Budget-Constrained Truss Maximization over Large Graphs: A Component-based Approach

Xin Sun
College of Intelligence and Computing, Tianjin University
Tianjin, China
sun_xin@tju.edu.cn

Xin Huang
Hong Kong Baptist University
Hong Kong, China
xinhuang@comp.hkbu.edu.hk

Zitan Sun
Hong Kong Baptist University
Hong Kong, China
zitansun@comp.hkbu.edu.hk

Di Jin
College of Intelligence and Computing, Tianjin University
Tianjin, China
jindi@tju.edu.cn

ABSTRACT

Cohesive substructure identification is one fundamental task of graph analytics. Recently, a useful problem of dense subgraph maximization has attracted significant attentions, which aims at enlarging a dense subgraph pattern using a few new edge insertions, e.g., k-core maximization. As a more cohesive subgraph of k-core, k-truss requires that each edge has at least $k-2$ triangles within this subgraph. However, the problem of k-truss maximization has not been studied yet. In this paper, we motivate and formulate a new problem of budget-constrained k-truss maximization. Given a budget of b edges and an integer $k \geq 2$, the problem is to find and insert b new edges into a graph G such that the resulted k-truss of G is maximized. We theoretically prove the NP-hardness of k-truss maximization problem. To efficiently tackle it, we analyze non-submodular property of k-truss newcomers function and develop non-conventional heuristic strategies for edge insertions. We first identify high-quality candidate edges with regard to $(k-1)$-light subgraphs and propose a greedy algorithm using per-edge insertion. Besides further improving the efficiency by pruning disqualified candidate edges, we finally develop a component-based dynamic programming algorithm for enlarging k-truss mostly, which makes a balance of budget assignment and inserts multiple edges simultaneously into all $(k-1)$-light components. Extensive experiments on nine real-world graphs demonstrate the efficiency and effectiveness of our proposed methods.

CCS CONCEPTS

• **Theory of computation** → *Graph algorithms analysis*; • **Mathematics of computing** → *Graph theory*.

KEYWORDS

k-truss; subgraph maximization; edge insertions;

ACM Reference Format:
Xin Sun, Xin Huang, Zitan Sun, and Di Jin. 2021. Budget-Constrained Truss Maximization over Large Graphs: A Component-based Approach. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482324

1 INTRODUCTION

Graph is an essential model to represent entities and their connected relationships in many real-world networks, such as social networks [23, 32, 42], the Web [10], collaboration network [45], biological networks [14], to name a few. A fundamental task of complex graph analytics is to identify cohesive portions revealing latent and critical community structures, which are frequently modeled as cohesive subgraphs [21].

In the literature, many cohesive subgraph notions have been proposed including k-clique, n-clan, n-club, and k-plex [33, 39]. The computations of all the above cohesive subgraphs are NP-hard and not scalable over large graphs. A popular dense subgraph of k-truss, receives many attentions recently [11, 17, 22, 24, 53]. As a relaxation of k-clique, k-truss requires that every edge is contained in at least $(k-2)$ triangles in the k-truss [7], which can be computed efficiently in polynomial time. The k-truss investigation has been conducted in various important applications and different kinds of networks, including community search [30], truss minimization [53], network visualization [50], public-private social networks [11], and probabilistic networks [22, 40].

One useful problem of dense subgraph maximization has attracted significant attentions recently, which has several applications in improving communication network stability and online social network services. One representative task is core maximization [6, 51], which studies to enlarge another dense subgraph of k-core by inserting a few edges into a graph. The definition of k-core requires that every vertex has at least k neighbors. In comparison, the k-truss is conceptually more rigorous than k-core, as k-truss is defined on the edge and its strength measured by the number of triangles whereas k-core is defined on the node and its strength measured by the simple degree [17, 36]. The k-truss model ensures the strong tie strength among users with at least k-2 common neighbors. However, to our best knowledge, the problem

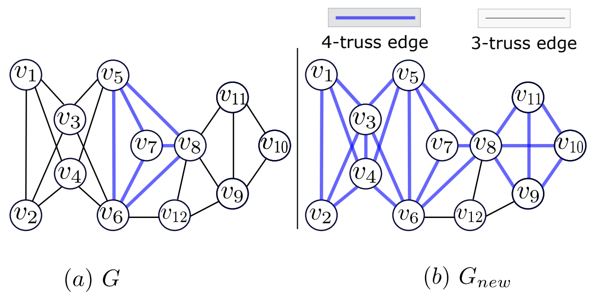

4-truss edge
3-truss edge
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12}
$(a)\ G$
$(b)\ G_{new}$

Figure 1: An example of k-truss maximization on graph G, where $k = 4$ and the budget $b = 2$. After inserting two edges $E_\Delta = \{(v_3, v_4), (v_8, v_{10})\}$ into graph G, a new graph G_{new} has the maximum 4-truss with 22 edges.

of dense subgraph maximization on the k-truss model has not been studied yet.

In this paper, we investigate a new problem of budget-constrained k-truss maximization, that is, given a number k and a budget b, it aims at inserting a set of b new edges into graph G such that the size of k-truss is maximized. We motivate the k-truss maximization problem using a wide range of real-life representative applications, for example, improving flight network connectivity and social group engagement, enhancing stability of P2P networks, and identifying missing defense links in military networks, as analyzed in the dense subgraph maximizing problem of another k-core model [6, 51] and further illustrated in Section 3.3.

Motivating example. Consider a social network G with 12 vertices as shown in Figure 1(a), where each vertex represents a user and an edge represents a friendship between two users. The whole graph G is the 3-truss, as each edge is contained in at least one triangle in G. The induced subgraph of G by $\{v_5, v_6, v_7, v_8\}$ is the 4-truss in blue, where each pair of users have two common friends. Assume that $k = 4$ and the budget $b = 2$, we seek to enlarge 4-truss by inserting 2 new edges into graph G. An optimal solution is to insert two edges (v_3, v_4) and (v_8, v_{10}), which maximizes 4-truss from the original six edges of G to the new 22 edges of G_{new} in blue as shown in Figure 1(b).

Different edge insertion strategies have significantly different performances, e.g., considering an alternative answer for the above example G in Figure 1(a) by inserting two edges (v_1, v_8) and (v_1, v_{10}), which leads to no any increment size of 4-truss at all. However, developing effective algorithms for k-truss maximization brings non-trivial challenges. We analyze the hardness of k-truss maximization problem and theoretically prove that it is NP-hard reduced from a well-known maximum coverage problem. Even worse, we observe that the objective function of k-truss newcomer does not enjoy the submodularity for a simple greedy approximation algorithm design. Therefore, we explore heuristic strategies to develop efficient algorithms for enlarging k-truss using a few new edges. The key idea is to identify those edges that are easily converted into k-truss using a small cost of edge insertions. We first give a definition of $(k-1)$-*light*, which is a subgraph of $(k-1)$-truss formed by all edges of the trussness of $k-1$. Then, we prune those candidate $(k-1)$-light edges of low quality, in terms of the triangle weights. Our first greedy algorithm is a per-edge insertion method, which iteratively inserts one edge with the largest k-truss newcomers into graph G until the budget of b edges is used up. To improve the efficiency, we develop advanced techniques to further prune disqualified candidate edges, by distinguishing *stable edges* and *unstable edges*. Moreover, we propose a novel component-based approach, which divides the graph into several independent components and uses dynamic programming techniques to enlarge k-truss in a *global* optimization. This is an integrated method by balancing the budget assignment such that a limited number of inserted edges enlarges the k-truss mostly.

In summary, we make the contributions in this paper as follows:

- We motivate a new problem of k-truss maximization, which inserts a given budget of b new edges into a graph to enlarge k-truss (Section 3).
- We theoretically prove that the k-truss maximization problem is NP-hard. We also analyze the property of k-truss newcomer function and show its non-submodularity (Section 4).
- We give a formal definition of candidate weight and formulate the candidates for edge insertions. We then propose a greedy approach to iteratively select one best candidate edge to insert into graphs. Moreover, we further design an optimization technique for pruning candidates (Section 5).
- For improving the efficiency and quality, we propose a component-based approach CBTM for k-truss maximization. It first partitions the graph into multiple components and makes k-truss estimations for each component, and then conducts an edge insertion assignment to maximize k-truss newcomers by dynamic programming techniques (Section 6).
- We conduct extensive experiments on nine real-world large graphs. We show that our component-based approach CBTM can efficiently and effectively enlarge k-truss using a limited budget b (Section 7).

We discuss related work in Section 2 and conclude the paper in Section 8.

2 RELATED WORK

Our work is related to *k-truss mining* and *subgraph enhancement*.

K-truss mining. The k-truss is the largest dense subgraph where every edge exists in at least $k-2$ triangles in the subgraph [7]. Truss decomposition is to find the trussness of every edge in a graph. An in-memory algorithm of truss decomposition is presented in [41]. The k-truss decomposition has numerous applications in large graph analysis, including visualization [2], community search [9, 18, 29, 38], probabilistic graphs [12], modular centrality [15] and maximal clique finding [35]. Parallel algorithms for accelerating k-truss decomposition have also been studied [4, 5, 25, 37]. There also exist many studies on k-truss mining [7] over different networks, including dynamic graphs [49], directed graphs [30], uncertain graphs [22], heterogeneous graphs [43] and public-private graphs [11]. Recently, the problem of k-truss minimization is to shrink k-truss as most as possible by deleting b edges [53]. Several k-truss based community models have been developed for finding communities in an online manner [20].

Subgraph enhancement. Several studies work on the subgraph enhancement, e.g., the anchored k-core to enlarge the k-core by strengthening a set of nodes, these nodes will retain within the anchored k-core, even if their degree within the k-core subgraph is less than k [3, 27, 47]. This problem can identify critical vertices (e.g., people) whose participation is critical to overall engagement of the networks. Corò et al. [8] investigate the problem of improving the graph reachability through edge insertions. The closest work to us is the edge addition approach to maximize the k-core [6, 51]. Chitnis et al. [6] propose solutions for graphs with bounded tree-width. Zhou et al. [51] propose a heuristic algorithm and use a layer structure to reduce the candidate set and the time for computing followers. Compared with the k-core maximization problem where each vertex has degree at least k within the k-core [6, 51], k-truss enjoys more cohesive structure than k-core. On the other hand, some studies aim at minimizing the size of k-core [31, 52]. Related problems also include the anchored k-truss problem [48] and the collapsed k-truss problem [46]. Different from all the above studies, we study a new problem of k-truss maximization in this paper, and propose novel techniques of $(k-1)$-light based per-edge insertion strategy and component-based dynamic programming algorithm w.r.t. a limited budget of new edge insertions.

3 PRELIMINARY

We consider an undirected simple graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. Given a subgraph $H = (V(H), E(H)) \subseteq G$, we denote the set of neighbors of a vertex v in H by $N_H(v)$, i.e., $N_H(v) = \{u \in V(H) : (v, u) \in E(H)\}$. We denote the degree of v in graph G by $d(v) = |N_G(v)|$ and the maximal degree by $d_{max} = \max_{v \in V} d(v)$, respectively. A triangle is a 3-clique formed by three vertices v, u, w, denoted as $\triangle_{vuw}$. The support of an edge e, denoted as $sup_H(e)$, is the number of triangles containing e in H. When it is clear from the context, we drop the subscript H and G of our notations.

3.1 K-Truss Newcomers

We begin with a definition of k-truss.

Definition 1 (K-Truss [41]). *Given a graph G and an integer $k \geq 2$, the k-truss, denoted as T_k, is the largest subgraph of G such that each edge is contained in at least $k - 2$ triangles in T_k.*

The k-truss subgraphs inherit good hierarchical properties, such as the k-truss is always a subgraph of k' truss for $k' \leq k$, i.e., $T_k \subseteq T_{k'}$. Thus, an edge may appear in multiple k-trusses T_k for different k. The trussness of an edge e in G is the largest k such that there exists a $T_k \subseteq G$ containing e, which is represented as $\tau(e)$. Consider the graph G in Figure 1, the edge (v_5, v_6) appears both in 3-truss and 4-truss. The trussness of (v_5, v_6) is $\tau((v_5, v_6)) = 4$.

In this paper, we are interested to add a few new edges E_Δ into graph G to enlarge k-truss T_k. The new graph and the corresponding new k-truss are denoted as $G^* = (V, E \cup E_\Delta)$ and T_k^*, respectively. We give a new definition of *k-truss newcomers* as follows.

Definition 2 (K-Truss Newcomers). *The k-truss newcomers are a set of edges that do not appear in T_k of graph G but belong to k-truss T_k^* of new graph G^* after the edge insertions of E_Δ. The number of k-truss newcomers is denoted as $\mathsf{f}(E_\Delta, k)$= $|T_k^*| - |T_k|$.*

The k-truss newcomers may be either the newly inserted edges or the original edges in graph G. The edges, which may be inserted to the graph, are called candidate edges.

3.2 Problem Formulation

Based on the above definitions, we formulate the problem of budget-constrained k-truss maximization (KTM-Problem) as follows.

Problem 1. *Given a graph $G(V, E)$, an integer $k \geq 2$, and a budget of $b \geq 1$ edges that can be inserted into G, the truss maximization problem aims to find a set of new edges $E_\Delta = \{(v, u) : v, u \in V, (v, u) \notin E\}$ and $|E_\Delta| \leq b$ such that the k-truss newcomers $\mathsf{f}(E_\Delta, k)$ is maximized in new graph $G^* = (V, E \cup E_\Delta)$, i.e.,*

$$E_\Delta^* = \underset{|E_\Delta| \leq b}{\arg\max}\, |T_k^*| - |T_k|.$$

Example 1. *Consider the graph G in Figure 1. Assume that $k = 4$ and $b = 2$. Our problem of k-truss maximization is to insert two new edges into G such that the new 4-truss is maximized. One choice is to insert two edges $\{(v_1, v_5), (v_2, v_6)\}$ into G with 11 newcomers. However, this solution is not optimal. The best choice is to insert $\{(v_3, v_4), (v_8, v_{10})\}$ edges into G, which brings 16 k-truss newcomers. The enlarged 4-truss of G_{new} is shown in Figure 1(b).*

3.3 Applications

We motivate the KTM-Problem using two useful applications.

Flight network connectivity improvement. It is well known that adding more routes bring a connectivity improvement of transportation network, e.g., flight network, ferry route networks, rail networks, and so on. A strong network connectivity certainly gives more route choices for users and facilitates traveling. A connected k-truss enjoys good structural property of $(k-1)$-edge-connectivity, indicating that the k-truss network keeps connected by removing any fewer than k-2 edges [7]. In flight networks, one route between any two airports can be suspended due to unexpected circumstances. We need to offer more alternative transit plans when direct flights fail. Therefore, our KTM-Problem applied on flight networks aims at maximizing a strong-connected k-truss network by adding a budget of b new routes.

Social group engagement. Many online social network applications feature a friend suggestion function to help form friendships, e.g., Facebook, Instagram, and so on [13]. The newly formed friendships are corresponding to our problem setting of edge insertions. Given a social network, we can reinforce social groups by creating new friend relationships and construct a larger k-truss community. Every two people have at least k-2 common friends under the k-truss community model, ensuring solid and stable relationships. People are more likely to share and contributes user-generated contents if they have enough common friends [1, 44]. Social-based activities such as group shopping and friend recommendations may achieve better outcomes in an enlarged k-truss group.

Besides the above applications, our problem is also applicable to other application scenarios of k-core maximization, due to that k-truss is a more generalization and denser subgraph of k-core, e.g., *enhancing stability of P2P networks*, and *identifying missing defense links in military networks* [6, 51].

4 PROBLEM HARDNESS ANALYSIS

In this section, we first analyze the hardness of k-truss maximization problem. Then, we present the non-submodular property of k-truss newcomers function $\mathsf{f}(E_\Delta, k)$, with regard to w.r.t. E_Δ for a given k.

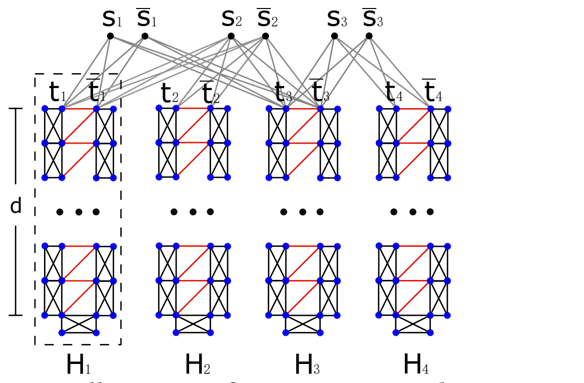


Figure 2: An illustration of constructing graph G in KTM-Problem **from an instance of** MC-Problem **where** $S_1 = \{t_1, t_3\}$, $S_2 = \{t_1, t_2, t_3\}$, $S_3 = \{t_3, t_4\}$.

Theorem 1. *The* KTM-Problem *is NP-hard.*

Proof. We reduce the well-known NP-hard problem of maximum coverage (MC-Problem) to our problem [26]. Given a ground set $T = \{t_1, \ldots, t_n\}$ and a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, where $T = \bigcup_{j=1}^{m} S_j$. For two given numbers X and b where $b \leq m$, the decision version of MC-Problem is asking whether there exists a selection of b sets such that $| \cup_{j=1}^{b} S_{l_j} | \geq X$ where $1 \leq l_j \leq m$ for $1 \leq j \leq b$, which is also NP-hard.

Given an instance of MC-Problem, we construct an instance of KTM-Problem as follows. For each set $S_i \in \mathcal{S}$, we create two vertices s_i and $\bar{s}_i$. For each element $t_j \in S_i$, we create two vertices t_j and $\bar{t}_j$, and add edge between any pair nodes of $\{s_i, \bar{s}_i, t_j, \bar{t}_j\}$, except for the edge $(s_i, \bar{s}_i)$ as shown in Figure 2. In addition, for each edge $(t_j, \bar{t}_j)$, we link $(t_j, \bar{t}_j)$ with one d-extendible subgraph, denoted as H_j, which is with all blue nodes and surrounded with multiple 4-cliques as shown in Figure 2. Here, $d \gg 4nm$. Except for the edge $(t_j, \bar{t}_j)$, other each edge $e' \in E(H_j)$ has at least two triangles in graph H_j, reflecting that H_j is 3-truss but not 4-truss. Now, we have a completed graph $G = (V, E)$ where $V = \{s_1, \bar{s}_1, \ldots, s_m, \bar{s}_m\} \cup (\bigcup_{j=1}^{n} V(H_j))$ and $E = \{(s_i, t_j), (s_i, \bar{t}_j), (\bar{s}_i, t_j), (\bar{s}_i, \bar{t}_j) : \forall t_j \in S_i, 1 \leq i \leq m\} \cup (\bigcup_{j=1}^{n} E(H_j))$. Figure 2 shows an example of graph G with $n = 4$ and $m = 3$. For $k = 4$, let be the decision version of KTM-Problem as whether there exists E_Δ edges to be inserted into G such that $|E_\Delta| = b$ and $\mathsf{f}(E_\Delta, k) \geq 2dX$. The hardness follows from this.

$(\Rightarrow)$: Suppose there exists $\mathcal{S}^* \subseteq \mathcal{S}$ as a YES-instance of MC-Problem, i.e. $|\mathcal{S}^*| = b$ and $| \cup_{S_i \in \mathcal{S}^*} S_i| \geq X$. For each set $S_i \in \mathcal{S}^*$, we add an edge $(s_i, \bar{s}_i)$ into G. Let G_{new} be the new graph after edge insertions. Thus, $(s_i, \bar{s}_i)$ becomes a 4-truss newcomer in G_{new}. Let $T^* = \cup_{S_i \in \mathcal{S}^*} S_i \subseteq T$ and $|T^*| \geq X$, indicating that at least X edges of $(t_j, \bar{t}_j)$ become 4-truss newcomers for $t_j \in T^*$. Consequently, at least X subgraphs of H_j belong to 4-truss in G_{new}. Each graph H_j has $2d$ newcomers. Thus, the total number of newcomers has at least $2dX$, which is a YES-instance of KTM-Problem.

$(\Leftarrow)$: Suppose E^*_Δ is a YES-instance of KTM-Problem, i.e., it achieves at least $2dX$ newcomers as $\mathsf{f}(E^*_\Delta, k) \geq 2dX$. Let be the graph $G_{new} = (V, E \cup E^*_\Delta)$. We prove it by contradiction. Assume that at least X subgraphs of H_j become as the part of 4-truss in G_{new}; Otherwise, we have less than X subgraphs of H_j that belong to 4-truss, reflecting $f(E^*_\Delta, k) < 2dX$ as $d \gg 4nm$. As the most newcomer benefit of inserting such edges $(s_{l_j}, \bar{s}_{l_j})$ and $d \gg 4nm$, it should add the b edges $E^*_\Delta = \{(s_{l_j}, \bar{s}_{l_j}) : 1 \leq j \leq b, 1 \leq l_j \leq m\}$ into G. Let be the set $T^* = \{t_j : H_j$ belong to 4-truss $\}$ and $\mathcal{S}^* = \{S_j : (s_{l_j}, \bar{s}_{l_j}) \in E^*_\Delta\}$. This reflects that there exists $\mathcal{S}^*$ such that $|\mathcal{S}^*| = b$ and $| \bigcup_{S_j \in \mathcal{S}^*} S_j| \geq |T^*| \geq X$, which is a YES-instance of MC-Problem.

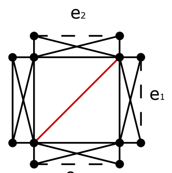


Figure 3: An example for graph G in submodular analysis.

Theorem 2. $\mathsf{f}(E_\Delta, k)$ *is not submodular w.r.t.* E_Δ.

Proof. Consider a graph G in Figure 3 and $k = 4$. The function of $\mathsf{f}(E_\Delta, k)$ w.r.t. E_Δ, is reformulated as $\mathsf{g}(E_\Delta) = \mathsf{f}(E_\Delta, 4)$. Let $S = \{e_2\}$, $T = \{e_2, e_3\}$, and $S \subset T$. We have $\mathsf{g}(S) = 6$ and $\mathsf{g}(T) = 12$. As $e_1 \notin T$ and $e_1 \notin S$, we have $\mathsf{g}(S \cup \{e_1\}) = 12$ and $\mathsf{g}(T \cup \{e_1\}) = 19$. Thus, $\mathsf{g}(S \cup \{e_1\}) - \mathsf{g}(S) = 6$ and $\mathsf{g}(T \cup \{e_1\}) - \mathsf{g}(T) = 7$. As a result, we have $\mathsf{g}(S \cup \{e_1\}) - \mathsf{g}(S) < \mathsf{g}(T \cup \{e_1\}) - \mathsf{g}(T)$, indicating $\mathsf{g}(E_\Delta)$ and $\mathsf{f}(E_\Delta, k)$ is not submodular w.r.t. E_Δ.

In view of the above NP-hardness and non-submodularity of KTM-Problem, it infers that the prospects for efficient approximation algorithms are not promising.

5 PER-EDGE INSERTION GREEDY METHOD

In this section, we propose a heuristic approach that greedily inserts one edge into G at each round of budget consumption. Then, we develop candidate pruning techniques to improve the efficiency.

5.1 Candidate Edges

To address KTM-Problem, the first key step is to identify high-quality candidate edges that can be inserted into graph G to enlarge k-truss. There exists a large room for candidate edge selection in large sparse graphs as $E_\Delta \subseteq V \times V \setminus E$. We give theoretical analysis to prune candidate edges as follows.

Let us consider one simplest case for $b = 1$, i.e., inserting an edge $e_0 \notin E$ into G. According to the rules of k-truss maintenance over dynamic graphs [19], the trussness of any possible edge increases at most by one for an edge insertion. Thus, only the edges with trussness $k - 1$, have chances to appear in the enlarged k-truss T^*_k. We give a new definition of $(k - 1)$-light as follows.

Definition 3 ($(k-1)$-Light). *A $(k - 1)$-light is an induced subgraph of G by all edges e with $\tau(e) = k - 1$, denoted by L_{k-1}.*

Thus, for a $(k - 1)$-light edge (u, v), we identify an candidate edge surrounding the $(k - 1)$-light edge to form a new triangle $\triangle_{uvw}$, which may increase the support of (u, v) and enlarge k-truss by creating k-truss newcomers. However, not all newly formed triangles are likely to be valid.

Definition 4 (Triangle Weight). *The weight of a triangle $\triangle_{uvw}$ is defined as the minimum trussness of three edges as $\min\{\tau((u, v)), \tau((u, w)), \tau((v, w))\}$. A triangle with weight k is also called a k-level triangle. Obviously, a k-level triangle can exist in k-truss. We denote the k-level triangles containing e by $\triangle^k_e$, and the cardinality of $\triangle^k_e$ by $|\triangle^k_e|$.*

Algorithm 1 Per-Edge Insertion Greedy Algorithm

```
Input: $G = (V, E)$, truss value $k$, budget $b$
Output: A set $E_\Delta$ of newly inserted edges
1: Initialization: $E_\Delta \leftarrow \emptyset$;
2: Find the candidate edges $C$ using Algorithm 2;
3: while $|E_\Delta| < b$ do
4:     for each edge $e \in C$ do
5:         Compute $k$-truss newcomers: $\mathrm{f}(\{e\}, k) \leftarrow |T_k^*| - |T_k|$;
6:     $e^* \leftarrow \arg\max_{e \in C} \mathrm{f}(\{e\}, k)$;
7:     $E_\Delta \leftarrow E_\Delta \cup \{e^*\}$;
8:     $C \leftarrow C \setminus \{e^*\}$;
9:     Update graph $G$ by adding the edge $e^*$ into $G$;
10: return $E_\Delta$;
```

Only the newly formed triangles with a weight of $k-1$ may contribute to the newcomers. To evaluate goodness of candidate edges, we give a useful definition of candidate weight.

Definition 5 (Candidate Weight). *For a candidate edge $e = (u, v) \notin E$, the candidate weight of e is defined as $\lambda(e) = |\{w \in N(u) \cap N(v) : \min\{\tau((u,w)), \tau((v,w))\} \geq k-1\}|$.*

Actually, we can find that only candidate edges e_0 with $\lambda(e_0) \geq k-2$ are feasible to contribute k-truss newcomers, which is motivated by the k-truss maintenance rules [19]. In other words, those candidate edges e_0 with $\lambda(e_0) < k-2$ cannot appear in k-truss after one edge insertion. As a result, we have

Theorem 3. *The feasible candidate set for an edge insertion is $C = \{e \in V \times V \setminus E : \lambda(e) \geq k-2\}$, which can form $(k-1)$-level triangles with other edges $e' \in L_{k-1}$.*

5.2 Per-Edge Insertion Greedy Algorithm

We present a greedy algorithm Baseline for k-truss maximization by per-edge insertions, which is outlined in Algorithm 1. The general idea of Algorithm 1 is to insert one edge $e^* \in C$ with the largest $\mathrm{f}(\{e^*\}, k)$ into G, until all b budgets of edge insertions are used up. Specifically, the algorithm initializes E_Δ as an empty set and finds all candidate edges C by invoking a procedure of Find-Candidates in Algorithm 2. Next, it iteratively computes the newcomers for each edge and adds one into E_Δ until $|E_\Delta| = b$ (lines 3-9). At each iteration, it computes the k-truss newcomers $\mathrm{f}(\{e\}, k)$ for each edge $e \in C$, which simulates an edge insertion of e into the current graph G and calculates the number of new edges in k-truss [19] (lines 4-5). Then, for the best candidate edge e^* with the largest newcomer contribution, it adds e^* into answer E_Δ and graph G, and removes e^* from candidates C (lines 6-7).

Candidate identification. The procedure of finding all candidate edges is described in Algorithm 2. Specifically, the algorithm identifies the feasible candidates C as the edges $e \notin E$ with weight $\lambda(e) \geq k-2$ by Theorem 3. For each $(k-1)$-light edge $e = (u, v) \in L_{k-1}$, it checks the candidate feasibility of two possible edges (w, v) for $w \in N(u)$ and (w, u) for $w \in N(v)$, respectively (lines 2-8). Following the rule that forming a new triangle $\triangle_{uvw}$ with weight $k-1$, it adds one possible edge $e = (w, v)$ into C when the other two edges (w, u) and (u, v) have trussness no less than $k-1$ (lines 3-5). Similar

Algorithm 2 Find-Candidates

```
Input: Graph $G(V, E)$, truss value $k$
Output: Candidate edge set $C$
1: Initialization: $C \leftarrow \emptyset$;
2: for each edge $e = (u, v) \in L_{k-1}$ do
3:     for each $w \in N(u)$ do
4:         if $\tau((w, u)) \geq k-1$ and $(w, v) \notin E \cup C$ then
5:             $C \leftarrow C \cup \{(w, v)\}$;
6:     for each $w \in N(v)$ do
7:         if $\tau((w, v)) \geq k-1$ and $(w, u) \notin E \cup C$ then
8:             $C \leftarrow C \cup \{(w, u)\}$;
9: for each edge $e = (u, v) \in C$ do
10:     Compute the candidate weight $\lambda(e)$ by Def. 5;
11:     if $\lambda(e) < k-2$ then
12:         $C \leftarrow C \setminus \{(u, v)\}$;
13: return $C$;
```

checking are done for edges $e = (w, u)$ for $w \in N(v)$. Next, the algorithm further deletes from C the disqualified candidate edges e with $\lambda(e) < k-2$ (lines 9-12).

Example 2. *Consider the graph G in Figure 1, $k = 4$, and $b = 2$. Algorithm 1 first selects the edge (v_3, v_4) with the largest newcomers of 10 and then the edge (v_8, v_{10}) with 6 newcomers. Finally, two inserted edges are $E_\Delta = \{(v_3, v_4), (v_8, v_{10})\}$, and $\mathrm{f}(E_\Delta, 4) = 16$.*

Complexity analysis. We analyze the time and space complexity of Algorithm 1. Let be the number of candidate edges as $c = |C|$. Moreover, we assume that $n \leq m+1$, w.l.o.g., by considering that the graph G is connected [41].

Theorem 4. *Algorithm 1 takes $O(bcm^{1.5})$ time in $O(m)$ space.*

Proof. Algorithm 1 takes $O(m^{1.5})$ time to find all candidate edges, which invokes the triangle listing in Algorithm 2 (line 2). For each edge insertion, we simulate the insertion of all candidate edges and compute the gain of k-truss newcomers in $O(cm^{1.5})$ (lines 4-5). Overall, Algorithm 1 takes $O(bcm^{1.5})$ time for b new edge insertions. As we do not store all the candidate edges, the space complexity is optimized to $O(m+n) = O(m)$.

5.3 Candidate Pruning Optimizations

Algorithm 1 suffers from inefficiency, due to the time-consuming step of calculating the k-truss newcomers for all candidate edges $e \in C$ (line 5 of Algorithm 1). We introduce an optimization technique to further prune the candidate set. We call the new method in Algorithm 1 equipped with the following pruning strategy, as the Greedy Truss Maximization (GTM).

Candidate edges reduction. We reduce the size of candidate edges C as follows. Recall that the truss decomposition [41] iteratively removes an edge with the support less than $k-2$ to find the k-truss. Some edges in $(k-1)$-light have no less than $k-2$ triangles at the beginning, but are finally removed from k-truss, due to a collapse by other edges e with $|\Delta_e^{k-1}| < k-2$. Since the edges in $(k-1)$-light satisfy $|\Delta_e^{k-1}| \geq k-3$, the edges e leading the collapse can be denoted by $|\Delta_e^{k-1}| = k-3$. In other words, we

Table 1: An example of stable edges and unstable edges in $(k-1)$-light of graph G in Figure 1. Here, $k=4$.

Edge type	Included edges
Stable edge	$(v_1,v_2),(v_9,v_{11}),(v_8,v_9),(v_8,v_{12})$
Unstable edge	$(v_1,v_3),(v_2,v_3),(v_1,v_4),(v_2,v_4),(v_3,v_5)$ $(v_3,v_6),(v_4,v_5),(v_4,v_6),(v_6,v_{12})$ $(v_8,v_{11}),(v_{10},v_{11}),(v_9,v_{12})(v_9,v_{10})$

can partition the edges of $(k-1)$-light into two categories: *stable edges* and *unstable edges*. Specifically, we have

- stable edges $E_s = \{e \in L_{k-1} : |\Delta_e^{k-1}| \geq k-2\}$ and
- unstable edges $E_u = \{e \in L_{k-1} : |\Delta_e^{k-1}| = k-3\}$.

For a stable edge $e \in E_s$, it already gets an enough triangle support to be contained in k-truss. For a unstable edge $e \in E_u$, it must need at least one new valid triangle to increase the edge support, which is able to join the final enlarged k-truss. Thus, the unstable edges are essentially important for k-truss maximization. In this way, one edge is qualified to become a candidate edge as long as it can form a $(k-1)$-level triangle with unstable edges of $(k-1)$-light. Thus, we prune from C those edges that cannot form $(k-1)$-level triangles with unstable edges.

Example 3. *Suppose that we want to enlarge the 4-truss of the graph in Figure 1. The edge (v_8,v_9) is a stable edge in 3-truss with $|\Delta^3_{(v_8,v_9)}| = |\{\triangle_{v_8v_9v_{11}}, \triangle_{v_8v_9v_{12}}\}| = 2 \geq 2$, however the edge (v_9,v_{10}) is unstable with $|\Delta^3_{(v_9,v_{10})}| = |\{\triangle_{v_9v_{10}v_{11}}\}| = 1 < 2$. Specifically, we list all the stable edges and unstable edges in Table 1.*

6 A COMPONENT-BASED APPROACH

In this section, we propose a component-based approach for k-truss maximization called CBTM, which improves the quality and efficiency of per-edge insertion method in Algorithm 1.

6.1 Solution Overview

We identify two limitations of Algorithm 1 as follows. First, it costs expensive to check and calculate newcomers for each candidate edge. Moreover, the size of candidate edges is enormous in a large sparse graph. Another limitation is that Algorithm 1 cannot consider to insert multiple edges simultaneously. The addition of the first candidate edge may make no difference on newcomer increment, but the addition of a few more candidate edges may brings lots of newcomer increment significantly. For example, consider the graph G in Figure 1(a). The number of newcomers is 0 by inserting an edge (v_1,v_5) into G. However, when we add two edges (v_1,v_5) and (v_2,v_6) at the same time, the number of newcomers is 11. This motivates us to improve the idea of edge insertions and design an improved algorithm of CBTM below.

An overview of CBTM. The general idea of CBTM is to convert the critical edges located in $(k-1)$-light and avoid the expensive computation of newcomers for each candidate edge. The whole algorithm consists of three key steps:

(1) $(k-1)$-light partition: This step partitions the subgraph of $(k-1)$-light into several components and imposes an edge connectivity constraint to improve the independence of the partition.
(2) Component-based k-truss estimation: This step explores the strategy to convert each component to a k-truss. Due to the high connectivity of k-truss and multiple insertions, we make an estimation of newcomers for each component. We propose efficient insertion strategies that consume a small budget to complete the conversion.
(3) Dynamic programming based edges selection: Based on the above exploration of insertion strategy, we obtain solutions for each component. This step selects the $(k-1)$-light components for edge insertions. We use the dynamic programming optimizations to maximize the total number of newcomers by inserting candidate edges into $(k-1)$-light components. Finally, we compute the precise newcomers for the selected candidate edges.

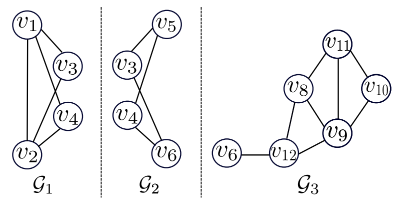


Figure 4: The split components of graph G in Figure 1

6.2 Three Key Steps

Partition $(k-1)$-light into components. It is observed that candidate edges may fall in the same local neighborhood of G and lead to the same newcomers. For example, (v_5,v_{12}) and (v_7,v_{12}) bring the same result of newcomers in Figure 1, i.e., $\mathsf{f}(\{(v_5,v_{12})\},4) = \mathsf{f}(\{(v_7,v_{12})\},4) = \{(v_6,v_{12}),(v_8,v_{12})\}$. Recall that the process of computing newcomers of a given candidate edge e is to traverse the edges from e based on the $(k-1)$-level triangle connectivity [19]. Obviously, the visited edges can compose a connected component. The unvisited edges can not be affected by the candidate edges. The trussness of all the visited edges is no less than $k-1$. Thus, the component can be seen as a partition of the $(k-1)$-truss.

Based on this observation, we decompose $(k-1)$-light L_{k-1} into multiple components as $\mathcal{P}(L_{k-1}) = \{\mathcal{G}_1,\dots,\mathcal{G}_h\}$ where $\bigcup_{i=1}^h \mathcal{G}_i = L_{k-1}$. The partition rule requires that any two edges should belong to the same triangle or can be reachable from each other through a series of adjacent $(k-1)$-level triangles in the $(k-1)$-light component. The essential advantage of the proposed partition rule is that even if one component $\mathcal{G}_i$ fails to form a new k-truss by candidate edges, it does not affect other components $\mathcal{G}_j \in \mathcal{P}(L_{k-1})$. Algorithm 3 depicts the procedure of $(k-1)$-light partition. The graph G in Figure 1 can be split into three components as shown in Figure 4.

Component-based k-truss estimation. We have identified the keys of a collapse in T_{k-1} when computing k-truss in Section 5.3, i.e, the unstable edges. Compared with the stable edges, they only lack one $(k-1)$-level triangle support.

The general idea is to record the unstable edges that belong to the same triangle with the candidate edges. After that, we recursively choose the best candidate edge to convert the most unstable edges in the component until all the unstable edges have a new triangle with

Algorithm 3 $(k-1)$–Light Partition

```
Input: (k − 1)-light $L_{k-1}$.
Output: the split components $\mathcal{P}(L_{k-1}) = \{\mathcal{G}_1, \ldots, \mathcal{G}_h\}$ of $L_{k-1}$.
1: Initialization: $h \leftarrow 0$, Queue $Q \leftarrow \emptyset$;
2: Mark all edges in $L_{k-1}$ as unvisited;
3: for edge $e = (u, v) \in L_{k-1}$ and $e$ is unvisited do
4:     $h \leftarrow h + 1$; $\mathcal{G}_h \leftarrow \emptyset$;
5:     $Q \leftarrow Q \cup \{(u, v)\}$;
6:     while $Q \neq \emptyset$ do
7:         $e = (u, v) \leftarrow$ Q.pop();
8:         for each $w \in N(u) \cap N(v)$ do
9:             if $\tau((w, u)) \geq k - 1$ and $\tau((w, v)) \geq k - 1$ then
10:                if $\tau((w, u)) = k - 1$ then
11:                    $\mathcal{G}_h \leftarrow \mathcal{G}_h \cup \{(w, u)\}$;
12:                    mark $(w, u)$ as visited;
13:                    $Q \leftarrow Q \cup \{(w, u)\}$;
14:                if $\tau((w, v)) = k - 1$ then
15:                    $\mathcal{G}_h \leftarrow \mathcal{G}_h \cup \{(w, v)\}$;
16:                    mark $(w, v)$ as visited;
17:                    $Q \leftarrow Q \cup \{(w, v)\}$;
18:     $\mathcal{P}(L_{k-1}) \leftarrow \mathcal{P}(L_{k-1}) \cup \mathcal{G}_h$;
19: return $\mathcal{P}(L_{k-1}) = \{\mathcal{G}_1, \ldots, \mathcal{G}_h\}$;
```

weight $k-1$. Note that other edges whose trussness are less than $k-1$ may also become newcomers under multiple insertions. Thus, We make an estimation of k-truss maximization in each component $\mathcal{G}_i \in \mathcal{P}(L_{k-1})$.

Algorithm 4 outlines the details of k-truss estimation in a component $\mathcal{G}_i$. The algorithm first collects E_u of unstable edges in $\mathcal{G}_i$ (line 2). For a unstable edge $e \in E_u$, if there is a candidate edge that can form a $(k-1)$-level triangle with e, it can convert e to a stable edge. Similar with Algorithm 2, we find candidate edges C_i, and additionally use an hash table $P(e)$ for $e \in C_i$ to keep records of those unstable edge that can be converted into stable edge by e (line 3). For example in Figure 4, we have $P(v_8, v_{10}) = \{(v_8, v_{11}), (v_9, v_{10}), (v_{10}, v_{11})\}$, $P(v_{11}, v_{12}) = \{(v_8, v_{11}), (v_9, v_{12})\}$ for component $\mathcal{G}_3$.

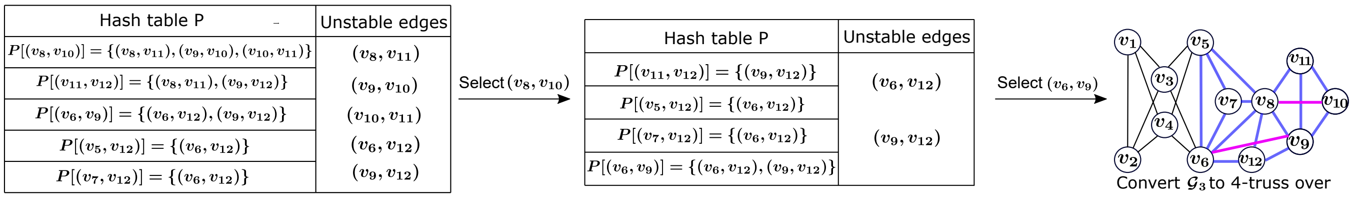
However, due to the candidate weight constraint and the graph topology, it may be not possible to insert candidate edges in any graph location. In other words, unstable edges may be not fully converted into stable edges in some components (lines 6-8). They lead to a cascade of support decrease, thus the selected candidate edges for this component are wasted. To address this issue, we have a pruning strategy to refine $\mathcal{G}_i$ into a smaller component (line 7). Specifically, we invoke a peeling procedure to keep removing from $\mathcal{G}_i$ those unstable edges E_u, which cannot be converted. After that, we recalculate the unstable edges in the refined component. The removed edges may lead to a k-truss collapse and generate some new unstable edges. Then we find new candidate edges to convert. The above process can be recursively invoked until we convert all the unstable edges in the refined component as $E_u = \emptyset$. Note that if one component $\mathcal{G}_i$ cannot convert all the unstable edges into k-truss or the size of $\mathcal{S}_i$ is larger than b, we return the empty result (line 12). Finally, the algorithm returns the selected candidate edges $\mathcal{S}_i$ and the corresponding newcomers $\mathcal{N}_i$ in $\mathcal{G}_i$ (line 14).

Algorithm 4 Component-based Truss Estimation

```
Input: A component $\mathcal{G}_i$.
Output: $\mathcal{S}_i$: the selected candidate edges in $\mathcal{G}_i$, $\mathcal{N}_i$: the estimated
    newcomers for $\mathcal{S}_i$ in $\mathcal{G}_i$.
1: Initialization: $C_i \leftarrow \emptyset, \mathcal{S}_i \leftarrow \emptyset, \mathcal{N}_i \leftarrow \emptyset$ ;
2: $E_u \leftarrow$ Compute all the unstable edges of $\mathcal{G}_i$;
3: $C_i \leftarrow$ Compute candidate edges for $E_u$ using a variant of Algo-
   rithm 2 by keeping records of $P(e)$ for $e \in C_i$;
4: while $E_u \neq \emptyset$ do
5:     $e^* \leftarrow \arg\max_{e \in C_i} |P(e)|$;
6:     if $P(e^*) = \emptyset$ then
7:         Remove the unstable edges $E_u$ from $\mathcal{G}_i$;
8:         goto line 1 of this algorithm;
9:     $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{e^*\}$ , $C_i \leftarrow C_i \setminus \{e^*\}$;
10:    $E_u \leftarrow E_u \setminus P(e^*)$;
11:    $P(e) \leftarrow P(e) \setminus P(e^*), \forall e \in C_i$;
12:    if $|\mathcal{S}_i| > b$ then return $\emptyset$;
13: $\mathcal{N}_i \leftarrow \mathcal{G}_i \cup \mathcal{S}_i$;
14: return $\{(\mathcal{S}_i, \mathcal{N}_i)\}$;
```

Example 4. *For the component $\mathcal{G}_3$ in Figure 4, we first find the candidate edges and create the hash table to maintain the unstable edges that the candidate edges can convert. The hashtable after initialization is shown on the left of Figure 5. According to the Algorithm 4, it first collects the candidate edge (v_8, v_{10}) into $\mathcal{S}_3$ since it can convert the most unstable edges. Next, it updates the hashtable by removing the unstable edges that have been converted by (v_8, v_{10}). The candidate edge chosen in the second iteration is (v_6, v_9). There are no remaining unstable edges after the selection of the above two candidate edges. Thus, the whole edges in the component as well as the selected candidate edges are the newcomers. Finally, the algorithm returns the conversion solution of this component.*

Dynamic programming based budget assignment. Based on the estimation of the selected candidates and newcomers in all components of L_{k-1}, we are going to distribute b budget of edge insertions for different components to maximize the total newcomers. The objective function of our candidate assignment is formulated as maximizing $\sum_{j=1}^{h} |\mathcal{N}_j| x_j$, where the variables $x_j \in \{0, 1\}$ for $j \in [1, h]$ and $\sum_{j=1}^{h} |\mathcal{S}_j| x_j \leq b$. For a component $\mathcal{G}_i$, $x_i = 1$ indicates that it selects the $\mathcal{S}_i$ candidate edges for insertions, which costs a budget of $|\mathcal{S}_i|$ and achieves a gain of $|\mathcal{N}_i|$ k-truss newcomers; otherwise, it takes the zero budget and zero gain for $x_i = 0$. We adopt a similar dynamic programming solution with the 0-1 knapsack problem [16], but our problem is more challenging due to the dependent property of candidate edges. For example, consider the two components $\mathcal{G}_1$ and $\mathcal{G}_2$ in Figure 4. The edge (v_3, v_4) belongs to a candidate edge of $\mathcal{G}_1$ and $\mathcal{G}_2$ simultaneously. Thus, we need to ensure the edge (v_3, v_4) that costs one budget of edge insertions.



Hash table P	Unstable edges
$P[(v_8,v_{10})] = \{(v_8,v_{11}),(v_9,v_{10}),(v_{10},v_{11})\}$	(v_8,v_{11})
$P[(v_{11},v_{12})] = \{(v_8,v_{11}),(v_9,v_{12})\}$	(v_9,v_{10})
$P[(v_6,v_9)] = \{(v_6,v_{12}),(v_9,v_{12})\}$	(v_{10},v_{11})
$P[(v_5,v_{12})] = \{(v_6,v_{12})\}$	(v_6,v_{12})
$P[(v_7,v_{12})] = \{(v_6,v_{12})\}$	(v_9,v_{12})

Hash table P	Unstable edges
$P[(v_{11},v_{12})] = \{(v_9,v_{12})\}$	(v_6,v_{12})
$P[(v_5,v_{12})] = \{(v_6,v_{12})\}$	
$P[(v_7,v_{12})] = \{(v_6,v_{12})\}$	(v_9,v_{12})
$P[(v_6,v_9)] = \{(v_6,v_{12}),(v_9,v_{12})\}$	

Figure 5: An example of the component-based k-truss estimation.

Algorithm 5 CBTM

```
Input: Graph $G(V,E)$, truss value $k$, budget $b$
Output: a set $E_\Delta$ of newly inserted edges
1: $L_{k-1} \leftarrow$ the subgraph of G induced by $(k-1)$-light;
2: Initialization: $|E_\Delta| \leftarrow \emptyset$;
3: Apply Algorithm 3 to partition $L_{k-1}$ into multiple components
   $\mathcal{P}(L_{k-1}) = \{\mathcal{G}_1,\dots,\mathcal{G}_h\}$ by $(k-1)$-level triangle adjacency;
4: for each component $\mathcal{G}_i \in \mathcal{P}(L_{k-1})$ do
5:     Apply Algorithm 4 on component $\mathcal{G}_i$;
6: $x \leftarrow |\mathcal{P}(L_{k-1})|$, $y \leftarrow b - |E_\Delta|$;
7: $f(i,j) \leftarrow 0$ for all $0 \le i \le x$, $0 \le j \le y$;
8: for $i \leftarrow 1$ to $x$ do
9:     for $j \leftarrow 1$ to $y$ do
10:        if $j < |\mathcal{S}_i|$ then
11:            $f(i,j) = f(i-1,j)$;
12:        else
13:            $f(i,j) = max(f(i-1,j), f(i-1, j-|\mathcal{S}_i|) + |\mathcal{N}_i|)$;
14: $x_i \leftarrow 1$, $\mathcal{P}(L_{k-1}) \leftarrow \mathcal{P}(L_{k-1}) \setminus \mathcal{G}_i$, if component $\mathcal{G}_i$ is chosen;
15: $E_\Delta \leftarrow \cup \mathcal{S}_i$, where $x_i = 1$ for $i \in [1,h]$;
16: if there are duplicate edges in $E_\Delta$ then
17:     Remove duplicate edges and goto line 6 of this algorithm;;
18: Compute the $k$-truss newcomers: $\mathsf{f}(E_\Delta,k) \leftarrow |T_k^*| - |T_k|$;
19: return $E_\Delta$;
```

6.3 CBTM Algorithm

Integrating with all the above techniques, the CBTM algorithm is presented in Algorithm 5. The algorithm first partitions the $(k-1)$-light L_{k-1} into multiple components $\mathcal{P}(L_{k-1}) = \{\mathcal{G}_1,\dots,\mathcal{G}_h\}$ (lines 1-3). It then applies Algorithm 4 to effectively select candidate edges and give estimation of newcomers on each component $\mathcal{G}_i \in \mathcal{P}(L_{k-1})$ (lines 4-5). Next, the algorithm applies a variant of 0-1 knapsack dynamic programming technique [16] to tackle our optimization function (lines 6-17). Finally, the algorithm returns the answer of $|E_\Delta|$ and corresponding the k-truss newcomers of $\mathsf{f}(E_\Delta,k)$ (lines 18-19).

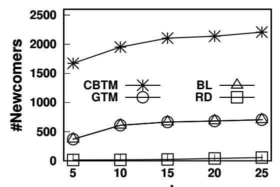
7 EXPETRIMENT

In this section, we conduct experiments to evaluate the effectiveness and efficiency of our proposed algorithms.

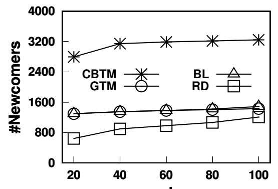
Datasets. We use nine real-world graph datasets. Syracuse56 is from [34], and the others are from SNAP [28]. Table 2 reports the network statistics of all graph datasets, listed in increasing order of their edge numbers.

Compared Algorithms. To our best knowledge, there is no existing studies for the k-truss maximization problem. We compare our proposed algorithms with one baseline method RD below. Besides RD, we also test other edge insertion methods, e.g., inserting the edges to form triangles with the edges whose trussness less than $k-1$ or connecting vertices with high degree. However, these methods have poor quality performance with only a few newcomers. This result also shows that identifying high-quality candidate edges is a very critical step in the k-truss maximization problem. Thus, we implement and compare four algorithms as follows:

- RD: a method selects b edges from C randomly.
- BL: the per-edge insertion greedy method in Algorithm 1.
- GTM: an improved method of Algorithm 1 using candidate pruning optimizations in Section 5.3.
- CBTM: our component-based method in Algorithm 5.



(a) Twitter



(b) Syracuse56

Figure 6: Quality evaluation by varying b.

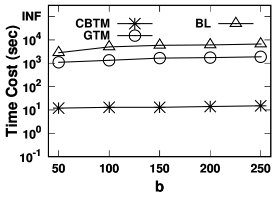
To evaluate the effectiveness, we use two metrics as the number of candidate edges and k-truss newcomers. We compare the algorithm efficiency in terms of running time (in seconds). By default, we set $k = 20$ for the first five small graphs, and $k = 40$ for the last four large graphs. We set the budget $b = 200$. We treat the running time as infinite if the algorithm runs exceeding 48 hours.

Exp-1: Evaluation of different algorithms on all datasets. Table 2 reports the effectiveness and efficiency results of different algorithms on all datasets. Our method CBTM outperforms all the other algorithms on all datasets by achieving the smallest candidate edges and the highest newcomers using the least running time. This reflects the superiority of $(k-1)$-light partition and dynamic programming techniques of candidate assignments by CBTM. The worse performance of GTM may be caused by only inserting one edge at a time for calculation of newcomers, which is equivalent to only converting the unstable edges within one-hop neighborhood of the candidate, and the remaining unstable edges will cause collapse and bring negative effects. Moreover, GTM and BL achieve the competitive results of k-truss newcomers in contrast to CBTM, as all algorithms attempt to convert the same edge pool of L_{k-1} to k-truss. The effectiveness of RD is also competitive, indicating that the candidate edges we located by Algorithm 2 are very effective. In addition, CBTM and GTM have the same number of candidates, which is much less than the candidate edges of BL, thanks to our pruning techniques.

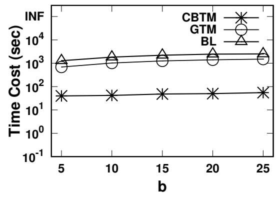
Exp-2: Effectiveness evaluation by varying budget b. We vary the budget b to evaluate the effectiveness of all algorithms. Figure 6 reports the results of newcomers on two graphs with $k = 45$ and the increased budget b, all methods achieve an improved result of

Table 2: Network statistics of nine graph datasets. $|\mathcal{P}(L_{k-1})|$ is the number of the components. We report the number of the candidate edges, newcomers and the running time of the proposed algorithms.

Dataset	$\|V\|$	$\|E\|$	k_{max}	$\|\mathcal{P}(L_{k-1})\|$	Candidate edges		Newcomers				Running time(s)		
					BL	GTM/CBTM	RD	BL	GTM	CBTM	BL	GTM	CBTM
Facebook	4,039	88,234	97	100	23,837	**10,898**	647	1,169	1,169	**1,845**	1,304	549	**6**
Enron	36,692	183,831	22	9	22,105	**7,898**	1,765	2,832	2,667	**3,858**	16,621	8,339	**45**
Brightkite	58,228	214,078	43	55	12,648	**5,794**	645	902	878	**989**	279	100	**15**
Syracuse56	13,654	543,982	59	354	273,786	**81,287**	1,277	-	7,241	**7,261**	-	97,702	**164**
Gowalla	196,591	950,327	29	110	46,850	**22,397**	1,633	3,881	3,819	**4,439**	135,970	75,757	**183**
Twitter	81,306	1,768,149	82	82	1,533,582	**587,476**	2,102	-	4,233	**6,986**	-	30,502	**312**
Stanford	281,903	2,312,497	62	655	135,088	**102,164**	3,186	4,026	4,021	**4,200**	10,080	2,993	**133**
wiki-Talk	2,394,385	5,021,410	53	115	173,865	**97,062**	1,412	-	-	**5,400**	-	-	**1,402**
LiveJournal	3,997,962	34,681,189	352	433	94,022	**65,730**	1,369	-	12,969	**16,393**	-	15,829	**748**

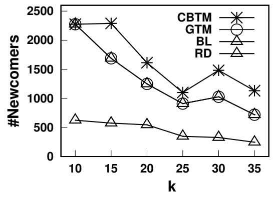


(a) Brightkite

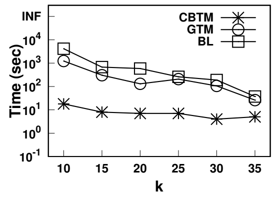


(b) Twitter

Figure 7: Efficiency evaluation by varying b.



(a) Facebook



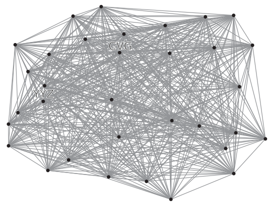
(b) Brightkite

Figure 8: Quality and efficiency evaluation by varying k.

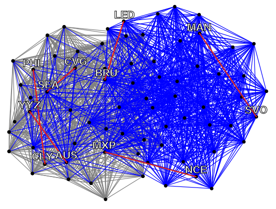
k-truss newcomers generally. CBTM wins the most cases among all methods. The increasing budget b has a bottleneck when it is large enough, the number of newcomers does not increase significantly. This is because the edges in L_{k-1} in the graph have basically been updated to k-truss. GTM and BL have similar performance on newcomers, because GTM's techniques mainly aimed at accelerating the efficiency.

Exp-3: Efficiency evaluation by varying b. We evaluate the efficiency of three algorithms BL, GTM, and CBTM by varying b. We set $k = 45$ and $k = 10$ on Twitter and Brightkite, respectively. The results of running time are shown in Figure 7. CBTM runs much faster than BL and GTM. Moreover, CBTM is scalable well with the increased b, which achieves the stable performance of running time. This is because CBTM avoids the expensive computation of newcomers for each candidate edge. On the other hand, GTM runs faster than BL, which validates the effectiveness of the pruning optimizations in Section 5.3.

Exp-4: Effectiveness and efficiency evaluation by varying k. We vary parameter k to evaluate the the proposed algorithms with $b = 100$. The results of newcomers and running time are shown in Figure 8(a) and 8(b), respectively. Once again, CBTM achieves the best performance, and RD performs in worst. We can see that as k increases, the number of newcomers by GTM and CBTM generally has a decreasing trend in Figure 8(a). This is because there are fewer edges in $(k-1)$-light with a larger k, in which the size of L_{k-1} directly determines the upper bound of newcomers for a small number b. Also, CBTM is the fastest method under different k in Figure 8(b).



(a) T_k



(b) T_k^*

Figure 9: A case study of truss maximization on a flight network. Here, $k = 26$ and $b = 5$.

Exp-5: Case study on flight networks. We construct a flight network from an openflights dataset.[1] A vertex represents an airport. An edge is added between two airports if there is at least one airline between them. We apply CBTM on this flight network to enlarge k-truss by adding a few new edges to increase the route connectivity. Here, we set k as the largest trussness $k = 26$ and the budget $b = 5$. The origin k-truss T_k is shown in Figure 9(a). CBTM inserts a set of new edges E_Δ = {(YYZ, AUS), (BRU, LED), (MAN, SVO), (NCE, MXP), (CVG, SEA) } in red, and returns a much larger k-truss T_k^* with 692 newcomers marked in blue, as shown in Figure 9(b). The abbreviation denotes a 3-letter (IATA) code of airport, e.g., MAN denotes the Manchester Airport in England.[2]

8 CONCLUSION

In this paper, we formulate and study the budget-constrained k-truss maximization problem, which finds b new edges to be inserted into a graph for enlarging k-truss as large as possible. Due to its problem NP-hardness, we first propose a greedy algorithm and further improve the efficiency by pruning optimizations. Furthermore, we propose a component-based dynamic programming algorithm to effectively use a budget of b new edges in a balanced way. Extensive experimental results on large networks validate the effectiveness and efficiency of our proposed algorithms.

ACKNOWLEDGMENTS

This work was supported by the Natural Science Foundation of China under grants 61772361, 61876128, and HK RGC Grant No. 22200320. Di Jin is the corresponding author.

[1]https://openflights.org/data.html
[2]https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat

REFERENCES

[1] Vinti Agarwal and Kamal K Bharadwaj. 2011. Trust-enhanced recommendation of friends in web based social networks using genetic algorithms to learn user preferences. In *International Conference on Computational Science, Engineering and Information Technology*. 476–485.

[2] J Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. 2006. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in neural information processing systems*. 41–50.

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

[4] Yulin Che, Zhuohang Lai, Shixuan Sun, Yue Wang, and Qiong Luo. 2020. Accelerating truss decomposition on heterogeneous processors. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1751–1764.

[5] Pei-Ling Chen, Chung-Kuang Chou, and Ming-Syan Chen. 2014. Distributed algorithms for k-truss decomposition. In *IEEE International Conference on Big Data*. 471–480.

[6] Rajesh Chitnis and Nimrod Talmon. 2018. Can we create large k-cores by adding few edges?. In *International Computer Science Symposium in Russia*. 78–89.

[7] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16 (2008), 3–29.

[8] Federico Corò, Gianlorenzo D'Angelo, and Cristina M Pinotti. 2020. Adding Edges for Maximizing Weighted Reachability. *Algorithms* 13, 3 (2020), 68.

[9] Safaa Diab, Mhd Ghaith Olabi, and Izzat El Hajj. 2020. KTRussExPLORER: Exploring the Design Space of K-truss Decomposition Optimizations on GPUs. In *IEEE High Performance Extreme Computing Conference*. 1–8.

[10] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. 2007. Extraction and classification of dense communities in the web. In *WWW*. 461–470.

[11] Soroush Ebadian and Xin Huang. 2019. Fast algorithm for k-truss discovery on public-private graphs. *IJCAI* (2019), 2258–2264.

[12] Fatemeh Esfahani, Jian Wu, Venkatesh Srinivasan, Alex Thomo, and Kui Wu. 2019. Fast Truss Decomposition in Large-scale Probabilistic Graphs.. In *EDBT*. 722–725.

[13] Facebook. 2020. How does facebook suggest friends for me? *https://www.facebook.com/help/1059270337766380* (2020).

[14] Eugene Fratkin, Brian T Naughton, Douglas L Brutlag, and Serafim Batzoglou. 2006. MotifCut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics* 22, 14 (2006), e150–e157.

[15] Zakariya Ghalmane, Mohammed El Hassouni, Chantal Cherifi, and Hocine Cherifi. 2018. K-truss decomposition for modular centrality. In *IEEE International Symposium on Signal, Image, Video and Communications*. 241–248.

[16] Olivier Goldschmidt, David Nehme, and Gang Yu. 1994. Note: On the set-union knapsack problem. *Naval Research Logistics (NRL)* 41, 6 (1994), 833–842.

[17] Jinbin Huang, Xin Huang, and Jianliang Xu. 2021. Truss-based Structural Diversity Search in Large Graphs. *TKDE* (2021).

[18] Sitao Huang, Mohamed El-Hadedy, Cong Hao, Qin Li, Vikram S Mailthody, Ketan Date, Jinjun Xiong, Deming Chen, Rakesh Nagi, and Wen-mei Hwu. 2018. Triangle counting and truss decomposition using fpga. In *IEEE High Performance extreme Computing Conference*. 1–7.

[19] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1311–1322.

[20] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[21] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2019. *Community Search over Big Graphs*. Morgan & Claypool Publishers.

[22] Xin Huang, Wei Lu, and Laks VS Lakshmanan. 2016. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *SIGMOD*. 77–90.

[23] Di Jin, Cuiying Huo, Chundong Liang, and Liang Yang. 2021. Heterogeneous Graph Neural Network via Attribute Completion. In *Proceedings of the Web Conference 2021*. 391–400.

[24] Di Jin, Zhizhi Yu, Pengfei Jiao, Shirui Pan, Philip S Yu, and Weixiong Zhang. 2021. A survey of community detection approaches: From statistical modeling to deep learning. *arXiv preprint arXiv:2101.01669* (2021).

[25] Humayun Kabir and Kamesh Madduri. 2017. Shared-memory graph truss decomposition. In *IEEE International Conference on High Performance Computing*. 13–22.

[26] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

[27] Ricky Laishram, Ahmet Erdem Sar, Tina Eliassi-Rad, Ali Pinar, and Sucheta Soundarajan. 2020. Residual Core Maximization: An Efficient Algorithm for Maximizing the Size of the k-Core. In *SDM*. 325–333.

[28] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[29] Penghang Liu and A Erdem Sarıyüce. 2020. Characterizing and Utilizing the Interplay Between Core and Truss Decompositions. (2020), 957–962.

[30] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2183–2197.

[31] Sourav Medya, Tiyani Ma, Arlei Silva, and Ambuj Singh. 2020. K-core minimization: A game theoretic approach. *IJCAI* (2020), 3473–3479.

[32] Rafael T Mikolajczyk and Mirjam Kretzschmar. 2008. Collecting social contact data in the context of disease transmission: prospective and retrospective study designs. *Social Networks* 30, 2 (2008), 127–135.

[33] Robert J Mokken et al. 1979. Cliques, clubs and clans. *Quality & Quantity* 13, 2 (1979), 161–173.

[34] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. http://networkrepository.com

[35] Ryan A Rossi, David F Gleich, and Assefaw H Gebremedhin. 2015. Parallel maximum clique algorithms with applications to network analysis. *SIAM Journal on Scientific Computing* 37, 5 (2015), C589–C616.

[36] Rahmtin Rotabi, Krishna Kamath, Jon Kleinberg, and Aneesh Sharma. 2017. Detecting strong ties using network motifs. In *WWW Companion*. 983–992.

[37] Ahmet Erdem Sarıyüce, C Seshadhri, and Ali Pinar. 2017. Parallel local algorithms for core, truss, and nucleus decompositions. *arXiv. org e-Print archive, https://arxiv.org/abs/1704.00386* (2017).

[38] Ahmet Erdem Sariyuce, C Seshadhri, Ali Pinar, and Umit V Catalyurek. 2015. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*. 927–937.

[39] Stephen B Seidman and Brian L Foster. 1978. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology* 6, 1 (1978), 139–154.

[40] Zitan Sun, Xin Huang, Jianliang Xu, and Francesco Bonchi. 2021. Efficient probabilistic truss indexing on uncertain graphs. In *Proceedings of the Web Conference 2021*. 354–366.

[41] Jia Wang and James Cheng. 2012. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment* 5, 9 (2012).

[42] Qianqian Xu, Jiechao Xiong, Xiaochun Cao, Qingming Huang, and Yuan Yao. 2018. From social to individuals: a parsimonious path of multi-level models for crowdsourced preference aggregation. *IEEE transactions on pattern analysis and machine intelligence* 41, 4 (2018), 844–856.

[43] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and efficient truss computation over large heterogeneous information networks. In *ICDE*. 901–912.

[44] Ting Yuan, Jian Cheng, Xi Zhang, Qingshan Liu, and Hanqing Lu. 2015. How friends affect user behaviors? An exploration of social relation analysis for recommendation. *Knowledge-Based Systems* 88 (2015), 70–84.

[45] Baichuan Zhang, Tanay Kumar Saha, and Mohammad Al Hasan. 2014. Name disambiguation from link data in a collaboration graph. In *ASONAM*. 81–84.

[46] Fan Zhang, Conggai Li, Ying Zhang, Lu Qin, and Wenjie Zhang. 2018. Finding critical users in social communities: The collapsed core and truss problems. *IEEE Transactions on Knowledge and Data Engineering* 32, 1 (2018), 78–91.

[47] Fan Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, and Xuemin Lin. 2017. OLAK: an efficient algorithm to prevent unraveling in social networks. *Proceedings of the VLDB Endowment* 10, 6 (2017), 649–660.

[48] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2018. Efficiently reinforcing social networks over user engagement and tie strength. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 557–568.

[49] Yikai Zhang and Jeffrey Xu Yu. 2019. Unboundedness and efficiency of truss maintenance in evolving graphs. In *SIGMOD*. 1024–1041.

[50] Feng Zhao and Anthony KH Tung. 2012. Large scale cohesive subgraphs discovery for social network visual analysis. *Proceedings of the VLDB Endowment* 6, 2 (2012), 85–96.

[51] Zhongxin Zhou, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Chen Chen. 2019. K-Core Maximization: An Edge Addition Approach.. In *IJCAI*. 4867–4873.

[52] Weijie Zhu, Chen Chen, Xiaoyang Wang, and Xuemin Lin. 2018. K-core minimization: An edge manipulation approach. In *CIKM*. 1667–1670.

[53] Weijie Zhu, Mengqi Zhang, Chen Chen, Xiaoyang Wang, Fan Zhang, and Xuemin Lin. 2019. Pivotal relationship identification: the K-truss minimization problem. In *IJCAI*. 4874–4880.