eliminate non-promising candidates. All of these techniques form the final algorithm, named `AKT`, which can efficiently identify the best anchors for the problem.

**Contributions.** Our principal contributions are the followings.

- We propose and investigate the anchored $k$-truss problem which simultaneously considers user engagement and tie strength in network structure to reinforce the social networks. We prove the problem is NP-hard and non-submodular when $k \geq 4$.
- We introduce a novel *edge layer* structure $\mathcal{L}$, which divides a small set of *edges* by layers. The structure also contains all the endpoints of the edges inside it. Then only the vertices in $\mathcal{L}$ need to be considered as candidate anchors, and we can efficiently find a best anchor. The follower computation is also restricted on $\mathcal{L}$, which significantly reduces the search space.
- We develop an efficient algorithm, named `AKT`, based on the well-organized structure $\mathcal{L}$, such that `AKT` can compute the followers for the candidate anchor in a layer-by-layer paradigm. With the concept of triangle hold path, we only need to explore a very small portion of the edges in $\mathcal{L}$. Together with early termination and pruning techniques, we further reduce the number of anchor and follower candidates, which significantly enhances performance of the algorithm.
- Our comprehensive experiments on 10 real-life networks demonstrate the model effectiveness and algorithm efficiency. For instance, `AKT` can further preserve 1184 vertices in $k$-truss by anchoring 20 vertices in the `Orkut` network. Regarding the running time, our `AKT` algorithms outperform the baselines by orders of magnitude.

## II. Preliminaries

In this section, we first give some necessary notations and introduce the $k$-truss. Then, we formally define the anchored $k$-truss problem and show its hardness.

### A. Problem Definition

We consider an unweighted and undirected graph $G = (V, E)$, where $V$ (resp. $E$) represents the set of vertices (resp. edges) in $G$. We denote $n = |V|$, $m = |E|$ and assume $m > n$.

TABLE I
SUMMARY OF NOTATIONS

| Notation | Definition |
|---|---|
| $G$ | an unweighted and undirected graph |
| $u$, $v$, $x$ | a vertex in the graph |
| $e$ | an edge in the graph |
| $n$, $m$ | the number of vertices and edges in $G$ |
| $A$ | a set of anchor vertices |
| $NB(u, G)$ | the set of adjacent vertices of $u$ in $G$ |
| $deg(u, G)$ | the number of adjacent vertices of $u$ in $G$ |
| $sup(e, G)$ | the number of triangles each contains $e$ in $G$ |
| $G_x$; $G_A$ | the graph $G$ anchored by $x$; $A$ |
| $C_k(G)$; $T_k(G)$ | the $k$-core of $G$; the $k$-truss of $G$ |
| $T_k(G_A)$ | the anchored $k$-truss with $A$ in $G$ |
| $H_k(G)$; $H_k^+(G)$ | $k$-hull of $G$; $k$-hull$^+$ of $G$ |
| $\mathcal{L}$ | the *edge layers* of $G$ |
| $L_0^s$ | the edge set of $\mathcal{L}$ in $G$ (with $s+1$ layers) |
| $L_i$; $L_i^j$ | the edges on $i$-th layer of $\mathcal{L}$; $\bigcup_{i\le k\le j} L_k$ |
| $l(e)$ | the layer index of the edge $e$ in $\mathcal{L}$ |
| $\mathcal{F}(x)$ ($\mathcal{F}(A)$) | the followers of an anchor $x$ ($A$) |
| $CF(x)$ | the candidate followers of an anchor $x$ |
| $s^+(e)$ | support upper bound of $e$ in $T_k(G_x)$ |
| $V(e, G)$ | the vertex set where each vertex is incident to $e$ in $G$ |
| $V(S, G)$ | the union set of $V(e, G)$ for each $e \in S$ |
| $V_\triangle(e, G)$ | the set of vertices where each vertex from a containing-$e$-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 $e \in S$ |
| $E(u, G)$ | the edge set where each edge is incident to $u$ and is in $G$ |
| $E(S, G)$ | the union set of $E(u, G)$ for each $u \in S$ |

$NB(u, G)$ is the set of adjacent vertices of $u$ in $G$. Let $S$ denote a subgraph of $G$. We use $deg(u, S)$, the degree of $u$ in $S$, to represent the number of adjacent vertices of $u$ in $S$. A triangle is a cycle of length 3 in the graph. A containing-$e$-triangle is a triangle which contains $e$. We use $sup(e, S)$, the *support* of $e$ in $S$, to represent the number of containing-$e$-triangles in $S$. We say a vertex $u$ is incident to an edge $e$, or $e$ is incident to $u$, if $u$ is one of the endpoints of $e$. When the context is clear, we omit the the second parameter in notations, such as $sup(e)$ for $sup(e, G)$. The notations are summarized in Table I.

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

Besides $k$-core [20], the model of $k$-truss [6] is also a widely studied subgraph model in social networks.

**Definition 2.** *$k$-**truss**. Given a graph $G$, a subgraph $S$ is the $k$-truss of $G$, denoted by $T_k(G)$, if (i) $sup(e, S) \ge k-2$ for every edge $e \in S$; (ii) $deg(u, S) \ge k-1$ for every vertex $u \in S$; (iii) $S$ is maximal, i.e., any subgraph $S' \supset S$ is not a $k$-truss; and (iv) $S$ is non-trivial, i.e., no isolated vertex in $S$.*

Note that we have $T_k(G) \subseteq C_{k-1}(G)$ and $T_k(G) \subseteq T_{k-1}(G)$. As shown in Algorithm 1, we firstly compute ($k$-1)-core $C_{k-1}(G)$ of a graph $G$, then recursively remove every edge whose support is less than $k-2$. We can get the $k$-truss after removing isolated vertices. 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. We define $k$-hull and $k$-hull$^+$ as follows.

**Algorithm 1**: **ComputeTruss($G$, $k$)**

**Input** : $G$ : a social network, $k$ : support constraint
**Output** : $T_k(G)$
**while** exists $u \in G$ with $deg(u, G) < k-1$ **do**
  $G$:= $G \setminus \{u \cup E(u, G)\}$;
**while** exists an edge $e \in G$ with $sup(e, G) < k-2$ **do**
  $G$:= $G \setminus \{e\}$;
Delete isolated vertices in $G$;
**return** $G$

**Definition 3.** *$k$-**hull** and $k$-**hull**$^+$. Given a graph $G$, the $k$-hull of $G$ is denoted by $H_k(G)$, and $H_k(G) = T_k(G) \setminus T_{k+1}(G)$; the $k$-hull$^+$ of $G$ is denoted by $H_k^+(G)$, and $H_k^+(G) = H_k(G) \cup V(H_k(G), G)$.*

The only difference between $k$-hull and $k$-hull$^+$ is that $k$-hull$^+$ further contains some vertices in $k$-truss which are incident to an edge in $k$-hull and these vertices do not exist in $k$-hull. The edges in $k$-hull and $k$-hull$^+$ are the same.

In the problem for $k$-truss, once a vertex $x$ in $G$ is **anchored**, it is always retained in $k$-truss. For every edge $e$ which is incident to $x$ ($e$ is called an **anchor edge**), $e$ is retained in $k$-truss (denoted by $T$) if $sup(e, T) > 0$. This definition avoids keeping those relatively isolated vertices in anchored $k$-truss such as the anchor $x$'s neighbors who are only adjacent to $x$.

**Definition 4.** *anchored $k$-**truss**. Given a graph $G$ and a vertex set $A \subseteq G$, the anchored $k$-truss, denoted by $T_k(G_A)$, is the corresponding $k$-truss of $G$ with vertices in $A$ anchored.*

The computation of anchored $k$-truss is the same with the computation of $k$-truss except that (1) for anchor edges, we delete them if and only if their supports are less than one; and (2) we delete every vertex whose degree is less than $k-1$. In addition to the vertices in $T_k(G)$, more vertices (i.e., **followers**, denoted by $\mathcal{F}(A, G)$) might be retained in the $T_k(G_A)$ due to the contagious nature of the $k$-truss computation. Formally, we have $\mathcal{F}(A, G)$ = the vertices in $T_k(G_A) \setminus T_k(G)$.

**Problem Statement.** Given a graph $G$, a support constraint $k$ and a budget $b$, the **anchored $k$-truss problem** aims to find a set $A$ of $b$ vertices in $G$ such that the size of anchored $k$-truss, $T_k(G_A)$, is maximized; that is, $\mathcal{F}(A, G)$ is maximized.

### B. Problem Complexity

**Theorem 1.** *Given a graph $G$, the anchored $k$-truss problem is NP-hard when $k \ge 4$.*

*Proof:* When $k \le 2$, the $k$-truss of $G$ is the graph $G$. When $k = 3$, an edge exists in $k$-truss if its support is at least 1, thus no followers exist for anchoring any vertex according to the definition of anchor in Section II-A. When $k > 3$, we reduce the anchored $k$-truss problem to the maximum coverage problem [15]; that is, finding at most $b$ sets to cover the largest number of elements, where $b$ is a given budget. We consider an arbitrary instance of maximum coverage problem with $c$

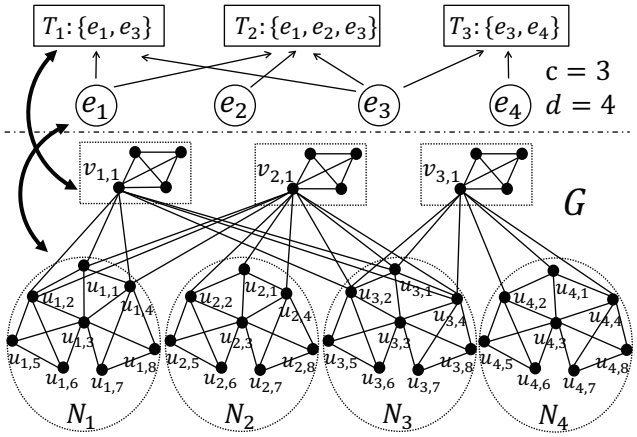



Fig. 2. Construction Example for NP-hardness Proof, $k = 4$

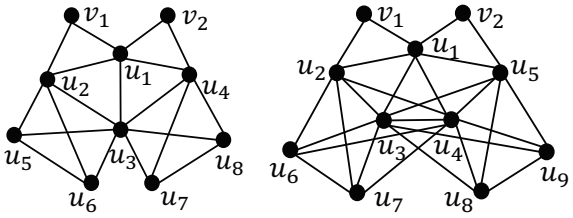
sets $T_1, T_2, ..., T_c$ and $d$ elements $\{e_1, e_2, ..., e_d\} = \cup_{1 \le i \le c} T_i$. Then we construct a corresponding instance of the anchored $k$-truss problem in a graph $G$ as follows.

There are two sets of vertices in $G$, denoted by $\cup_{1 \le i \le c} M_i$ ($M$ in short) and $\cup_{1 \le j \le d} N_j$ ($N$ in short). Each subset $M_i (1 \le i \le c)$ contains $k$ vertices, i.e., $M = \{v_{i,p} \mid 1 \le p \le k\}$, and forms a $k$-clique, i.e, every vertex pair has an edge. Each subset $N_j (1 \le j \le d)$ contains $k+4$ vertices, i.e., $N_j = \{u_{j,p} \mid 1 \le p \le k+4\}$. Particularly, the vertex set $\{u_{j,p} \mid 1 \le p \le k\}$ forms a lack-one-edge clique, i.e., there is an edge between each pair of vertices in the set, except for vertices $u_{j,2}$ and $u_{j,k}$. Besides, the vertex set $\{u_{j,p} \mid 2 \le p \le k-1 \ or \ k+1 \le p \le k+2\}$ and $\{u_{j,p} \mid 3 \le p \le k \ or \ k+3 \le p \le k+4\}$ form a $k$-clique, respectively. For each set $T_i (1 \le i \le c)$ and each element $e_j (1 \le j \le d)$, if $e_j \in T_i$, we add 1 edge from $v_{i,1}$ to $u_{j,1}, u_{j,2}$ and $u_{j,k}$, respectively.

With the construction, we ensure that for any $i$ and $j$ ($1 \le i \le c, 1 \le j \le d$) (i) there is no edge between $v_{i,1}$ and $\{u_{j,p} \mid 3 \le p \le k-1 \ or \ k+1 \le p \le k+4\}$, thus the supports of $e(v_{i,1}, u_{j,1})$, $e(v_{i,1}, u_{j,2})$ and $e(v_{i,1}, u_{j,k})$ are 2, 1 and 1, respectively; (ii) there is no edge between $u_{j,2}$ and $u_{j,k}$, thus the support of $e(u_{j,1}, u_{j,2})$ in $N$ is $k-3$ because there are only $k-3$ common neighbors $\{u_{j,p} \mid 3 \le p \le k-1\}$. Similarly, the support of $e(u_{j,1}, u_{j,k})$ in $N$ is $k-3$ for the same common neighbors with $e(u_{j,1}, u_{j,2})$; (iii) for each edge $e(u_{j,1}, u_{j,q}) (3 \le q \le k-1)$, its support in $G$ is $k-2$ because there are only $k-2$ common neighbors $\{u_{j,p} \mid 2 \le p < q \ or \ q < p \le k\}$; (iv) all the supports of other edges are at least $k-2$ because they belong to at least 1 $k$-clique.

By doing this, we ensure that (i) the $k$-truss of $G$ consists of $\cup_{1 \le i \le c} M_i$, $\{u_{j,p} \mid 2 \le p \le k+4 \ and \ 1 \le j \le d\}$ and corresponding edges; (ii) if $e_j \in T_i$ and $v_{i,1}$ is anchored, we have and only have $u_{j,1}$ as its follower in $T_k(G_A)$; (iii) anchoring a vertex in $G \setminus \{v_{i,1} \mid 1 \le i \le c\}$ cannot have any followers. Consequently, the optimal solution of anchored $k$-truss problem corresponds to optimal solution of the maximum coverage problem which is NP-hard. Consequently, the anchored $k$-truss problem is NP-hard for $k \ge 4$. ■

**Theorem 2.** *Let $f(A) = |\mathcal{F}(A)|$. We have $f$ is monotone but not submodular when $k \ge 4$.*




(a) $k = 4$ (b) $k = 5$

Fig. 3. Examples for Non-submodular

*Proof:* When $k \le 3$, no followers (besides the anchors) exist for anchoring any vertex in $G$. When $k \ge 4$, suppose there is a set $A' \supseteq A$. For every vertex $u$ in $\mathcal{F}(A)$, $u$ will still exist in the anchored $k$-truss $T_k(G'_A)$, because anchoring more vertices in $A' \setminus A$ cannot decrease the supports of $E(u)$ and the degree of $u$. Thus $f(A') \ge f(A)$ and $f$ is monotone. For two arbitrary anchor sets $A$ and $B$, if $f$ is submodular, it must hold that $f(A \cup B) + f(A \cap B) \le f(A) + f(B)$. We show that the inequality does not hold using counterexamples. As Figure 3 shows, we firstly construct a graph $G$ consisting of two vertices $v_1$, $v_2$ and the induced subgraph of $N_1$ in the proof of Theorem 1, here $N_1 = \{u_i \mid 1 \le i \le k+4\}$. After adding edges from $v_1$ to $u_1$ and $u_2$, and edges from $v_2$ to $u_1$ and $u_k$, the construction of $G$ is completed. Suppose $A = \{v_1\}$ and $B = \{v_2\}$, we have $\mathcal{F}(A \cup B) = \{u_1\}$, $\mathcal{F}(A \cap B) = \mathcal{F}(A) = \mathcal{F}(B) = \emptyset$, so the inequation does not hold and thus $f$ is not submodular. ■

Note that there is no need to discuss the problem when $k \le 3$ since no followers (besides the anchors themselves) can be produced.

## III. Solution

Due to the non-submodular property on the anchored $k$-truss problem, it is unpromising to estimate or predict the anchoring result of multiple anchors. In a basic exact solution, we have to exhaustively compute the anchored $k$-truss on every possible set $A$ with size $b$. The time complexity of $\mathcal{O}(\binom{n}{b} m^{3/2})$ is cost-prohibitive. Besides, considering the NP-hardness of the problem, we adopt a greedy heuristic which iteratively finds the best anchor, i.e., the vertex with the largest number of followers in each iteration. Although the greedy algorithm does not have submodular property, our experiments show that the resulting numbers of followers are similar with the optimal result on available small settings, and are much larger than the results of other methods under all the settings.

As shown in Algorithm 2, we compute the number of followers (Line 4) for each vertex in the graph (Line 3). Note that a vertex $u$ in $k$-truss is also a candidate anchor because anchoring $u$ may have followers. Specifically, some non-$k$-truss edges of $u$ may be anchored and support other vertices to survive in the anchored $k$-truss. The time complexity is $\mathcal{O}(bn \times m^{3/2})$, where $n$ is the number of candidate anchors in each iteration and $m$ is the number of edges in follower computation, i.e., $k$-truss computation. However, this greedy algorithm is still unscalable on large datasets, which motivates us to significantly improve the following aspects: (1) the number of candidate anchors in each iteration (Line 3); and (2) the computation cost of finding followers (Line 4).

**Algorithm 2**: **GreedyAKT**($G$, $k$, $b$)

**Input** : $G$ : a social network, $k$ : support constraint, $b$ : number of anchor vertices
**Output** : $A$ : the set of anchor vertices

1 $A := \emptyset$; $i := 0$;
2 **while** $i < b$ **do**
3  **for** each $u \in G \setminus A$ **do**
4  Compute $\mathcal{F}(A \cup u, G)$;
5  $u^* \leftarrow$ the best anchor vertex in this iteration;
6  $A := A \cup u^*$; $i := i + 1$;
7 **return** $A$

To address the above issues, we firstly propose a novel structure called *edge layers*. For ease of understanding, we discuss the problem on the first iteration of the greedy algorithm (i.e., $A = \emptyset$ and $i = 0$) in the rest of the paper. The following techniques can be directly applied to any iteration of the greedy algorithm when we substitute the $k$-truss with the anchored $k$-truss (Section III-E).

### A. The Edge Layers

The deletion order of non-$k$-truss edges in Algorithm 1 is not unique because there may exist multiple edges with insufficient supports to delete. We say a deletion order of edges in $k$-truss computation is *valid* if (1) every edge in the order violates the support constraint at the time it is deleted; and (2) all the remaining edges satisfy the support constraint when the computation terminates. Theorem 3 shows any valid order can leads to the same $k$-truss.

**Theorem 3.** *Algorithm 1 always returns the same $T_k(G)$ w.r.t any valid deletion order of non-$k$-truss edges.*

*Proof:* Suppose there are two different valid deletion orders, $O_1$ and $O_2$, leading to two different $k$-trusses $T_1$ and $T_2$, respectively. Let $M = T_1 \setminus T_2$ and $M \neq \emptyset$. This implies that all edges in $M$ are discarded in the access order $O_2$. Suppose $e_1$ is the first removed edge in $M$, we have $sup(e_1, M \cup T_2) \geq k-2$ because $sup(e_1, T_1) \geq k-2$ and none of the edges in $T_2$ or $M$ are removed when $e_1$ is accessed. This implies $O_2$ is not a valid order. Consequently, the theorem holds. ∎

Theorem 3 motivates us that we can utilize the inner property in a specific order to compute the anchored $k$-truss. An *edge layer* structure $\mathcal{L}$ mainly based on $k$-hull$^+$ is proposed to facilitate computation. $\mathcal{L}$ consists of $s+1$ layers of edges, $\{L_0, L_1, \ldots, L_s\}$ (i.e., $L_0^s$) where $s$ is the largest layer number, and corresponding vertices which is incident to at least one edge in $L_0^s$. The pseudo-code is shown in Algorithm 3. We first compute $T_{k-1}(G)$ at Line 1, then start to peel the ($k$-1)-hull by removing *all* edges not satisfying the support constraint at the same time (Lines 2 and 6), which are kept in the same layer (Line 4). When the peeling process terminates, we have $i = s$, the edge set of $\mathcal{L}$ is $L_0^s$ which is same to the edge set of $H_{k-1}(G)$, and $N = T_k(G)$ after removing isolated vertices in $N$. Then we put the edges (excluding the ones in $T_{k-1}(G)$) between common neighbors of endpoints and the endpoints, for every edge in $H_{k-1}(G)$, to $L_0$ as the layer 0 (Line 7).

**Algorithm 3**: **ProduceLayers**($G$, $k$)

**Input** : $G$ : a social network, $k$ : support constraint
**Output** : the edge set of *edge layers*, i.e., $L_0^s$

1 $N := T_{k-1}(G)$; $i := 0$;
2 $P := \{e \mid sup(e, N) < k-2 \ \& \ e \in N\}$;
3 **while** $P \neq \emptyset$ **do**
4  $i := i + 1$; $L_i := P$;
5  $N := N \setminus P$;
6  $P := \{e \mid sup(e, N) < k-2 \ \& \ e \in N\}$;
7 $L_0 := \{(u, v) \mid u \in V_\triangle(e, G) \ \& \ v \in V(e, G) \ \& \ (u, v) \notin T_{k-1}(G)$, for each $e \in L_1^s\}$;
8 **return** $L_0^i$

The time complexity of Algorithm 3 is $\mathcal{O}(m^{3/2})$. Although we need to update the *edge layers* in each iteration of the greedy algorithm, the cost is dominated by the large number of follower computations. We use $L_i^j$ ($i < j$) to denote the edges between layer $i$ and layer $j$ (inclusive), and $l(e)$ to denote the layer index of an edge $e$ in $\mathcal{L}$.

**Definition 5.** ***Edge Layers:* $\mathcal{L}$.** *Given a graph $G$ and a support constraint $k$, the edge set of $\mathcal{L}$ is $L_0^s$, the output of Algorithm 3; that is, $L_0^s = \{e \mid e \in H_{k-1}(G)\} \cup \{(u, v) \mid u \in V_\triangle(e, G) \ \& \ v \in V(e, G) \ \& \ (u, v) \notin T_{k-1}(G)$, for each $e \in L_1^s\}$. The full set $\mathcal{L}$ further contains the corresponding vertices, i.e., $\mathcal{L} = L_0^s \cup H_{k-1}^+(G) \cup \{V_\triangle(H_{k-1}(G), G) \setminus T_{k-1}(G)\}$.*

**Example 2.** *In Figure 4 with $k = 4$, we have 3-truss $T_3$ which is the whole graph except vertex $v_{12}$ and edge $(v_{11}, v_{12})$, and the 4-truss $T_4$ induced by $\{v_5, v_6, v_7, v_8, v_9\}$. According to Algorithm 3, we firstly label 1 to the edges in $T_3$ whose supports are less than $k-2$. After removing these edges, new edges with insufficient supports are produced and labeled. Recursively, we label every edge in $\mathcal{L}$ as the figure shows. Note that there is no $L_0$ edges when $k = 4$, and $\mathcal{L}$ also contains the endpoints of each edge in $L_0^s$.*

### B. Candidate Anchors and Followers

The following theorem shows that the follower computation can be done on a small set, because only the vertices in $H_{k-1}(G)$ can become followers of an anchor.

**Theorem 4.** *Given a graph $G$ and its (k-1)-hull $H_{k-1}(G)$, if a vertex $x$ is anchored, all of its followers except $x$ come from (k-1)-hull; that is, $u \in F(x, G)$ implies $u \in H_{k-1}(G)$ or $u = x$.*

*Proof:* Let $E^*$ be all the edges in $G \setminus T_{k-1}(G)$, We firstly prove that the edges in $E^* \setminus E(x, G)$ will still be deleted in computing $T_k(G_x)$. Let $O$ be the deletion order of edges in computing $T_{k-1}(G)$, the support of each edge is less than $k-3$ when it is to be deleted in the order of $O$. Then in the computation of $T_k(G_x)$, we still follow the order $O$. Because anchoring $x$ can only increase the support of an edge in $E^* \setminus E(x, G)$ by at most 1, the support of every edge in $O$ is less than $k-2$ when it is visited (i.e., deleted). Consequently, all the edges in $E^* \setminus E(x, G)$ cannot exist in $T_k(G_x)$. After deleting $E^* \setminus E(x, G)$, all the vertices in $G \setminus \{T_{k-1}(G) \cup x \cup NB(x, G)\}$ are also deleted since they are isolated. For every vertex $u$ in $(G \setminus T_{k-1}(G)) \cap NB(x, G)$, the anchor edge $e(u, x)$ will be

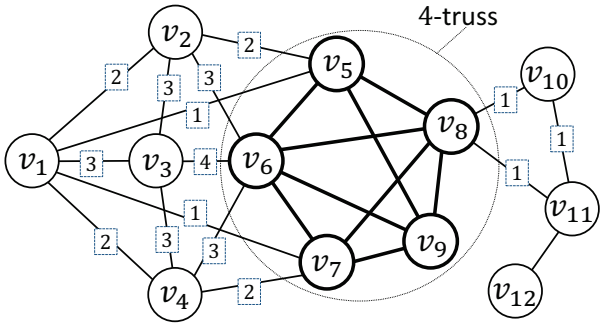



Fig. 4. Examples for Edge Layers $\mathcal{L}$, $k = 4$

deleted, because all of its adjacent non-anchor edges have been deleted and thus the support of $e(u, x)$ is 0. So the vertices in $(G \setminus T_{k-1}(G)) \cap NB(x, G)$ are deleted, which means all the vertices in $G \setminus \{T_{k-1}(G) \cup x\}$ are deleted. Since a vertex in $T_k(G)$ cannot be a follower, all the followers (except $x$ itself) of $x$ are from $T_{k-1}(G) \setminus T_k(G)$, i.e., $H_{k-1}(G)$. ■

We can also significantly reduce candidate anchors, because an anchor outside of $\mathcal{L}$ is unpromising.

**Theorem 5.** *Given a graph $G$, if an anchored vertex $x$ has at least one follower besides $x$, $x$ is from $\mathcal{L}$; that is, $|F(x, G) \setminus \{x\}| > 0$ implies that $x \in \mathcal{L}$.*

*Proof:* According to the proof of Theorem 4, we can only save some $H_{k-1}(G) \cup E(x, G)$ edges in $T_k(G_x)$. Suppose the vertex $x$ has at least one follower besides itself, $x$ should involve in at least one triangle which contains an edge in $H_{k-1}(G)$, i.e., $x \in V_{\triangle}(H_{k-1}(G), G)$ or $x \in H^+_{k-1}(G)$; otherwise, we cannot save any edges outside $T_k(G)$ into $T_k(G_x)$. If the anchor $x \in T_k(G)$, there is at least one edge in $E(x, H_{k-1}(G))$, which implies $x \in H^+_{k-1}(G)$. So we have $x \in H^+_{k-1}(G) \cup \{V_{\triangle}(H_{k-1}(G), G) \setminus T_k(G)\}$. Because $T_{k-1}(G) \setminus T_k(G) \subseteq H^+_{k-1}(G)$, we have $x \in H^+_{k-1}(G) \cup \{V_{\triangle}(H_{k-1}(G), G) \setminus T_{k-1}(G)\}$, i.e., $x \in \mathcal{L}$. ■

**Example 3.** *In Figure 4 with $k = 4$, we label every edge $e$ in $L^s_0$ with its layer index $l(e)$. Note that $\mathcal{L}$ also contains all the endpoints of the labeled edges. According to Theorem 4, only vertices in $T_3 \setminus T_4 = \{v_1, v_2, v_3, v_4, v_{10}, v_{11}\}$ and the anchor itself can be followers of an anchor. By Theorem 5, if a vertex $x$ has at least one follower besides $x$, $x$ comes from $\mathcal{L}$.*

### C. Efficiently Finding Candidate Followers

In this section, we define the candidate follower set for anchoring a vertex based on the *edge layers*. We prove that the candidate follower set is adequate for computing the followers of an anchor. We first introduce the concept of a *triangle hold path*, and show how to find the candidate followers of an anchor $x$, denoted by $CF(x)$.

If the vertex $x$ is anchored, every edge in $E(x, \mathcal{L})$ immediately becomes a candidate edge for $T_k(G_A)$. Note that an anchor edge $e \in E(x, G) \setminus E(x, \mathcal{L})$ cannot exist in $T_k(G_x)$.

**Definition 6.** ***Strong Triangle Hold***. *We say there is a strong triangle hold from an edge $e_1$ to another edge $e_2$ if there is a triangle $\{e_1, e_2, e_3\}$ in $\mathcal{L}$ such that $l(e_1) < l(e_2)$ and $l(e_1) < l(e_3)$.*

Note that $l(e)$ is the layer index of the edge $e \in \mathcal{L}$. The triangle hold is called strong because the edge $e_2$ can immediately be a candidate edge in $T_k(G_x)$, once $e_1$ became a candidate edge in $T_k(G_x)$.

**Definition 7.** ***Weak Triangle Hold***. *We say there is a weak triangle hold from an edge $e_1$ to another edge $e_2$ if there is a triangle $\{e_1, e_2, e_3\}$ in $\mathcal{L}$ such that $l(e_1) < l(e_2)$ and $l(e_1) = l(e_3)$.*

The triangle hold is called weak because the edge $e_2$ becomes a candidate edge in $T_k(G_x)$ if and only if $e_1$ and $e_3$ both became candidate edges in $T_k(G_x)$. In above two definitions, we also say $e_1$ *triangle holds* $e_2$ or $e_2$ is *triangle held* by $e_1$, if $e_2$ can become a candidate edge by $e_1$.

**Definition 8.** ***Triangle Hold Path***. *We say there is a triangle hold path from a given anchor vertex $x$ to a vertex $u \in H_{k-1}(G)$ (i.e., $x \rightsquigarrow u$) if there is an edge $e_x \in E(x, \mathcal{L})$, an edge $e_u \in E(u, \mathcal{L})$ and an edge set $E$ such that each edge in $E \cup \{e_x, e_u\}$ is triangle held by at least one edge in $E \cup \{e_x, e_u\}$.*

**Theorem 6.** *If a vertex $u \in H_{k-1}(G)$ is a follower of the anchor $x$, there is a triangle hold path $x \rightsquigarrow u$ if $x \neq u$.*

*Proof:* Let $E_x$ denote the anchor edges of $x$ in $\mathcal{L}$, i.e., $E_x = E(x, \mathcal{L})$. We can use Line 1 to 6 of Algorithm 3 to compute the anchored $k$-truss $T_k(G_x)$ as long as (i) $N := \mathcal{L} \cup T_{k-1}(G)$ at line 1; (ii) for each anchor edge $e_x \in E_x$, we only push $e_x$ into $P$ when $sup(e_x, N) < 1$ at Line 2 and 6; and (iii) all isolated vertices are deleted in $N$. Finally, $T_k(G_x)$ is exactly $N$. In the computation of $T_k(G)$ (without any anchors), all edges in $L^s_0$ are removed in Algorithm 3, while some of the edges may survive the computation of $T_k(G_x)$ (with anchoring $x$). The followers of $x$ are the vertices which are incident to a survived edge and do not exist in $T_k(G)$. Let $i$ denote the minimum layer index of an edge in $E_x$ (i.e, $i = min\{l(e) \mid e \in E_x\}$). In the computation of $T_k(G_x)$, we use the same edge accessing order in computing $T_k(G)$. For an edge $e_1 \in L^{i-1}_0$, when $e_1$ is accessed, the support of $e_1$ is less than $k-2$ because the support at current time is exactly the same as $e_1$ is accessed in the computation of $T_k(G)$. So all edges in $L^{i-1}_0$ are still deleted in computing $T_k(G_x)$. Then, when edges in $L^{i-1}_0$ have been deleted and no edges in $L_i$ have been deleted, for every edge $e_2 \in L_i \setminus E_x$, the support of $e_2$ is less than $k-2$ because the support at current time is the same as in the computation of $T_k(G)$. Consequently, all edges in $L^i_1 \setminus E_x$ cannot be candidate edges and are deleted. At this point, only edges which are triangle held by an edge in $E_x$ become candidate edges and clearly they have triangle hold paths. For a candidate edge $e_3$, only its triangle held edges in $L^s_{j+1}$ ($j = l(e_3)$) become candidate edges because $e_3$ cannot save other edges in computing $T_k(G_x)$, i.e., the supports of other edges cannot increase with the existence of $e_3$ when the edges are accessed. Consequently, the candidate spread from $x$ is strictly a top-down search through $E_x$'s triangle held edges and candidate edges' triangle held edges, which constitute triangle hold paths. For an edge $e_4$ without any triangle hold paths, when non-candidate edges in $L^{p-1}_0$ ($p = l(e_4)$) have

been deleted and no edge in $L_p$ has been deleted, the support of $e_4$ is less than $k-2$ because it is same as $e_4$ is accessed in computing $T_k(G)$. A follower of $x$ must be incident to at least one candidate edge in computing $T_k(G_x)$. So there is always a triangle hold path from $x$ to $u$ if $x \neq u$. ■

The candidate follower set $CF(x)$ consists of vertices where each vertex $u$ has at least one triangle hold path from $x$ and $u \in H_{k-1}(G)$. According to the proof of Theorem 6, we can generate $CF(x)$ by finding *candidate edges* (these edges may survive in $T_k(G_x)$) which is to iteratively activate the triangle held edges by candidate edges and anchor edges.

**Example 4.** *In Figure 4 with $k = 4$, we label every edge $e$ in $L_0^s$ with its layer index $l(e)$. If the vertex $v_2$ is anchored, all the edges in $E(v_2)$ become candidate edges. There is a strong triangle hold from $(v_2, v_1)$ to $(v_1, v_3)$, so the latter becomes a candidate edge. For the edge $(v_3, v_6)$, there is a triangle $\triangle_{v_2,v_3,v_6}$ with two weak triangle holds. Since $(v_2, v_3)$ and $(v_2, v_6)$ are already candidate edges, $(v_3, v_6)$ becomes a candidate edge. No other edges can become candidate edges and we only need to compute the anchored $k$-truss on the candidate edges with the original $k$-truss.*

### D. Finding Followers with Early Termination

To find the true followers of an anchor $x$, we need to conduct $k$-truss computation on the subgraph induced by $CF(x) \cup T_k(G) \cup \{x\}$. To further speed up the follower finding procedure, we introduce an early termination technique in generating $CF(x)$ to directly retrieve the true followers.

**Layer-by-Layer Search.** In the search of finding candidate edges, each edge has **three status**. We say an edge in $\mathcal{L}$ is **unexplored** if it has not been checked with the support constraint in our layer-by-layer traversal. An edge in $\mathcal{L}$ is **survived** if it has survived the support check, otherwise it becomes **discarded**. For a given anchor, a *discarded* edge will never be involved in the following computation, and a *survived* edge may become *discarded* later due to the deletion cascade. Note that some edges are *implicitly* marked as *discarded* since they are not involved in any triangle hold paths from $x$, and will never be explored.

For an edge $e \in \mathcal{L}$, the triangles containing $e$ can be divided into six disjoint sets $\triangle_{s,s}$, $\triangle_{s,u}$, $\triangle_{s,t}$, $\triangle_{u,u}$, $\triangle_{u,t}$, $\triangle_{t,t}$ where we use subscripts $s$, $u$ and $t$ to represent that an edge is *survived*, *unexplored* and in $T_k(G)$, respectively. For example, $\triangle_{s,u}$ represents the triangle set where each containing a *survived* edge, an *unexplored* edge and $e$. We use $s^+(e) = |\triangle_{s,s}| + |\triangle_{s,u}| + |\triangle_{s,t}| + |\triangle_{u,u}| + |\triangle_{u,t}| + |\triangle_{t,t}|$ to denote the upper bound of $sup(e, T_k(G_x))$. The following theorem indicates that we can safely exclude a candidate edge $e$ if its support upper bound is insufficient. The removal of an edge may invoke the deletion of other edges, where details are described in Algorithm 4. When the shrink function terminates, all of the edge supports and edges affected by the removal of $e$ will be correctly updated.

**Theorem 7.** *For an edge $e \in \mathcal{L}$, $e$ cannot exist in $T_k(G_x)$ if one of the following conditions is satisfied: (i) $e \in E(x)$ and $s^+(e) < 1$; (ii) $e \notin E(x)$ and $s^+(e) < k-2$.*

*Proof:* Since the edges in $G \setminus \{\mathcal{L} \cup T_k(G)\}$ cannot exist in $T_k(G_x)$, we only need to consider triangles in $\mathcal{L} \cup T_k(G)$ to compute the support upper bound of the edge $e$. Since *discarded* edges cannot provide triangle hold to any edge, the six disjoint triangle sets are adequate for computing $sup(e, T_k(G_x))$. So $s^+(e)$ is a correct upper bound of $sup(e, T_k(G_x))$. ■

**Example 5.** *In Figure 4 with $k = 4$, if the vertex $v_2$ is anchored, all the edges in $E(v_2)$ become candidate edges and is marked unexplored. Without the support check, $(v_1, v_3)$ can become a candidate edge. However, according to Theorem 7, $(v_1, v_3)$ cannot become a candidate edge and is marked discarded because $s^+((v_1, v_3)) = 1 < 2$. The same holds for $(v_3, v_6)$. Note that some edges are implicitly marked discarded such as $(v_1, v_4)$. Thus no edges outside $T_k(G)$ can survive in $T_k(G_{v_2})$.*

**Finding Followers.** Algorithm 5 lists the pseudo-code of the follower computation for a chosen anchor $x$. A min heap $H$ is used to keep the candidate edges, and the key of an edge $e$ is $l(e)$ with ties broken by the edges' IDs. In this way, we explore the candidate edges in a layer-by-layer fashion and it is easy to check whether an edge $e$ has been *explored* based on its ID and layer index $l(e)$. For each popped edge $e$, Line 6 computes its support upper bound $s^+(u)$. If $e$ survives the support check, $e$ will be set to *survived* (Line 8) and the edges *triangle held* by $e$ in lower layers (i.e., unexplored candidate edges) will be pushed into $H$ if they are not already in $H$ (Lines 9-10). Otherwise, $e$ is set to *discarded* and the early termination process is invoked (Lines 12-13). The deletion may be cascaded and some *survived* edges may be set to *discarded* during the process. When the algorithm terminates, the vertices which are incident to at least one *survived* edge are the followers of $x$. The time complexity of the algorithm is $\mathcal{O}(|\triangle|)$ because each triangle is accessed at most six times: to push candidate edges into $H$, compute upper bounds and compute the cascades of the deletion.

**Algorithm Correctness.** We show the deletion of the edges in Algorithm 5 has a *valid order* $O$ for the computation of $T_k(G_x)$. An edge $e$ may be *implicitly* deleted when $e$ never becomes a candidate edge, i.e., (1) no triangle hold paths (without early termination) exist for $e$; or (2) some edges on $e$'s triangle hold paths (without early termination) are *discarded*, thus no triangle hold paths exist when applying early termination. According to Theorem 6, we can safely discard such edge $e$. An edge $e$ may also be *explicitly* deleted in $O$ if (3) $u$ is set *discarded* at Line 12 because it fails the support check when it is popped; or (4) $s^+(u)$ becomes insufficient due to the deletions of the other edges (Line 6 of Algorithm 4). Because $s^+(u)$ is correctly computed (Line 6) and maintained (Algorithm 4), $e$ does not satisfy the support constraint when $e$ is deleted in cases (3) and (4). Let $M$ denote the remaining edges when Algorithm 5 terminates. As all of the edge in $\mathcal{L}$ have been *explored* explicitly or implicitly, we have $s^+(u) = sup(e, M \cup T_k(G)) \geq k-2$, for every vertex $e \in M$. Consequently, none of the edges in $M \cup T_k(G)$ can be discarded. As such, $O$ is a *valid order* and $T_k(G_x) = V(M) \cup M \cup T_k(G)$.

**Algorithm 4**: **ShrinkEdge**($e$)

**Input** : $e$ : the edge for support check
1 **for** each *survived* edge $e_0$ which forms a triangle with $e$ and a *non-discarded* edge **do**
2  $s^+(e_0) := s^+(e_0) - 1$;
3  $T \leftarrow v$ **If** $s^+(e_0) < 1$ and $e_0 \in E(x)$;
4  $T \leftarrow v$ **If** $s^+(e_0) < k-2$ and $e_0 \notin E(x)$;
5 **for** each $e_1 \in T$ **do**
6  $e_1$ is set *discarded*;
7  **ShrinkEdge**($e_1$);

**Algorithm 5**: **FindFollowers**($x$, $\mathcal{L}$)

**Input** : $x$ : the anchor; $\mathcal{L}$ : *edge layers*
**Output** : $F$ : the followers of $x$
1 $H := \emptyset$; $F := \emptyset$ //$H$ is a min heap, key is layer index;
2 **for** each $e \in E(x, \mathcal{L})$ **do**
3  $H.push(e)$ ;
4 **while** $H \neq \emptyset$ **do**
5  $e_0 \leftarrow H.pop()$;
6  Compute $s^+(e_0)$;
7  **if** $s^+(e_0) \geq k-2$ or ($s^+(e_0) \geq 1$ and $e \in E(x, \mathcal{L})$) **then**
8  $e_0$ is set *survived*;
9  **for** each $e_1$ *triangle held* by $e_0$ and $e_1 \notin H$ **do**
10  $H.push(e_1)$;
11  **else**
12  $e_0$ is set *discarded* ;
13  **ShrinkEdge**($e_0$);
14 **for** each *survived* edge $e$ **do**
15  $F := F \cup V(e)$ ;
16 **return** $F \setminus T_k(G)$

### E. The AKT Algorithm

In the procedure of finding the best anchor, we do not need to compute the followers of a vertex $u$ if $u$ is found to be a follower of another vertex $x$, because the followers of $u$ is always less than the followers of $x$ if $u \in \mathcal{F}(x)$. Suppose $u \in \mathcal{F}(x)$, $u$ will be preserved in $k$-truss if $x$ is anchored. For every vertex $v$ in $\mathcal{F}(u)$, $v$ will be preserved if $x$ is anchored, because $u$ will be preserved and anchoring $x$ cannot decrease the edge supports. So $\mathcal{F}(u) \subseteq \mathcal{F}(x)$. Since $x \in \mathcal{F}(x) \setminus \mathcal{F}(u)$, we have $\mathcal{F}(u) \subset \mathcal{F}(x)$. So all the followers produced during finding the best anchor can be excluded from candidate anchors.

Algorithm 6 illustrates the details of AKT which finds the best anchor for a graph $G$ (i.e., $b = 1$). Particularly, we firstly apply Algorithm 3 to compute the *edge layers* of $G$ (Line 1). Initially, the candidate anchor set $T$ is set to $\mathcal{L}$ according to Theorem 5. Then we sequentially access vertices in $T$ based on their degrees in $T_{k-1}(G)$ in decreasing order, and compute their followers by Algorithm 5. According to the follower-based pruning, Line 6 excludes the followers of current accessed vertex from $T$. We continuously maintain the current best anchor with the largest number of followers (denoted by $\lambda$) seen so far. We have the best anchor when the algorithm terminates.

To handle general cases where $b > 1$, our AKT algorithm can easily fit within the greedy algorithm (Replacing Lines 3-4 of Algorithm 2) to find the best vertex in each iteration. The only difference is that we need to enforce that the support constraint for each anchor edge in previous iterations to be only 1 and $\mathcal{L}$ is computed from anchored ($k$-1)-truss. Note that in order to avoid computing $T_{k-1}(G_A)$ (Line 1 of Algorithm 3) from scratch in each iteration, we firstly maintain $C_{k-2}(G_A)$ since it should be a supergraph of $T_{k-1}(G_A)$. Moreover, if the ($k$-1)-truss consists of a set of disconnected subgraphs, we can avoid the re-computation of the followers of a subgraph in the next iteration unless there is a new anchor in this subgraph. The time complexity of the algorithm remains $\mathcal{O}(bn \times m^{3/2})$. Nevertheless, our empirical study shows we can significantly improve performance of the straightforward implementation (Algorithm 2) by orders of magnitude, due to the much smaller number of candidate anchors and the much more efficient follower computation.

**Algorithm 6**: **AKT**($G$, $k$)

**Input** : $G$ : a social network, $k$ : support constraint
**Output** : the best anchor vertex
1 Compute *edge layers* $\mathcal{L}$ (Algorithm 3);
2 $T \leftarrow \mathcal{L}$ (Theorem 5); $\lambda := 0$;
3 **for** each vertex $x \in T$ with decreasing order of $deg(x, T_{k-1}(G))$ **do**
4  $\mathcal{F}(x) \leftarrow$ **FindFollowers**($x$, $\mathcal{L}$) (Algorithm 5);
5  **if** $\mathcal{F}(x) \neq \emptyset$ **then**
6  $T := T \setminus \mathcal{F}(x)$;
7  **if** $|\mathcal{F}(x)| > \lambda$ **then**
8  $\lambda := |\mathcal{F}(x)|$;
9 **return** the best anchor

**Algorithm Correctness.** (1) For the anchored $k$-truss problem with $b = 1$ on graph $G$, we get the correct result immediately based on the correctness of proposed techniques. (2) Assume the algorithm is correct when $b = i$, $i \in N^+$ and returns the anchor set $A$. (3) Consider the problem with $b = i + 1$, now the $k$-truss of $G$ is $T_k(G_A)$ since we have $sup(e, G_A)$ satisfies support constraint for any $e \in T_k(G_A)$ and $T_k(G_A)$ is maximal. Then the ($k$-1)-truss is updated correctly by maintaining the $C_{k-2}(G_A)$. Thus, we get the updated *edge layers* $\mathcal{L}$ correctly by Algorithm 3. Since all the techniques are based on $\mathcal{L}$, after running AKT on $G$ with $b = 1$ again, we get the correct result $A \cup \{x\}$ for the case of $b = i + 1$ on $G$. Note that in the ($k$-1)-truss $N$ of $G$, for every connected subgraph $S$ with $S \subset N$ and $x \notin S$, $S$ remains the same after anchoring $x$, thus, the previous result of anchoring any vertex in $S$ can be reused.

### F. Directed Graphs

We show that AKT can be applied to solve the anchored $k$-truss problem on directed graphs. Firstly, the truss on a directed graph can be defined as either cycle truss or flow truss [25]. For each model, the support of an edge in the graph is the number of cycle triangles or flow triangles which contain the edge. Then AKT can be directly applied on the directed graph where the only difference is that the support value is updated based on the dynamic of cycle triangles or flow triangles. In other words, a directed graph is regarded as an undirected graph where the triangles are defined as the cycle triangles or flow triangles in the directed graph.

TABLE II
STATISTICS OF DATASETS

| Dataset | Nodes | Edges | $d_{avg}$ | $k_{max}$ |
|---|---|---|---|---|
| Facebook | 4,039 | 88,234 | 43.7 | 97 |
| Brightkite | 58,228 | 194,090 | 6.7 | 42 |
| Gowalla | 196,591 | 456,830 | 4.7 | 23 |
| Amazon | 334,863 | 925,872 | 5.5 | 7 |
| Yelp | 552,339 | 1,781,908 | 6.5 | 73 |
| YouTube | 1,134,890 | 2,987,624 | 5.3 | 19 |
| DBLP | 1,566,919 | 6,461,300 | 8.3 | 119 |
| Pokec | 1,632,803 | 8,320,605 | 10.2 | 20 |
| LiveJournal | 3,997,962 | 34,681,189 | 17.4 | 352 |
| Orkut | 3,072,441 | 117,185,083 | 76.3 | 78 |

## IV. EXPERIMENTAL EVALUATION

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

### A. Experimental Setting

**Datasets.** Ten real-life networks are deployed in our experiments and we assume all vertices in each network are initially engaged. The original data of Yelp is downloaded from https://www.yelp.com.au/dataset_challenge, DBLP is from http://dblp.uni-trier.de/ and the others are from http://snap.stanford.edu/. In DBLP, we consider each author as a vertex and there is an edge for a pair of authors if they have at least one co-authored paper. In other datasets, there are existing vertices and edges. Table II shows the statistics of the 10 datasets, listed in increasing order of their edge numbers.

**Algorithms.** To the best of our knowledge, no existing work investigates the anchored $k$-truss problem. Towards the effectiveness, we tested 7 algorithms (Rand, Rand+, Deg, Sup, Exact, OLAK and AKT) to produce different $b$ anchors to show the number of followers, statistical results and case studies. We also implement and evaluate the algorithms to assess our techniques incrementally, from a naive algorithm (Naive) through to the final advanced algorithm (AKT). One baseline algorithm (BaselineM) based on truss maintenance is also evaluated. Table III shows the summary for algorithms.

**Parameters.** We conduct experiments under different settings by varying the support constraint $k$ and the budget for the anchors $b$. According to the characteristics of different datasets, the default values of $k$ are 6 for Amazon, 40 for Orkut and 15 for other datasets, respectively. The default value of $b$ is 20.

All programs are implemented in standard C++ and compiled with G++ in Linux. All experiments are performed on Intel Xeon 2.3GHz CPU and Redhat Linux System. The running time is set as INF if it exceeds $10^5$ seconds.

### B. Effectiveness of AKT

We conducted a series of experiments to evaluate the number of followers and some statistical results. Case studies are also depicted to show real-world examples.

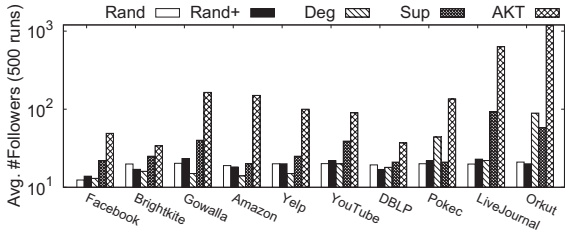
**Comparing Follower Numbers.** Figure 5 compares the number of followers w.r.t 20 anchors identified by AKT with that of two random approaches (Rand, Rand+), one degree based approach (Deg) and one triangle number based approach (Sup), with default $k$. We report the average number of followers for 500 independent tests in two random methods. The resulting numbers for other methods are always unique because we choose the anchors with the largest degree in Deg, with largest vertex support in Sup and find the best anchors in AKT. In Figure 5(a), the numbers of followers from Rand and Rand+ are always near 20, because there are usually no followers (except the anchors) and the anchors outside of $k$-truss are also regarded as followers. Rand+ does not necessarily have more followers than Rand because the chosen anchors by Rand+ are more likely from the $k$-truss and not counted as followers. In Figure 5, Deg and Sup basically outperform Rand and Rand+, but they are still significantly outperformed by AKT. Sup is beaten by Rand+ in Pokec because a vertex involved in large number of triangles does not necessarily have a large number of followers. Deg may also fail in some cases such as in LiveJournal. Figures 5(b)-(e) show the numbers of followers achieved by AKT are significantly larger than other algorithms when $k$ and $b$ vary.

TABLE III
SUMMARY OF ALGORITHMS

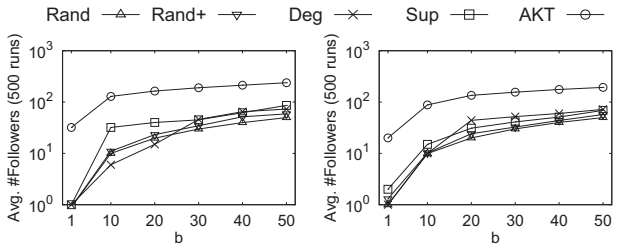
| Algorithm | Description |
|---|---|
| Rand | randomly chooses $b$ anchors from $G$ |
| Rand+ | randomly chooses $b$ anchors from $\mathcal{L}$ |
| Deg | chooses the $b$ vertices from $\mathcal{L}$, with the largest degrees in $\mathcal{L}$, as the anchors |
| Sup | chooses the $b$ vertices from $\mathcal{L}$, with the largest vertex supports in $\mathcal{L}$, as the anchors. The vertex support of $u$ in $\mathcal{L}$ is defined by the number of containing-$u$-triangles in $\mathcal{L}$ |
| Exact | identifies the optimal solution by exhaustively searching all possible combinations of $b$ anchors by Algorithm 5 |
| OLAK | the algorithm in [30] to find $b$ best anchors for the anchored $k$-core problem |
| Naive | computes a $k$-truss on $G$ for each candidate anchor $u \in G$ to find the best anchor in each iteration of Algorithm 2 |
| BaselineT | computes a $k$-truss on $G$ for each candidate anchor $x$ in $\mathcal{L}$ (Theorem 5) |
| BaselineM | applies the state-of-the-art truss maintenance algorithm [34] to compute followers for each candidate anchor in BaselineT |
| BLT+C | computes a $k$-truss on $\{x\} \cup T_{k-1}(G)$ (Theorem 4) for each candidate anchor $x$ in BaselineT |
| AKT | finds followers from $CF(x)$ through layer-by-layer search on $\mathcal{L}$ (Theorem 6 and 7) for each candidate anchor $x$ in BaselineT, i.e., arrives at Algorithm 6 |

We also compare the performance of AKT with Exact, which identifies the optimal solution by exhaustively searching two relatively small datasets. Figure 6 shows that the numbers of followers from AKT are comparable with Exact. We can see that Exact is cost-prohibitive and AKT is very efficient.

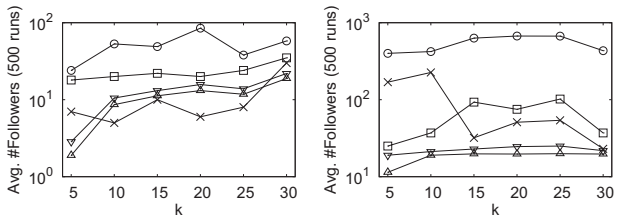
**Comparing $k$-Core and $k$-Truss.** In Figure 7, we report the global clustering coefficient and modularity values on the induced subgraphs of some representative vertices in each DBLP dataset with different time stamp. In DBLP with Year X, an author exists if he/she has one or more publications before Year X, and each edge between two authors represents there is one or more publications before Year X from the two




(a) Number of Followers on Different Datasets

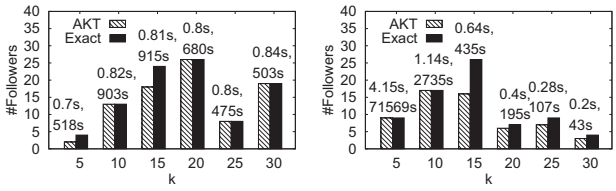



(b) Gowalla, k=15 (c) Pokec, k=15




(d) Facebook, b=20 (e) LiveJournal, b=20

Fig. 5. Number of Followers

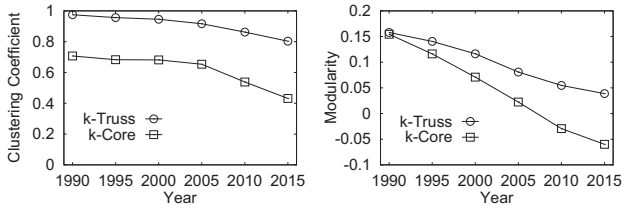



(a) Facebook, b=2 (b) Brightkite, b=2

Fig. 6. Greedy vs Exact

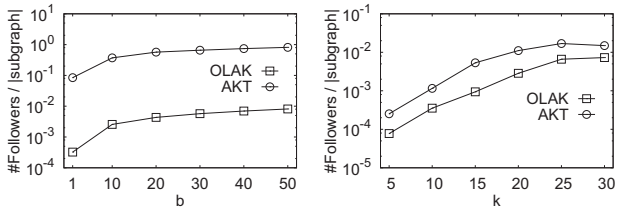
authors. To fairly compare the two models, we find a vertex set consisting of 10% vertices in the graph which have the largest core numbers for $k$-core model (resp. truss numbers for $k$-truss model). These two set of vertices are the user groups with high user engagement according to $k$-core and $k$-truss models, respectively. The figures show both clustering coefficient and modularity are significantly higher on the vertices from truss. The margins become larger as time goes by, which indicates $k$-truss becomes more effective on current social networks.

**Comparing Anchoring Gain.** In Figure 8, we report the anchoring gain in the result of `OLAK` for the anchored $k$-core problem and the result of the proposed `AKT` algorithm. For a given budget $b$, we consider the increase percentage of the original non-anchored subgraph as the engagement gain, i.e., the number of followers 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 value of $k$ in Figure 8 is the input value for `AKT`, and the input for `OLAK` is $k-1$ correspondingly.




(a) Top 10% Vertices in DBLP (b) Top 10% Vertices in DBLP

Fig. 7. Comparing Core and Truss




(a) Pokec, k=15 (b) Orkut, b=20

Fig. 8. Engagement Gain of Anchoring

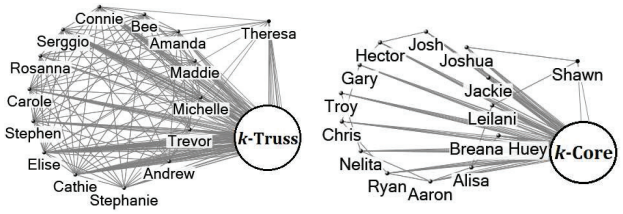
We can see the anchored $k$-truss can better enlarge the original subgraph under the same anchoring budget, e.g., when $b = 20$, the anchored 15-truss by `AKT` on `Orkut` has 1184 followers while the anchored 14-core by `OLAK` only has 217 followers. It shows our anchored $k$-truss model effectively increase the volume of social communities with a limited anchoring budget.

**Case Studies.** Figure 9(a) depicts the anchored $k$-truss in the result of `AKT` on `Yelp` with $k = 10$ and $b = 1$. When the best anchor "Theresa" alone is anchored, there are 15 followers (including "Theresa"). It is interesting that only 7 of them are neighbors of "Theresa", and the others are supported indirectly. We can see that the anchors and followers saved in the anchored $k$-truss have strong connections to the original $k$-truss, and among themselves, while they are excluded by the network unraveling. Figure 9(b) shows the result of `OLAK` for the anchored $k$-core problem on `Yelp` with $k = 9$ and $b = 1$. "Shawn" is the best anchor returned by `OLAK`. We can see the connection among followers is fairly weak, which means the anchored $k$-core preserves some relatively unstable users compared with the anchored $k$-truss model. The followers in Figure 9(a) themselves form a cohesive sub-group which is worthwhile and reasonable to be preserved by the anchor. Besides, since the $k$-truss represents tighter social groups, the $k$-truss and anchored $k$-truss present better effectiveness on modeling social communities and the anchoring, respectively.

### C. *Efficiency of* `AKT`

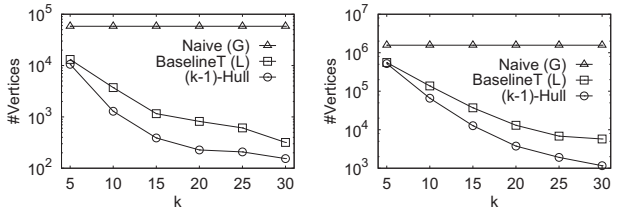
We conduct comprehensive experiments to evaluate the efficiency of proposed techniques and algorithms.

**Reducing Candidate Anchors.** Figure 10 reports the sizes of $G$, *edge layers* $\mathcal{L}$ and ($k$-1)-hull $H_{k-1}(G)$ on two networks `Brightkite` and `DBLP`. Recall that `Naive` needs to check all vertices in $G$ because anchoring vertices in $T_k(G)$ can also have followers, and the other algorithms (`BaselineT`, `BLT+C` and `AKT`) only consider the vertices from $\mathcal{L}$ as candidate anchors (Theorem 5). We also report the size of $H_{k-1}(G)$, which bounds the size of the candidate followers




(a) Anchored $k$-truss, k=10, b=1 (b) Anchored $k$-core, k=9, b=1

Fig. 9. Case Studies on Yelp




(a) Brightkite, b=1 (b) DBLP, b=1

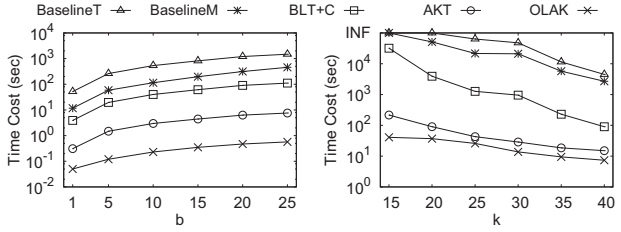
Fig. 10. Reducing Candidate Anchors and Followers

by Theorem 4. As expected, the sizes of $\mathcal{L}$ and $H_{k-1}(G)$ drop with the growth of $k$. The candidate size reductions from $G$ to $\mathcal{L}$ and $H_{k-1}(G)$ are significant for every $k$ value, which validates the effectiveness of Theorem 4 and 5.

**Pruning Candidate Followers.** Figure 11 demonstrates the effectiveness of the pruning techniques which help us to further eliminate non-promising candidate followers which cannot exist in the anchored $k$-truss. Four algorithms are evaluated by the running time on four networks with different $b$ or $k$. We report that each pruning technique significantly reduces the running time, especially the triangle hold path based layer-by-layer search in `AKT` (Theorem 6 and 7).

**Effect of $k$ and $b$.** Figure 11 also studies the impact of $k$ and $b$ on five algorithms. `AKT` significantly outperforms the two baseline algorithms under all settings. We terminate the algorithms when the running time exceeds $10^5$ seconds. We observe that the size of *edge layers* has a great impact on the running time of `AKT`, which makes the margin between `BLT+C` and `AKT` vary from about 5 times to 100 times. The `OLAK` algorithm outperforms `AKT` due to the computation efficiency of $k$-core. However, with the consideration of tie strength, the $k$-truss and anchored $k$-truss model show superior effectiveness on modeling social communities and their anchoring.

**Different Datasets.** Figure 12 reports the performance of four algorithms on 10 networks. The datasets are ordered by the number of edges. Not surprisingly, the performance of `BaselineT` is poor and can only finish computation on 3 networks within $10^5$ seconds. The running time is not strictly increasing with the order of datasets because the truss number distribution is quite different on all datasets. We can see that although `BaselineM` outperforms `BaselineT`, it is still significantly beaten by `AKT`. `OLAK` is faster than `AKT` by utilizing the $k$-core model while ignores the tie strength in modeling social communities and the anchoring action. The margin between `OLAK` and `AKT` is similar to the margin between $k$-core and $k$-truss computations. It further validates the effectiveness of proposed techniques.




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

Fig. 11. Running Time with Different $k$ and $b$

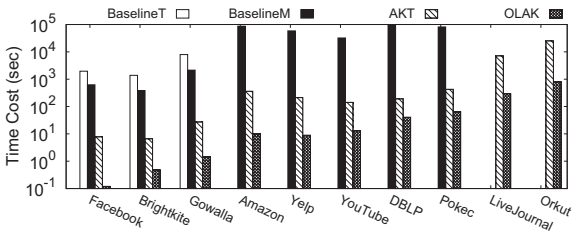



Fig. 12. Running Time on Different Datasets

## V. Related Work

Various cohesive subgraph models are proposed in the literature to accommodate different scenarios, such as clique [16], $k$-plex [21], $k$-core [20] and $k$-truss [6], and so on. Among these subgraph models, $k$-core and $k$-truss are the widely studied models with polynomial computation time.

Seidman [20] proposes the $k$-core model which is a maximal subgraph where each vertex has at least $k$ neighbors in the subgraph. The $k$-core has a wide spectrum of applications such as social contagion [26], community detection [32], event detection [18], network defence [31] and so on. The degeneration property of $k$-core can be used to quantify engagement dynamics in social networks [17]. Considering only the user engagement, Bhawalkar *et* al. [4] propose the problem of anchored $k$-core to prevent network unraveling. Zhang *et* al. [30] present an efficient algorithm to solve the anchored $k$-core problem by utilizing the vertex deletion order in $k$-core computation. However, the techniques in [30] cannot be applied to solve the anchored $k$-truss problem because the $k$-truss computation is inherently based on edges and triangles. Besides, $k$-core treats each edge equally and ignores the fact that the social ties have different strength and influence.

Further considering the strength of ties, Cohen [6] proposes the model of $k$-truss which is a maximal subgraph where every edge exists in at least $k-2$ triangles in the subgraph. There are significant studies on the model of $k$-truss while none of them utilizes $k$-truss to reinforce network structure. Wang and Cheng [27] present the in-memory truss decomposition algorithm with a time complexity to $O(m^{3/2})$ and study the I/O efficient truss decomposition. Huang *et* al. [14] extend the model of $k$-truss to probabilistic graphs. Shao *et* al. [23] study the k-truss detection problem on distributed systems and propose an efficient parallel algorithm. Zhao and Tung [33] use the $k$-truss to capture the cohesion in social interactions and propose a visualization system based on $k$-truss. Huang *et* al. [12] study the $k$-truss based community

model which further requires edge connectivity inside the community. Recently, Akbas and Zhao [1] propose a truss-equivalence based index to speed up the search of the truss based community. Huang and Lakshmanan [13] study the attributed $k$-truss community search where the largest attribute relevance score is satisfied. For the fact that $k$-truss is an enhanced version of ($k$-1)-core, the $k$-truss model not only ensures the strong tie strength among users but also captures users with high engagement inside the community.

Evaluating the activity of user engagement is crucial for social networking sites to improve user stickiness and avoid the collapse of network. Garcia *e*t al. [8] study the decline of Friendster, which was popular at early 21st century. Seki and Nakamura [22] explain that the mechanism in the collapse of Friendster by use of an individual-level model. Ugander *e*t al. [26] emphasize that the neighborhood structure hypothesis has formed the underpinning of essentially all current social contagion models. They find that the contagion is tightly controlled by the number of friends in current subgraph, like $k$-core or $k$-truss, rather than by the actual number of friends in the graph. Tie strength, introduced by Granovetter [11], is a fundamental social network characteristic and has been well studied in sociology communities. Recently, many social network researchers show that the strength of ties is a tenable theory on social networks nowadays, such as Facebook and Twitter [3], [10], [33]. Most existing methods for strong tie detection are based on structural information, especially on triangles, following the ideas from sociology [7], [9], [19], [24]. Aral and Walker [2] show that high edge embeddedness increases user influence in a large scale experiment.

## VI. Conclusion

In this paper, we propose and investigate the problem of anchored $k$-truss to reinforce social networks considering both user engagement and tie strength. The problem aims to *anchor* a set of vertices in a network such that the size of the resulting $k$-truss is maximized. We prove the problem is NP-hard when $k \geq 4$. Then we develop an efficient greedy algorithm, named `AKT`, based on the *edge layer* structure which divides a small set of edges by layers. Extensive experiments on 10 real-life networks demonstrate the demonstrate the effectiveness of our model and the efficiency of our methods.

## Acknowledgments

Fan Zhang is supported by Huawei YBN2017100007. Ying Zhang is supported by ARC FT170100128 and D-P180103096. Lu Qin is supported by ARC DP160101513. Wenjie Zhang is supported by ARC DP180103096 and Huawei YBN2017100007. Xuemin Lin is supported by NSFC 61672235, ARC DP170101628, DP180103096 and Huawei YBN2017100007.

## References

[1] E. Akbas and P. Zhao. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB*, 10(11):1298–1309, 2017.
[2] S. Aral and D. Walker. Tie strength, embeddedness, and social influence: A large-scale networked experiment. *Management Science*, 60(6):1352–1370, 2014.
[3] E. Bakshy, I. Rosenn, C. Marlow, and L. A. Adamic. The role of social networks in information diffusion. In *WWW*, pages 519–528, 2012.
[4] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.
[5] R. H. Chitnis, F. V. Fomin, and P. A. Golovach. Preventing unraveling in social networks gets harder. In *AAAI*, 2013.
[6] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, page 16, 2008.
[7] N. Eagle, A. S. Pentland, and D. Lazer. Inferring friendship network structure by using mobile phone data. *PNAS*, 106(36):15274–15278, 2009.
[8] D. Garcia, P. Mavrodiev, and F. Schweitzer. Social resilience in online communities: the autopsy of friendster. In *COSN*, pages 39–50, 2013.
[9] E. Gilbert and K. Karahalios. Predicting tie strength with social media. In *SIGCHI*, pages 211–220, 2009.
[10] P. A. Grabowicz, J. J. Ramasco, E. Moro, J. M. Pujol, and V. M. Eguíluz. Social features of online networks: the strength of weak ties in online social media. *CoRR*, abs/1107.4009, 2011.
[11] M. S. Granovetter. The strength of weak ties. *American journal of sociology*, 78(6):1360–1380, 1973.
[12] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.
[13] X. Huang and L. V. S. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.
[14] X. Huang, W. Lu, and L. V. S. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *SIGMOD*, pages 77–90, 2016.
[15] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.
[16] R. D. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.
[17] F. D. Malliaros and M. Vazirgiannis. To stay or not to stay: modeling engagement dynamics in social graphs. In *CIKM*, pages 469–478, 2013.
[18] P. Meladianos, G. Nikolentzos, F. Rousseau, Y. Stavrakas, and M. Vazirgiannis. Degeneracy-based real-time sub-event detection in twitter stream. In *ICWSM*, pages 248–257, 2015.
[19] R. Rotabi, K. Kamath, J. M. Kleinberg, and A. Sharma. Detecting strong ties using network motifs. In *WWW*, pages 983–992, 2017.
[20] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.
[21] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.
[22] K. Seki and M. Nakamura. The collapse of the friendster network started from the center of the core. In *ASONAM*, pages 477–484, 2016.
[23] Y. Shao, L. Chen, and B. Cui. Efficient cohesive subgraphs detection in parallel. In *SIGMOD*, pages 613–624, 2014.
[24] S. Sintos and P. Tsaparas. Using strong triadic closure to characterize ties in social networks. In *SIGKDD*, pages 1466–1475, 2014.
[25] T. Takaguchi and Y. Yoshida. Cycle and flow trusses in directed networks. *Royal Society open science*, 3(11):160270, 2016.
[26] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *PNAS*, 109(16):5962–5966, 2012.
[27] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.
[28] X. Wang, R. Donaldson, C. Nell, P. Gorniak, M. Ester, and J. Bu. Recommending groups to users using user-group engagement and time-dependent matrix factorization. In *AAAI*, 2016.
[29] S. Wu, A. D. Sarma, A. Fabrikant, S. Lattanzi, and A. Tomkins. Arrival and departure dynamics in social networks. In *WSDM*, pages 233–242, 2013.
[30] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin. OLAK: an efficient algorithm to prevent unraveling in social networks. *PVLDB*, 10(6):649–660, 2017.
[31] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *AAAI*, pages 245–251, 2017.
[32] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: Efficient (k, r)-core computation on social networks. *PVLDB*, 10(10):998–1009, 2017.
[33] F. Zhao and A. K. H. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB*, 6(2):85–96, 2012.
[34] R. Zhou, C. Liu, J. X. Yu, W. Liang, and Y. Zhang. Efficient truss maintenance in evolving networks. *CoRR*, abs/1402.2807, 2014.