Pivotal Relationship Identification: The K-Truss Minimization Problem

Weijie Zhu[1,3] , **Mengqi Zhang**[2] , **Chen Chen**[1*] , **Xiaoyang Wang**[2] , **Fan Zhang**[4] , **Xuemin Lin**[1,3,4]
[1]East China Normal University, China
[2]Zhejiang Gongshang University, China
[3]Zhejiang Lab, Hangzhou, China
[4]The University of New South Wales, Australia
{weijie.zhu93, mengqiz.zjgsu, fanzhang.cs}@gmail.com, {chenc, xiaoyangw}@zjgsu.edu.cn,
lxue@cse.unsw.edu.au

Abstract

In a social network, the strength of relationships between users can significantly affect the stability of the network. In this paper, we use the k-truss model to measure the stability of a social network. To identify critical connections, we propose a novel problem, named k-truss minimization. Given a social network G and a budget b, it aims to find b edges for deletion which can lead to the maximum number of edge breaks in the k-truss of G. We show that the problem is NP-hard. To accelerate the computation, novel pruning rules are developed to reduce the candidate size. In addition, we propose an upper bound based strategy to further reduce the searching space. Comprehensive experiments are conducted over real social networks to demonstrate the efficiency and effectiveness of the proposed techniques.

1 Introduction

As a key problem in graph theory and social network analysis, the mining of cohesive subgraphs, such as k-core, k-truss, clique, etc, has found many important applications in real life [Cohen, 2008; Tsourakakis *et al.*, 2013; Wen *et al.*, 2016; Yu *et al.*, 2013]. The mined cohesive subgraph can serve as an important metric to evaluate the properties of a network, such as network engagement. In this paper, we use the k-truss model to measure the cohesiveness of a social network. Unlike k-core, k-truss not only emphasizes the users' engaged activities (*i.e.,* number of friends), but also requires strong connections among users. That is, the k-truss of G is the maximal subgraph where each edge is involved in at least $k-2$ triangles. Note that triangle is an important building block for the analysis of social network structure [Xiao *et al.*, 2017; Cui *et al.*, 2018]. Thus the number of edges in the k-truss can be utilized to measure the stability of network structure.

The breakdown of a strong connection may affect other relationships, which can make certain relationships involved in less than $k-2$ triangles and removed from the k-truss. Hence, it will lead to a cascading breakdown of relationships eventually. To identify the critical edges, in this paper, we investigate the k-truss minimization problem. Given a social network G and a budget b, k-truss minimization aims to find a set B of b edges, which will result in the largest number of edge breaks in the k-truss by deleting B.

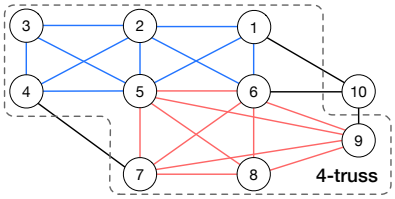


Figure 1: Motivation Example

Figure 1 is a toy social network with 10 users. Suppose k is 4. Then only the blue and red edges belong to the 4-truss. If we delete edge (v_2, v_5), it will affect the connections among other users and lead to the removal of all the blue edges from the 4-truss, since they no longer meet the requirement of 4-truss. We can see that the deletion of one single edge can seriously collapse the social network. The k-truss minimization problem can find many applications in real life. For instance, given a social network, we can reinforce the community by paying more attention to the critical relationships. Also, we can strengthen the important connections to enhance the stability of a communication network or detect vital connections in enemy's network for military purpose.

The main challenges of this problem lie in the following two aspects. Firstly, we prove that the problem is NP-hard. It means that it is non-trivial to obtain the result in polynomial time. Secondly, the number of edges in a social network is usually quite large. Even if we only need to consider the edges in k-truss as candidates, it is still a large amount of edges to explore. To the best of our knowledge, we are the first to investigate the k-truss minimization problem through edge deletion. We formally define the problem and prove its hardness. Novel pruning rules are developed to reduce the searching space. To further speed up the computation, an upper bound based strategy is proposed.

2 Preliminaries

2.1 Problem Definition

We consider a social network G as an undirected graph. Given a subgraph $S \subseteq G$, we use V_S (resp. E_S) to denote

*Corresponding author

the set of nodes (resp. edges) in S. $N(u, S)$ is the neighbors of u in S. $deg(u, S)$ equals $|N(u, S)|$, denoting the degree of u in S. $m = |E_G|$ is the number of edges in G. Assuming the length of each edge equals 1, a triangle is a cycle of length 3 in the graph. For $e \in E_G$, a **containing-e-triangle** is a triangle which contains e.

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

Definition 2 (**edge support**). *Given a subgraph $S \subseteq G$ and an edge $e \in E_S$, the edge support of e is the number of containing-e-triangles in S, denoted as $sup(e, S)$.*

Definition 3 (k**-truss**). *Given a graph G, a subgraph S is the k-truss of G, denoted by T_k, if (i) $sup(e, S) \geq k - 2$ for every edge $e \in E_S$; (ii) S is maximal, i.e., any supergraph of S cannot be a k-truss; and (iii) S is non-trivial, i.e., no isolated node in S.*

Definition 4 (**trussness**). *The trussness of an edge $e \in E_G$, denoted as $\tau(e)$, is the largest integer k that satisfies $e \in E_{T_k}$ and $e \notin E_{T_{k+1}}$.*

Based on the definitions of k-core and k-truss, we can see that k-truss not only requires sufficient number of neighbors, but also has strict constraint over the strength of edges. A k-truss is at least a (k-1)-core. Therefore, to compute the k-truss, we can first compute the (k-1)-core and then find the k-truss over (k-1)-core by iteratively removing all the edges that violate the k-truss constraint. The time complexity is $O(m^{1.5})$ [Wang and Cheng, 2012]. Given a set B of edges in G, we use T_k^B to denote the k-truss after deleting B. We use $|T_k^B|$ to denote the number of edges in T_k^B. We define the followers $F(B, T_k)$ of B as the edges that are removed from T_k due to the deletion of B. Then our problem can be formally defined as follows.

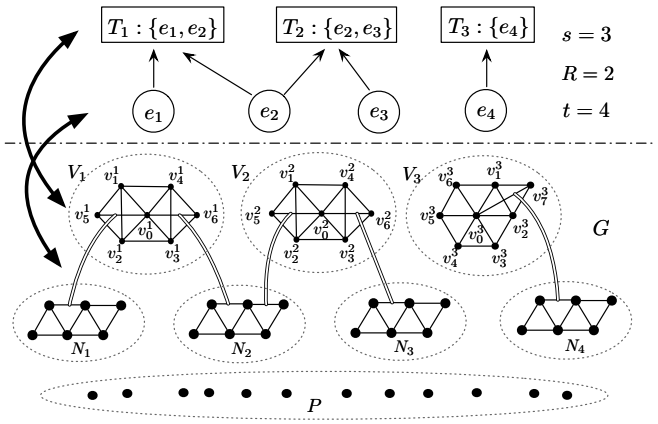
Given a graph G and a budget b, the k**-truss minimization problem** aims to find a set B^* of b edges, such that the $|T_k^{B^*}|$ is minimized. It is also equivalent to finding an edge set B^* that can maximize $|F(B^*, T_k)|$, *i.e.,*

$$B^* = \underset{B \subseteq E_G \wedge |B| = b}{\arg\max} |F(B, T_k)|.$$

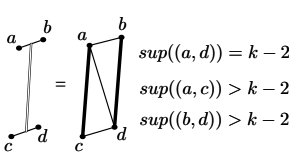
According to Theorem 1 and 2, the k-truss minimization problem is NP-hard for $k \geq 5$, and the objective function is monotonic but not submodular.

Theorem 1. *For $k \geq 5$, the k-truss minimization problem is NP-hard.*

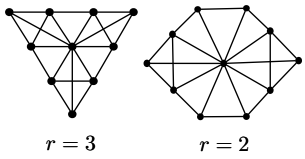
Proof. For $k \geq 5$, we sketch the proof for $k = 5$. A similar construction can be applied for the case of $k > 5$. When $k = 5$, we reduce the k-truss minimization problem from the maximum coverage problem [Karp, 1972], which aims to find b sets to cover the largest number of elements, where b is a given budget. We consider an instance of maximum coverage problem with s sets T_1, T_2, .., T_s and t elements $\{e_1, .., e_t\} = \cup_{1 \leq i \leq s} T_i$. We assume that the maximum number of elements inside T is $R \leq t$. Then we construct a corresponding instance of the k-truss minimization problem



(a) Constructed Example for NP-hard Proof



(b) Structure Illustration



(c) Construction of V for $R = 3$

Figure 2: Example for NP-hard

in a graph G as follows. Figure 2(a) is a constructed example for $s = 3, t = 4, R = 2$.

We divide G into three parts, V, N and P. 1) V consists of s parts. Each part V_i corresponds to T_i in the maximum coverage problem instance. 2) N consists of t parts. Each part N_i corresponds to e_i in the maximum coverage problem instance. 3) P is a dense subgraph. The support of edges in P is no less than $k - 2 + b$. Specifically, suppose T_i consists of $r_i \leq R$ elements, V_i consists of $4R - r_i + 1$ nodes and $8R - r_i$ edges. To construct V_i, we first construct a $(4R - 2r_i)$-polygon. Then, we add a node v_0^i in the center of $(4R - 2r_i)$-polygon and add $4R - 2r_i$ edges between v_0^i and $v_1^i, ..., v_{4R-2r_i}^i$. Finally, we further add r_i nodes $v_{4R-2r_i+1}^i, ..., v_{4R-r_i}^i$ and $3r_i$ edges $\{(v_0^i, v_{4R-2r_i+1}^i), (v_1^i, v_{4R-2r_i+1}^i), (v_2^i, v_{4R-2r_i+1}^i), ..., (v_0^i, v_{4R-r_i}^i), (v_{2r_i-1}^i, v_{4R-r_i}^i), (v_{2r_i}^i, v_{4R-r_i}^i)\}$. With the construction, the edges in V have support no larger than 3. We use P to provide support for edges in V and make the support of edges in V to be 3. Each part in N consists of $2R + 2$ nodes and the structure is a list of $4R$ triangles which is shown in Figure 2(a). For each element e_i in T_j, we add two triangles between N_i and V_j to make them triangle connected. The structure is shown in 2(b). Note that each edge in N_i and V_j can be used at most once. We can see that edges in N have support no larger than 3. Finally, we use P to provide support for edges in N and make the support of edges in N to be 3. Then the construction is completed. The construction of V_i for $R = 3$ is shown in Figure 2(c).

With the construction, we can guarantee that 1) deleting any edge in V_i can make all the edges in V_i and the edges in N_j who have connections with V_i deleted from the truss.

Algorithm 1: Baseline Algorithm

Input : G: a social network, k: truss constraint, b: the budget
Output: B: the set of deleted edges

```
1  B ← ∅ ; T_k ← k-truss of G;
2  while |B| < b do
3      e* ← arg max_{e∈E_{T_k}} |F(e, T_k)| ;
4      delete e* from T_k and update T_k ;
5      B ← B ∪ {e*} ;
6  end while
7  return B
```

2) Only the edges in V_i can be considered as candidates. 3) Except the followers in N, each V_i has the same number of followers. In Figure 2(a), deletion of each V_i can make $8R$ edges (except the edges in N) removed. Consequently, the optimal solution of k-truss minimization problem is the same as the maximum coverage problem. Since the maximum coverage problem is NP-hard, the theorem holds. □

Theorem 2. *The objective function $f(x) = |F(x, T_k)|$ is monotonic but not submodular.*

Proof. Suppose $B \subseteq B'$. For every edge e in $F(B, T_k)$, e will be deleted from the k-truss when deleting B'. Thus $f(B) \leq f(B')$ and f is monotonic. Given two sets A and B, if f is submodular, it must hold that $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$. We show that the inequality does not hold by constructing a counter example. In Figure 1, for $k = 4$, suppose $A = \{(v_5, v_6)\}$ and $B = \{(v_7, v_8)\}$. We have $f(A) = 4$, $f(B) = 0$, $f(A \cup B) = 12$ and $f(A \cap B) = 0$. The inequation does not hold. f is not submodular. □

2.2 Baseline Algorithm

For the k-truss minimization problem, a naive solution is to enumerate all the possible edge sets of size b, and return the best one. However, the size of a real-world social network is usually very large. The number of combinations is enormous to enumerate. Due to the complexity and non-submodular property of the problem, we resort to the greedy framework. Algorithm 1 shows the baseline greedy algorithm. It is easy to verify that we only need to consider the edges in the k-truss as candidates. The algorithm iteratively finds the edge with the largest number of followers in the current k-truss (Line 3). The algorithm terminates when b edges are found. The time complexity of the baseline algorithm is $O(bm^{2.5})$.

3 Group Based Solution

In this section, novel pruning techniques are developed to accelerate the search in baseline algorithm.

3.1 Candidate Reduction

Before introducing the pruning rules, we first present some definitions involved.

Algorithm 2: Group based Algorithm

Input : G: a social network, k: truss constraint, b: the budget
Output: B: the set of deleted edges

```
1  B ← ∅ ; T_k ← k-truss of G ;          /* compute k-truss */;
2  while |B| < b do
3      mark all edges in T_k as unvisited ;
4      T ← FindGroup (T_k) ;   /* Line 12-19 */;
5      for each e in T do
6          compute F(e, T_k) ;
7          T ← T\F(e, T_k) ;
8      e* ← the edge with the most followers ;
9      update k-truss T_k ;
10     B ← B ∪ {e*} ;
11 return B
12 Function FindGroup (S):
13     C ← ∅ ; gID ← 0;          /* C stores the candidates */;
14     for each e ∈ S do
15         if sup(e, S) = k − 2 and e is unvisited then
16             GroupExpansion (S, e);          /* Line 20-32 */;
17             gID++ ;
18     return C
19 End Function
20 Function GroupExpansion (S, e):
21     Q ← ∅ ; Q.enqueue(e); mark e as visited ;
22     while Q ≠ ∅ do
23         e'(u, v) ← Q.dequeue();
24         foreach a ∈ N(u, T_k) ∩ N(v, T_k) do
25             if (u, a) is unvisited and sup = k − 2 then
26                 Q.enqueue((u, a)) ;
27                 mark (u, a) as visited ;
28             if (v, a) is unvisited and sup = k − 2 then
29                 Q.enqueue((v, a)) ;
30                 mark (v, a) as visited ;
31             update C ;
32 End Function
```

Definition 5 (triangle adjacency). *Given two triangles $\triangle_1$, $\triangle_2$ in G, they are triangle adjacent if $\triangle_1$ and $\triangle_2$ share a common edge, which means $\triangle_1 \cap \triangle_2 \neq \varnothing$.*

Definition 6 (triangle connectivity). *Given two triangles $\triangle_s, \triangle_t$ in G, they are triangle connected, denoted as $\triangle_s \leftrightarrow \triangle_t$, if there exists a sequence of θ triangles $\triangle_1, \triangle_2, ..., \triangle_\theta$ in G, such that $\triangle_s = \triangle_1, \triangle_t = \triangle_\theta$, and for $1 \leq i < \theta$, $\triangle_i$ and $\triangle_{i+1}$ are triangle adjacent.*

For two edges e and e', we say they are **triangle adjacent**, if e and e' belong to the same triangle. As shown in the baseline algorithm, we only need to consider the edges in T_k as

candidates. Lemma 1 shows that we only need to explore the edges in Q.

Lemma 1. *Given a k-truss T_k, let $P = \{e \mid sup(e, T_k) = k-2\}$. If an edge e has at least one follower, e must be in Q, where $Q = \{e \mid e \in T_k \wedge \exists e' \in P$ where e and e' are triangle adjacent$\}$.*

Proof. We prove the lemma by showing that edges in $E_G \setminus Q$ do not have followers. We divide $E_G \setminus Q$ into two sets. 1) For edge with trussness less than k, it will be deleted during the k-truss computation. 2) For an edge e in T_k, if e is not triangle adjacent with any edge in P, it means e is triangle adjacent with edges such as e' whose $sup(e', T_k) > k-2$. If we delete e, all the edges triangle adjacent with e will still have support at least $k-2$ in T_k. Thus, e has no follower. The lemma is correct. □

Based on Lemma 2, we can skip the edges that are the followers of the explored ones.

Lemma 2. *Given two edges $e_1, e_2 \in T_k$, if $e_1 \in F(e_2, T_k)$, then we have $F(e_1, T_k) \subseteq F(e_2, T_k)$.*

Proof. $e_1 \in F(e_2, T_k)$, it implies that e_1 will be deleted during the deletion of e_2. Therefore, each edge in $F(e_1, T_k)$ will be deleted when e_2 is deleted. Consequently, we have $F(e_1, T_k) \subseteq F(e_2, T_k)$. □

To further reduce the searching space, we introduce a pruning rule based on k-support group.

Definition 7 (k**-support group**). *Given a k-truss T_k, a subgraph $S \subseteq T_k$ is a k-support group if it satisfies : 1) $\forall e \in S$, $sup(e, T_k) = k-2$. 2) $\forall e_1, e_2 \in S$, suppose $e_1 \in \triangle_s$, $e_2 \in \triangle_t$. There exists a sequence of $\theta \geq 2$ triangles $\triangle_1, ..., \triangle_\theta$ with $\triangle_s = \triangle_1$, $\triangle_t = \triangle_\theta$. For $i \in [1, \theta)$, $\triangle_i \cap \triangle_{i+1} = e$ and $sup(e, T_k) = k-2$. 3) S is maximal, i.e., any supergraph of S cannot be a k-support group.*

Lemma 3 shows that edges in the same k-support group are equivalent. The deletion of any edge in a k-support group can lead to the deletion of the whole k-support group.

Lemma 3. *S is a k-support group of T_k. For $\forall e \in S$, if we delete e, we can have S deleted from T_k.*

Proof. Since S is a k-support group of T_k, for $\forall e, e' \in S$, suppose that $e \in \triangle_s, e' \in \triangle_t$, there exists a sequence of θ triangles $\triangle_1, ..., \triangle_\theta$ with $\triangle_s = \triangle_1, \triangle_t = \triangle_\theta$. For $i \in [1, \theta)$, $\triangle_i \cap \triangle_{i+1} = e_i$ and $sup(e_i, T_k) = k-2$. The deletion of any edge inside the group will destroy the corresponding triangles and decrease the support of triangle adjacent edges by 1. It will lead to a cascading deletion of subsequent triangle edges in the group due to the violation of truss constraint. Therefore, the lemma holds. □

According to Lemma 3, we only need to add one edge from a k-support group to the candidate set, and the other edges in the group can be treated as the followers of the selected edge. In the following lemma, we can further prune the edges that are adjacent with multiple edges in a k-support group.

Lemma 4. *Suppose that $e \in T_k$ and $sup(e, T_k) = w > k-2$. For a k-support group S, if e belongs to more than $w-k+2$ triangles, each of which contains at least one edge in S, then e is a follower of S.*

Proof. According to Lemma 3, by removing an edge from S, we have S deleted from T_k. Since e belongs to more than $w-k+2$ triangles, each of which contains at least one edge in S, the support of e will decrease by more than $w-k+2$ due to the deletion of S. So its support will be less than $k-2$ and it will be deleted due to the support constraint. Thus, e is a follower of S. □

3.2 Group Based Algorithm

We improve the baseline algorithm by integrating all the pruning rules above, and the details are shown in Algorithm 2. In each iteration, we first find k-support groups of current T_k and compute the candidate set T according to Lemma 3 (Line 4). This process, *i.e.,* **FindGroup** function, corresponds to Line 12-19. It can be done by conducting BFS search from edges in T_k. We use a hash table to maintain the group id (*i.e.,* gID) for each edge and the gID starts from 0 (Line 13). For each unvisited edge with support of $k-2$, we conduct a BFS search from it by calling function **GroupExpansion** (Line 20-32). During the BFS search, we visit the edges that are triangle adjacent with the current edge, and push the edges with support of $k-2$ into the queue if they are not visited (Line 25 and 28). The edges, which are visited in the same BFS round, are marked with the current gID. For the visited edges with support larger than $k-2$, we use a hash table to record its coverage with the current k-support group, and update the candidate set based on Lemma 4 (Line 31). According to Lemma 2, we can further update the candidate set after computing the followers of edges (Line 7).

4 Upper Bound Based Solution

The group based algorithm reduces the size of candidate set by excluding the edges in the same k-support group and the followers of k-support groups, which greatly accelerates the baseline method. However, for each candidate edge, we still need lots of computation to find its followers. Given an edge, if we can obtain the upper bound of its follower size, then we can speed up the search by pruning unpromising candidates. In this section, we present a novel method to efficiently calculate the upper bound required.

4.1 Upper Bound Derivation

Before introducing the lemma, we first present some basic definitions. Recall that $\tau(e)$ denotes the trussness of e.

Definition 8 (k**-triangle**). *A triangle $\triangle_{uvw}$ is a k-triangle, if the trussness of each edge is no less than k.*

Definition 9 (k**-triangle connectivity**). *Two triangles $\triangle_s$ and $\triangle_t$ are k-triangle connected, denoted as $\triangle_s \overset{k}{\leftrightarrow} \triangle_t$, if there exists a sequence of $\theta \geq 2$ triangles $\triangle_1, ..., \triangle_\theta$ with $\triangle_s = \triangle_1, \triangle_t = \triangle_\theta$. For $i \in [1, \theta)$, $\triangle_i \cap \triangle_{i+1} = e$ and $\tau(e) = k$.*

We say two edges e, e' are k-triangle connected, denoted as $e \overset{k}{\leftrightarrow} e'$, if and only if 1) e and e' belong to the same k-triangle, or 2) $e \in \triangle_s, e' \in \triangle_t$, with $\triangle_s \overset{k}{\leftrightarrow} \triangle_t$.

Definition 10 (k**-truss group**). *Given a graph G and an integer $k \geq 3$, a subgraph S is a k-truss group if it satisfies: 1) $\forall e \in S, \tau(e) = k$. 2) $\forall e, e' \in S, e \overset{k}{\leftrightarrow} e'$. 3) S is maximal, i.e., there is no supergraph of S satisfying conditions 1 and 2.*

Based on the definition of k-truss group, Lemma 5 gives an upper bound of $|F(e, T_k)|$.

Lemma 5. *If e is triangle adjacent with θ k-truss groups g_1, g_2, ..., g_θ, we have $|F(e, T_k)| \leq \sum_{i=1}^{\theta} |E_{g_i}|$.*

Proof. Suppose $sup(e, T_k) = w$, we have $w \geq k - 2$, so e is contained by w triangles and is triangle adjacent with $2w$ edges. We divide the edges which are triangle adjacent with e in T_k into two parts. 1) $\tau(e') > k$. Since the deletion of e may cause $\tau(e')$ to decrease at most 1 [Huang *et al.*, 2014; Akbas and Zhao, 2017], we have $\tau(e') \geq k$ after deleting e, which means e' has no contribution to $F(e, T_k)$. 2) $\tau(e') = k$. Suppose $e' \in g_i$. The deletion of e can cause trussness of each edge in g_i to decrease at most 1. Then e' can contribute to $|F(e, T_k)|$ with at most $|E_{g_i}|$. Thus, $\sum_{i=1}^{\theta} |E_{g_i}|$ is an upper bound of $|F(e, T_k)|$. □

4.2 Upper Bound Based Algorithm

Based on Lemma 5, we can skip the edges whose upper bound of follower size is less than the best edge in the current iteration. However, given the trussness of each edge, it may still be prohibitive to find the k-truss group that contains an edge e, since in the worst case we need to explore all the triangles in the graph. To compute the upper bound efficiently, we construct an index to maintain the relationships between edges and their k-truss groups.

To find the k-truss group for a given edge e, we extend the GroupExpansion function in Line 20-32 of Algorithm 2. It also follows the BFS search manner. The difference is that when we explore an adjacent triangle, it must satisfy the k-triangle constraint, and we only enqueue an edge, whose trussness satisfies k-triangle connectivity constraint. After finishing the BFS search starting from e, its involved k-truss groups can be found.

After deleting an edge e in the current iteration, the constructed k-truss groups may be changed. Therefore, we need to update the k-truss groups for the next iteration. The update algorithm consists of two parts, *i.e.,* update the trussness and update the groups affected by the changed trussness. To update the edge trussness, we apply the algorithm in [Huang *et al.*, 2014], which can efficiently update the edge trussness after deleting an edge e. Given the edges with changed trussness, we first find the subgraph induced by these edges. Then we reconstruct the k-truss groups for the induced subgraph and update the original ones. Based on the k-truss groups constructed, we can compute the upper bound of followers for edges efficiently. The final algorithm, named **UP-Edge**, integrates all the techniques proposed in Section 3 and 4.

5 Experiment

5.1 Experiment Setting

In the experiments, we implement and evaluate the following algorithms. 1) Exact: naive algorithm that enumerates all the combinations. 2) Support: in each iteration, it selects the edge that is triangle adjacent with the edge with minimum support in the k-truss. 3) Baseline: baseline algorithm in Section 2.2. 4) GP-Edge: group based algorithm in Section 3. 5) UP-Edge: upper bound based algorithm in Section 4.

We employ 9 real social networks (*i.e.,* Bitcoin-alpha, Email-Eu-core, Facebook, Brightkite, Gowalla, DBLP, Youtube, Orkut, LiveJournal) to evaluate the performance of the proposed methods. The datasets are public available[1]. Since the Exact algorithm is too slow, we only run Exact algorithm on Email-Eu-core and Bitcoin-alpha dataset.

Since the properties of datasets are quite different, we set the default k as 10 for 4 datasets (Gowalla, Youtube, Brightkite, DBLP) and set the default k as 20 for 3 datasets (Facebook, LiveJournal, Orkut). We set default b as 5 for all datasets. All the programs are implemented in C++. All the experiments are performed on a machine with an Intel Xeon 2.20 GHz CPU and 128 GB memory running Linux.

5.2 Effectiveness Evaluation

To evaluate the effectiveness of the proposed methods, we report the number of followers by deleting b edges. Since UP-Edge only accelerates the speed of Baseline and GP-Edge, we only report the results of UP-Edge here. Due to the huge time cost of Exact, we show the result on 3 datasets, that is, Bitcoin-alpha, Email-Eu-core and Artificial network (generated by GTGraph with 500 nodes and 5000 edges).

We set $k = 11$ and 8 for Bitcoin-alpha and Artificial network respectively, and vary b from 1 to 4. In Figure 3(a), we can see that there is only a slight drop when b=3. In Figure 3(b), there is only a small drop when b=4. In Figure 3(c), as we can see, UP-Edge also shows comparable results with Exact and they all outperform Support significantly. Similar results can be observed in Figure 3(d)-3(f) over all the datasets and the selected datasets. Figure 3(e) and 3(f) show the results on LiveJournal by varying b and k. As observed, the number of followers for the two algorithms are positive correlated with b, and k has a great impact on follower size.

Figure 4 shows a case study on DBLP with $k = 10, b = 1$. We can see that the edge between Lynn A. Volk and David W. Bates is the most pivotal relationship. This edge has 264 followers (grey edges in the figure). It is interesting that most followers have no direct connection with them.

5.3 Efficiency Evaluation

To evaluate the efficiency, we compare the response time of UP-Edge and GP-Edge with Baseline . We first conduct the experiments on all the datasets with default settings. Figure 5 shows the response time of the three algorithms. We can see that UP-Edge and GP-Edge significantly outperform Baseline in all the datasets because of the pruning techniques developed. UP-Edge is faster than GP-Edge due to the contribution

[1] https://snap.stanford.edu/data/, https://dblp.org/xml/release/

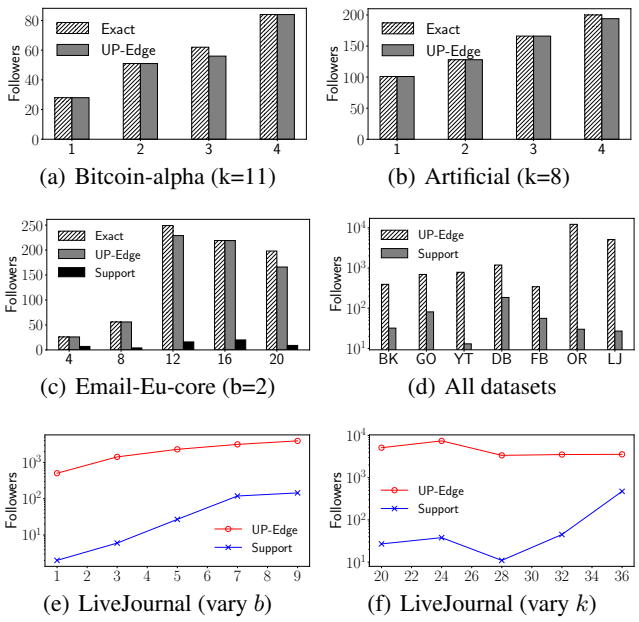


Figure 3: Effectiveness Evaluation

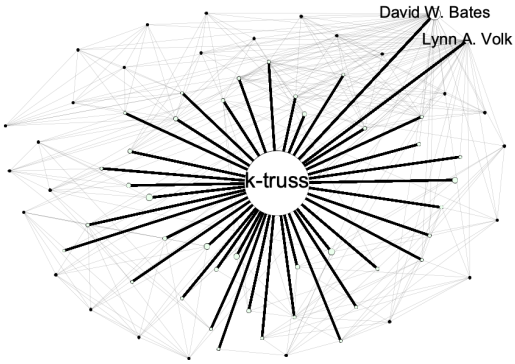


Figure 4: Case study on DBLP, k=10, b=1

of upper bound derived. Figure 6 shows the results conducted on LiveJournal by varying b and k. We can see that when b grows, the response time increases since more edges need to be selected. When k grows, the response time decreases since the searching space becomes smaller.

6 Related Work

Graph processing has been a hot topic in many areas recently, which usually requires much more computation comparing with some traditional queries [Luo *et al.*, 2008; Wang *et al.*, 2010; Wang *et al.*, 2015]. Cohesive subgraph identification is of great importance to social network analysis. In the literature, different definitions of cohesive subgraphs are proposed, such as k-core [Seidman, 1983; Wen *et al.*, 2016], k-truss [Huang and Lakshmanan, 2017], clique [Tsourakakis *et al.*, 2013], dense neighborhood graph [Kreutzer *et al.*, 2018], etc. In the literature, numerous research is conducted to

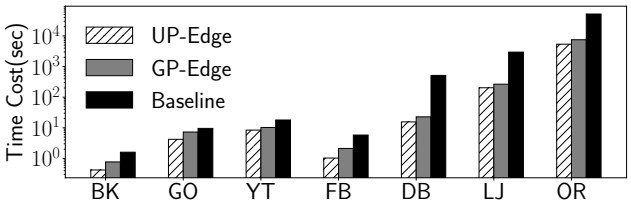


Figure 5: Running time on all datasets

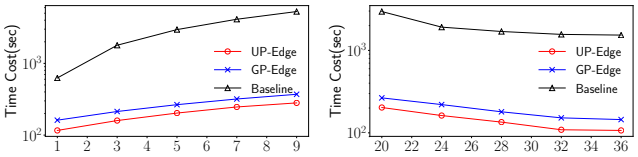


Figure 6: Efficiency Evaluation

investigate the k-truss decomposition problem under different settings, including in-memory algorithms [Cohen, 2008], external-memory algorithms [Wang and Cheng, 2012], distributed algorithms [Chen *et al.*, 2014], etc. In some studies, authors leverage the k-truss property to mine required communities [Huang *et al.*, 2014; Huang and Lakshmanan, 2017]. Huang et al. [Huang *et al.*, 2016] investigate the truss decomposition problem in uncertain graphs. Recently, some research focuses on modifying the graph to maximize/minimize the corresponding metric [Bhawalkar *et al.*, 2015; Zhang *et al.*, 2017; Zhu *et al.*, 2018; Medya *et al.*, 2018]. Bhawalkar et al. [Bhawalkar *et al.*, 2015] propose the anchored k-core problem, which tries to maximize the k-core by anchoring b nodes, while Zhang et al. [Zhang *et al.*, 2017] and Zhu et al. [Zhu *et al.*, 2018] investigate the problem of k-core minimization by deleting nodes and edges, respectively. In [Medya *et al.*, 2018], Medya et al. try to maximize the node centrality by adding edges to the graph. However, these techniques cannot be extended for our problem.

7 Conclusion

In this paper, we study the k-truss minimization problem. We first formally define the problem. Due to the hardness of the problem, a greedy baseline algorithm is proposed. To speed up the search, different pruning techniques are developed. In addition, an upper bound based strategy is presented by leveraging the k-truss group concept. Lastly, we conduct extensive experiments on real-world social networks to demonstrate the advantage of the proposed techniques.

Acknowledgments

Xiaoyang Wang is supported by NSFC61802345. Xuemin Lin is supported by 2018YFB1003504, NSFC61232006, DP180103096, DP170101628, DP150102728. Weijie Zhu and Mengqi Zhang are co-first authors for this paper.

References

[Akbas and Zhao, 2017] Esra Akbas and Peixiang Zhao. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB*, 10(11):1298–1309, 2017.

[Bhawalkar *et al.*, 2015] Kshipra Bhawalkar, Jon Kleinberg, Kevin Lewi, Tim Roughgarden, and Aneesh Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.

[Chen *et al.*, 2014] Pei-Ling Chen, Chung-Kuang Chou, and Ming-Syan Chen. Distributed algorithms for k-truss decomposition. In *IEEE International Conference on Big Data*, 2014.

[Cohen, 2008] Jonathan Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 2008.

[Cui *et al.*, 2018] Yi Cui, Di Xiao, and Dmitri Loguinov. On efficient external-memory triangle listing. *TKDE*, 2018.

[Huang and Lakshmanan, 2017] Xin Huang and Laks V. S. Lakshmanan. Attribute-driven community search. *PVLDB*, 2017.

[Huang *et al.*, 2014] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[Huang *et al.*, 2016] Xin Huang, Wei Lu, and Laks V.S. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *SIGMOD*, 2016.

[Karp, 1972] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[Kreutzer *et al.*, 2018] Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *ACM Trans. Algorithms*, 15(2), 2018.

[Luo *et al.*, 2008] Yi Luo, Wei Wang, and Xuemin Lin. Spark: A keyword search engine on relational databases. In *ICDE*, 2008.

[Medya *et al.*, 2018] Sourav Medya, Arlei Silva, Ambuj Singh, Prithwish Basu, and Ananthram Swami. Group centrality maximization via network design. In *ICDM*, 2018.

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

[Tsourakakis *et al.*, 2013] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, 2013.

[Wang and Cheng, 2012] Jia Wang and James Cheng. Truss decomposition in massive networks. *Proc. VLDB Endow.*, 2012.

[Wang *et al.*, 2010] Chaokun Wang, Jianmin Wang, Xuemin Lin, Wei Wang, Haixun Wang, Hongsong Li, Wanpeng Tian, Jun Xu, and Rui Li. Mapdupreducer: detecting near duplicates over massive datasets. In *SIGMOD*, 2010.

[Wang *et al.*, 2015] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE*, 2015.

[Wen *et al.*, 2016] Dong Wen, Lu Qin, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. I/O efficient core graph decomposition at web scale. In *ICDE*, 2016.

[Xiao *et al.*, 2017] Di Xiao, Yi Cui, Daren BH Cline, and Dmitri Loguinov. On asymptotic cost of triangle listing in random graphs. In *PODS*, 2017.

[Yu *et al.*, 2013] Weiren Yu, Xuemin Lin, Wenjie Zhang, Lijun Chang, and Jian Pei. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *PVLDB*, 7(1), 2013.

[Zhang *et al.*, 2017] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *AAAI*, 2017.

[Zhu *et al.*, 2018] Weijie Zhu, Chen Chen, Xiaoyang Wang, and Xuemin Lin. K-core minimization: An edge manipulation approach. In *CIKM*, 2018.