Higher-Order Truss Decomposition in Graphs

Zi Chen, Long Yuan, Li Han, and Zhengping Qian

Abstract—k-truss model is a typical cohesive subgraph model and has been received considerable attention recently. However, the k-truss model only considers the direct common neighbors of an edge, which restricts its ability to reveal fine-grained structure information of the graph. Motivated by this, in this paper, we propose a new model named (k, τ)-truss that considers the higher-order neighborhood (τ hop) information of an edge. Based on the (k, τ)-truss model, we study the higher-order truss decomposition problem which computes the (k, τ)-trusses for all possible k values regarding a given τ. Higher-order truss decomposition can be used in the applications such as community detection and search, hierarchical structure analysis, and graph visualization. To address this problem, we first propose a bottom-up decomposition paradigm in the increasing order of k values to compute the corresponding (k, τ)-truss. Based on the bottom-up decomposition paradigm, we further devise three optimization strategies to reduce the unnecessary computation. We evaluate our proposed algorithms on real datasets and synthetic datasets, the experimental results demonstrate the efficiency, effectiveness and scalability of our proposed algorithms.

Index Terms—Higher-order neighborhood, truss decomposition, graph algorithm

1 Introduction

Graphs have been widely used to represent the relationships of entities in real-world applications [1], [2], [3]. With the proliferation of graph applications, plenty of research efforts have been devoted to cohesive subgraph models for graph structure analysis [4]. Typical cohesive subgraph models include clique [5], [6], [7], [8], k-clique [9], quasi-clique [10], [11], k-core [12], [13], [14], k-truss [15], [16], and k-ECC [17], [18].

Among them, the k-truss model is a typical cohesive subgraph model and has received considerable attention due to its unique cohesive properties on degree and bounded diameter [19], [20], [21], [22]. Given a graph G, for an edge $e = (u, v)$ in G, the support of e is defined as the number of direct common neighbors of u and v. k-truss is the maximal subgraph G' of G such that the support of each edge in G' is not less than $k - 2$ [15]. Truss decomposition computes the k-truss in the graph for all possible k values in G.

Motivation. Although the k-truss model and the corresponding truss decomposition have been successful in many applications, the model lacks the ability to reveal fine-grained structure information of the graph. Consider the graph in Fig. 1. Fig. 1 shows part of the collaboration network in the DBLP (https://dblp.uni-trier.de/), in which each node represents an author and each edge indicates the co-author relationship between two authors. The results of traditional truss decomposition are shown in Fig. 1. The traditional truss decomposition treats the whole graph as a 3-truss and is not able to provide more fine-grained structure information.

On the other hand, the importance of the higher-order neighborhood (multiple-hop neighbors instead of direct neighbors) on the characterization of complex network has been well established [23], [24], and remarkable results have been obtained in the network science due to the introduction of higher-order neighborhood [25], [26], [27], [28], [29]. Motivated by this, we propose the (k, τ)-truss model by incorporating the higher-order neighborhood into the k-truss model. Formally, given a graph G and an integer τ, for an edge $e = (u, v)$, the higher-order support of e is the number of τ-hop common neighbors of u and v. (k, τ)-truss is the maximal subgraph G' of G such that the higher-order support of each edge in G' is not less than $k - 2$. Following the (k, τ)-truss model, we study the higher-order truss decomposition problem that computes the (k, τ)-truss for all possible k values in G regarding a given τ.

The benefits of the (k, τ)-truss model are twofold: (1) it inherits the unique cohesive properties on degree and bounded diameter of k-truss, which are shown in Property 1 and Property 2 in Section 3. (2) It acquires the ability to reveal fine-grained structure information due to the introduction of higher-order neighborhood. Reconsider G in Fig. 1, Fig. 1 shows the higher-order truss decomposition results. By considering the higher-order neighborhood information, the whole graph(3-truss) can be further split into (4,2)-truss, (5,2)-truss, (6,2)-truss and (10,2)-truss. The hierarchy structure of the graph is clearly characterized by the higher-order truss decomposition. Note that a cohesive subgraph model named

- *Zi Chen and Li Han are with Software Engineering Institute, East China Normal University, Shanghai 200050, China. E-mail: {zchen, hanli}@sei.ecnu.edu.cn.*
- *Long Yuan is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, Jiangsu 210094, China. E-mail: longyuan@njust.edu.cn.*
- *Zhengping Qian is with Alibaba Group, Hangzhou 310052, China. E-mail: zhengping.qzp@alibaba-inc.com.*

Manuscript received 21 Oct. 2021; revised 15 Dec. 2021; accepted 19 Dec. 2021. Date of publication 23 Dec. 2021; date of current version 7 Mar. 2023.
The work of Zi Chen was supported in part by China Postdoctoral Science Foundation under Grant 2021M701214. The work of Long Yuan was supported in part by NSFC under Grant 61902184, in part by NSF of Jiangsu Province under Grant BK20190453, and in part by Science and Technology on Information Systems Engineering Laboratory under Grant WDZC20205250411. The work of Li Han was supported in part by Shanghai Sailing Program under Grant 21YF1411100.
(Corresponding authors: Long Yuan and Li Han.)
Recommended for acceptance by Y. Zhang.
Digital Object Identifier no. 10.1109/TKDE.2021.3137955

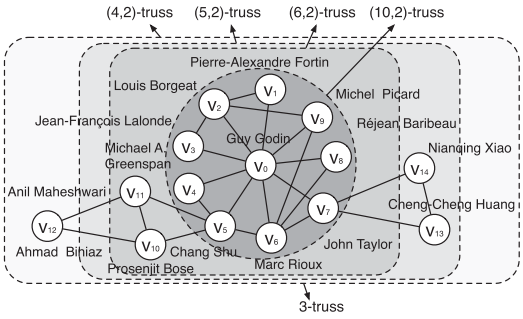

Fig. 1. Part of the collaboration network in DBLP.

(k,h)-core that also considers higher-order neighborhood information is studied in [27], [30]. However, the definition of our model is more rigorous, which makes our model can search "core" of a (k,h)-core (e.g. a (k+1,τ)-truss is a (k,h)-core but not vice versa,τ=h). As shown in our case study (Exp-5), our model has higher ability to reveal fine-grained structure information than (k,h)-core. *Applications.*Higher-order truss decomposition can be applied in the applications using the traditional k-truss decomposition since the k-truss model is a specific case of (k,τ)-truss model when $\tau = 1$, namely $(k,1)$-truss. These applications include community detection and search [20], [21]. Moreover, as the higher-order truss decomposition can reveal more fine-grained structure of graphs compared with the traditional truss decomposition as shown in Fig. 1, it can also be applied in the applications which focus on the hierarchical structure of graph, such like hierarchical structure analysis [31], [32], [33] and graph visualization [34], [35], [36]. Besides, due to consideration of the higher-order neighborhood with parameter τ, users can control the cohesiveness of decomposition result in a more flexible manner.

Challenges. To conduct the higher-order truss decomposition, we first propose a bottom-up decomposition paradigm by extending the peeling algorithm for the traditional truss decomposition [19]. It conducts the higher-order truss decomposition in the increasing order of k values. After computing the higher-order support for each edge, it iteratively removes the edge e with the minimum higher-order support in the graph and updates the higher-order support of the edges whose higher-order support may be changed due to the removal of e until the graph is empty.

Although the peeling algorithm is suitable for the traditional truss decomposition, following the above bottom-up decomposition paradigm directly cannot handle the higher-order truss decomposition efficiently. This is because, when an edge $e = (u, v)$ is removed, for the traditional truss decomposition, we just need to decrease the support of edges (u, w) and (v, w) by 1, where w is a common neighbor of u and v. However, for the higher-order truss decomposition, when e is removed, the scope of edges whose higher-order support may be changed due to the removal of e is enlarged to all the edges incident to u, v and their τ-hop common neighbors. Moreover, opposite to the traditional truss decomposition, we have no prior knowledge on the specific decreased value of the higher-order support of these edges. It means the higher-order support of these edges has to be recomputed based on its definition instead of just decreasing by 1 as in traditional truss decomposition, which is prohibitively costly. *The enlarged scope of influenced edges* and *the un-determination of the decreased higher-order support value* not only imply that the higher-order truss decomposition is harder than the traditional truss decomposition, but also are the reasons why following the above bottom-up decomposition paradigm directly is inefficient for the higher-order truss decomposition. *Our idea.* Revisiting the two reasons leading to the inefficiency of the bottom-up decomposition paradigm, for the un-determination of the decreased higher-order support value, it seems insoluable to obtain the higher-order support of an influenced edge without recomputation based on the definition. Hence, we focus on reducing the scope of influenced edges whose higher-order support has to be recomputed for each removal of an edge.

To achieve this goal, we follow the bottom-up decomposition paradigm. We define the higher-order truss number of an edge as the maximal value of k such that the edge is in the (k,τ)-truss, but not in the $(k+1,\tau)$-truss. When handling a specific k, we observe that for an edge with higher-order truss number bigger than k, the correctness of its higher-order support in the remaining graph does not affect the correctness of the higher-order truss number computation for the edges whose higher-order truss number is k. It means that recomputing the higher-order support of the edges with higher-order truss number bigger than k immediately after the removal of an edge is not necessary. Therefore, we propose a delayed update strategy and recompute the higher-order support when necessary. With this strategy, we can reduce the scope of the influenced edges whose higher-order support has to be recomputed. However, to fulfill this strategy, we have to know the higher-order truss number in prior, which is intractable. Consequently, we devise a tight lower bound of the higher-order truss number. When handling a specific k, we do not need to recompute the higher-order support for the edges whose lower bound of the higher-order truss number is bigger than k.

Moreover, we further explore two optimization strategies, namely early pruning strategy and unchanged support detection strategy, to further reduce the scope of the influenced edges. Experiments on real datasets show that our improved algorithm can achieve up to 4 orders of magnitude speedup compared with the baseline algorithm.

Contributions. We make the following contributions:

- *The first work to study the (k,τ)-truss model.* Motivated by the traditional k-truss model ignores the higher-order neighborhood information of an edge, we propose the (k,τ)-truss model. To the best of our knowledge, this is the first work considering the higher-order neighborhood information regarding the traditional k-truss model. Furthermore, we also prove the unique cohesive properties of the (k,τ)-truss model.
- *Efficient algorithms for the higher-order truss decomposition.* We first devise a bottom-up decomposition paradigm by extending the peeling algorithm for traditional k-truss decomposition. Based on the bottom-up paradigm, we propose three optimization strategies to further improve the decomposition efficiency. Moreover, considering that some applications are more interested in the (k,τ)-trusses with large k

values, we study the top r (k,τ)-trusses computation problem which returns the (k,τ)-trusses with top r k values in the graph. We also propose an efficient algorithm for this problem.

- *Extensive performance studies on real datasets and synthetic datasets.* We conduct extensive experimental studies on real datasets and synthetic datasets. For the efficiency, our improved algorithm can achieve up to 4 orders of magnitude speedup compared with the direct bottom-up decomposition paradigm. Besides, it also shows high effectiveness and scalability.

Outline. Section 2 provides the problem definition. Section 3 presents the theoretical cohesiveness properties of (k,τ)-truss model. Section 4 introduces the bottom-up decomposition paradigm. Section 5 presents our improved algorithms for higher-order truss decomposition problem. Section 6 presents our approach for top r (k,τ)-trusses computation problem which returns the (k,τ)-trusses with top r k values in the graph. Section 7 evaluates our algorithms and Section 8 reviews the related work. Section 9 concludes the paper.

2 Preliminaries

Given an undirected and unweighted graph $G=(V,E)$, where $V(G)$ and $E(G)$ represent the set of vertices and the set of edges in G, respectively, we use n and m to denote the number of vertices and the number of edges in G, i.e., $n=|V|, m=|E|$. In the graph G, a path is a sequence of vertices $p=(v_1,v_2,\ldots,v_j)$ where $(v_i,v_{i+1})\in E$ for each $1\le i<j$ and cycles are allowed on p. Given a path p, the path length of p, denoted by $l(p)$, is the number of edges on p. The shortest path between two vertices u and v in G is the path between these two vertices with the minimum length. We call the length of the shortest path between u and v is the distance of u and v, and denote it as $\mathsf{dis}_G(u,v)$. Given a graph G, the diameter of G, denoted by $\omega(G)$, is the maximum length of the shortest path between any pair of vertices in G. Given two vertices $u,v\in G$, u is τ-hop reachable from v, denoted by $v\rightarrow_\tau u$, if there is a path p between u and v with $l(p)\le\tau$. Since we consider the undirected graph in this paper, $v\rightarrow_\tau u$ if and only if $u\rightarrow_\tau v$. Given a vertex v, the τ-hop neighbors of v, denoted by $N_\tau(v,G)$, is the set of vertices u such that u is τ-hop reachable from v. Similarly, the τ-hop degree of v, denoted by $d_\tau(v,G)$, is the number of τ-hop neighbors of v, i.e., $d_\tau(v,G)=|N_\tau(v,G)|$.

Definition 2.1. (τ-Hop Common Neighbor) *Given a graph G and an integer τ, for an edge $e=(u,v)$ in G, w is a τ-hop common neighbor of e if $u\rightarrow_\tau w$ and $v\rightarrow_\tau w$.*

For an edge $e=(u,v)$, we use $\Delta_\tau(e,G)$ to denote the set of τ-hop common neighbors of e, i.e., $\Delta_\tau(e,G)=N_\tau(u,G)\cap N_\tau(v,G)$.

Definition 2.2. (Higher-Order Edge Support) *Given a graph G and an integer τ, for an edge e, the higher-order support of e, denoted by $\mathsf{sup}_\tau(e,G)$, is the number of τ-hop common neighbors of e, i.e., $\mathsf{sup}_\tau(e,G)=|\Delta_\tau(e,G)|$.*

Definition 2.3. ((k,τ)-Truss). *Given a graph G and an integer τ, a (k,τ)-truss is a maximal subgraph G' of G such that $\mathsf{sup}_\tau(e,G')\ge k-2$ for all $e\in E(G')$ and no more edges can be added into G'.*

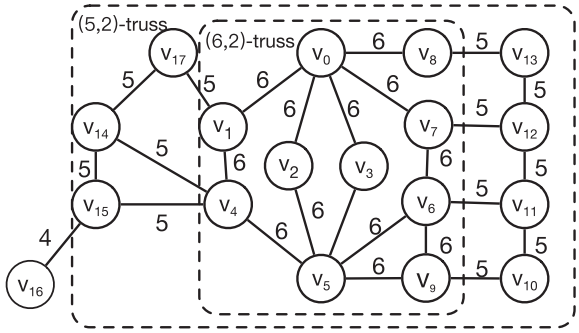

Fig. 2. An example of higher-order truss ($\tau=2$).

Definition 2.4. (Higher-Order Truss Number) *Given a graph G and an integer τ, for an edge e, the higher-order truss number of e, denoted by $\phi_\tau(e,G)$, is the maximum value of k such that e is contained in the corresponding (k,τ)-truss.*

Problem Statement. In the applications, the higher-order truss number of each edge is required. Given a graph G and an integer τ, in this paper, we study the higher-order truss decomposition problem which aims to compute the (k,τ)-trusses of G for all possible k values regarding τ. Straightforwardly, the (k,τ)-truss of G consists of the set of edges with higher-order truss number at least k, i.e., $\cup_{\phi_\tau(e,G)\ge k}e$. Therefore, the higher-order truss decomposition is equivalent to compute the higher-order truss number for each edge in G.

Example 2.1. Consider G in Fig. 2and assume $\tau=2$ (in the following examples, we always assume $\tau=2$). For v_0, its 2-hop neighbors $N_2(v_0,G)=\{v_1,\ldots,v_8,v_{12},v_{13},v_{17}\}$. For v_1, its 2-hop neighbors $N_2(v_1,G)=\{v_0,v_2,\ldots,v_5,v_7,v_8,v_{14},v_{15},v_{17}\}$. Therefore, $\Delta_2((v_0,v_1),G)=N_2(v_0,G)\cap N_2(v_1,G)=\{v_2,\ldots v_5,v_7,v_8,v_{17}\}$, $\mathsf{sup}_2((v_0,v_1),G)=7$. $\mathsf{sup}_2(e,G)$ for each edge e can be computed similarly and we can find that $\mathsf{sup}_2(e,G)\ge 2$. Thus, G itself is a $(4,2)$-truss. Moreover, we can find that the subgraph G' induced by $\{v_0,\ldots,v_{15}\}$ is a $(5,2)$-truss as $\mathsf{sup}_2(e,G')\ge 3$ for each edge e in G' and no more edges can be added to G' to make it as a bigger $(5,2)$-truss. $\phi_2((v_{15},v_{16}),G)$ is 4 as (v_{15},v_{16}) is contained in the $(4,2)$-truss but not in the $(5,2)$-truss. The value of $\phi_2(e,G)$ is shown near each edge in Fig. 2. The hierarchy structure of G is clearly illustrated by the value of $\phi_2(e,G)$.

3 Theoretical Properties of (k,τ)-Truss

Although τ-hop neighborhood is considered, (k,τ)-truss still have the cohesiveness properties on degree and diameter:

Property 1. (Minimum Degree) *Given a (k,τ)-truss G' of G, for each vertex $v\in V(G')$, $d_\tau(v,G')\ge k-1$.*

Proof. For a vertex v, let $e=(u,v)$ be an edge incident to v in G'. Based on Definition 2.3, we have $\mathsf{sup}_\tau(e,G')\ge k-2$. According to Definition 2.2, v has at least $k-2$ τ-hop neighbors in $\Delta_\tau(e,G)$. Moreover, u is a 1-hop neighbor of v. Thus, $d_\tau(v,G')\ge k-1$. □

Property 2. (Bounded Diameter) *Given a (k,τ)-truss G' of G, for each connected component G'' of G', the diameter of G'' $\omega(G'')\le\frac{2\tau(|V(G'')|-1)}{k}$.*

Proof. Without loss of generality, let $p = (v_1, v_2, ..., v_d)$ be the shortest path in G'' with $l(p) = \omega(G'')$. We use $V(p)$ to denote the set of vertices on p. For an edge $e_i = (v_i, v_{i+1})$ on p, we use Γ_i to denote the set of vertices in $\Delta_\tau(e_i, G'') \setminus V(p)$. According to Definition 2.1, it is clear that $|\Delta_\tau(e_i, G'') \cap V(p)| \leq 2\tau - 2$. Since $\Gamma_i = \Delta_\tau(e_i, G'') \setminus (\Delta_\tau(e_i, G'') \cap V(p))$, $|\Gamma_i| = |\Delta_\tau(e_i, G'')| - |\Delta_\tau(e_i, G'') \cap V(p)|$. According to Definition 2.3, $|\Delta_\tau(e_i, G'')| \geq k - 2$. Together with $|\Delta_\tau(e_i, G'') \cap V(p)| \leq 2\tau - 2$, we have $|\Gamma_i| \geq k - 2\tau$.

Meanwhile, for a vertex $w \in V(G'') \setminus V(p)$, we have $|\{\Gamma_i : w \in \Gamma_i, 1 \leq i \leq d-1\}| \leq 2\tau$. This can be proved by contraction. Assume that $|\{\Gamma_i : w \in \Gamma_i, 1 \leq i \leq d-1\}| \geq 2\tau + 1$, let E_Γ be the set of edges on p such that w is a τ-hop common neighbor. Based on the assumption, $|E_\Gamma| \geq 2\tau + 1$. Let $e_j = (v_j, v_{j+1})$ and $e_k = (v_k, v_{k+1})$ be two edges in E_Γ such that the edges on p between e_j and e_k are maximum. We can derive that the length of the sub-path from v_j to v_{k+1} on p is at least $2\tau + 1$ due to $|E_\Gamma| \geq 2\tau + 1$. On the other hand, since w is a τ-hop common neighbors of e_j and e_k, there exists a path p' from v_j to v_{k+1} through w with length less than 2τ. It contradicts with the assumption that p is a shortest path from v_1 to v_d as we can replace the sub-path from v_j to v_{k+1} on p with p' to obtain a shorter path. Therefore, $|\{\Gamma_i : w \in \Gamma_i, 1 \leq i \leq d-1\}| \leq 2\tau$. Based on this, we can derive that $|\Gamma_1 \cup \Gamma_2 ... \cup \Gamma_{d-1}| \geq \frac{(d-1)(k-2\tau)}{2\tau}$. Since $V(G'') = \Gamma_1 \cup \Gamma_2 ... \cup \Gamma_{d-1} \cup V(p)$, $|V(G'')| = |\Gamma_1 \cup \Gamma_2 ... \cup \Gamma_{d-1}| + |V(p)|$, we can derive that $|V(G'')| \geq \frac{(d-1)(k-2\tau)}{2\tau} + d$. As $\omega(G'') = d - 1$, $\omega(G'') \leq \frac{2\tau(|V(G'')|-1)}{k}$, the property holds. □

4 A Bottom-Up Decomposition Paradigm

In this section, we present a bottom-up decomposition algorithm for higher-order neighborhood truss decomposition, which is based on the following lemma:

Lemma 4.1. *Given a graph G and an integer τ, a $(k+1, \tau)$-truss is contained by a (k, τ)-truss.*

Proof. According to Definition 2.3, for each edge e in a $(k+1, \tau)$-truss G', $\mathsf{sup}_\tau(e, G') \geq k - 1 \geq k - 2$. It is clear that G' is also a (k, τ)-truss. □

Based on Lemma 4.1, for a given graph G, we can decompose G in the increasing order of k. For a specific k, we compute the edges whose higher-order truss number is identical to k. As $(k+1, \tau)$-truss is contained in the (k, τ)-truss, according to Definition 2.3, we can remove these edges from G and get the $(k+1, \tau)$-truss. We continue the procedure and the higher-order truss number for each edge can be obtained when all the edges are removed. The pseudocode of the decomposition algorithm is shown in Algorithm 1 *Algorithm*. Following the above idea, the paradigm to conduct the higher-order truss decomposition, HOTDecom, is shown in Algorithm 1. Given a graph G and an integer τ, it first computes the higher-order support for each edge in G (line 1-2). Then, it determines the higher-order truss number for each edge by iteratively removing the edges until G is empty (line 3-9). Specifically, it first assigns the value of the minimum higher-order support plus 2 among the edges in the remaining graph G to k (line 4). It means the remaining graph is at least a (k, τ)-truss. Therefore, the higher-order truss number for the edges e in the remaining graph with $\mathsf{sup}_\tau(e, G) \leq k - 2$ is k (line 6). After the higher-order truss number of e is obtained, Algorithm 1 removes e from G (line 7). Due to the removal of e, the τ-hop common neighbors of edges incident to vertices in $\{u\} \cup \{v\} \cup \Delta_\tau(e, G')$ could be changed in G. Consequently, Algorithm 1 recomputes the higher-order support for these edges with $\mathsf{sup}_\tau(e', G) > k - 2$ (line 8-9). Algorithm 1 continues until G is empty (line 3).

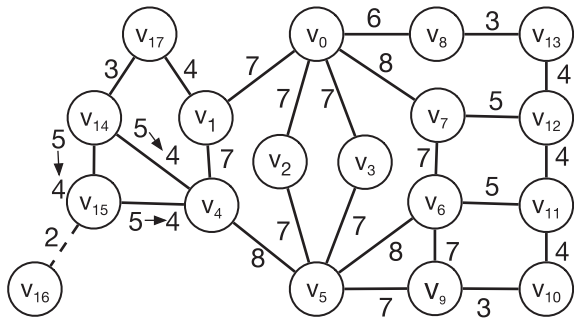

Fig. 3. The procedure of HOTDecom.

Algorithm 1. HOTDecom(G, τ)

1: **for each** $e \in E(G)$ **do**
2: compute $\mathsf{sup}_\tau(e, G)$;
3: **while** $G \neq \emptyset$ **do**
4: $k \leftarrow \mathsf{min}_{e \in E(G)} \mathsf{sup}_\tau(e, G) + 2$;
5: **while** $\exists e = (u, v) \in E(G)$ with $\mathsf{sup}_\tau(e, G) \leq k - 2$ **do**
6: $\phi_\tau(e, G) \leftarrow k$;
7: $G' \leftarrow G$; $G \leftarrow G \setminus e$;
8: **for each** $e' = (u', v') \in E(G)$ with $u', v' \in \{u\} \cup \{v\} \cup \Delta_\tau(e, G')$ with $\mathsf{sup}_\tau(e', G) > k - 2$ **do**
9: compute $\mathsf{sup}_\tau(e', G)$;

Example 4.1. Reconsider the graph G shown in Fig. 2. Fig. 3shows the procedure of HOTDecom to conduct the decomposition. It first computes $\mathsf{sup}_2(e, G)$ for each edge, which is shown near each edge. Since the minimum value of $\mathsf{sup}_2(e, G)$ among all the edges in G is 2, then k is assigned as 4 and $\phi_2((v_{15}, v_{16}), G)$ is 4. After that, (v_{15}, v_{16}) is removed and HOTDecom needs to update the higher-order support of (v_{14}, v_{15}). $\mathsf{sup}_2((v_{14}, v_{15}), G)$ decreases from 5 to 4, following (v_4, v_{14}) and (v_4, v_{15}). HOTDecom continues the above procedure until all the edges are removed. When it finishes, the higher-order truss number for each edge is obtained.

Theorem 4.1. *Given a graph G and an integer τ, Algorithm 1 computes the higher-order truss number for each edge correctly.*

Proof. To show the correctness of Algorithm 1, we only need to prove that the value assigned to $\phi_\tau(e, G)$ in line 6 of Algorithm 1 is the correct higher-order truss number for every edge $e \in E(G)$.

Based on the procedure of Algorithm 1, for the first edge e_1 of G processed in line 6, the value of k is the correct higher-order truss number of e_1. This is because when e_1 is processed, $\mathsf{sup}_\tau(e, G)$ is correctly computed for every edge of G in line 1-2 and the value of k is the minimum

反證法證明

value among all the computed $\text{sup}_\tau(e, G)$ plus 2. Based on Definition 2.3, the graph G itself is a (k, τ)-truss. Therefore, the higher-order truss number of e_1 is correctly computed.

Next, we show that the higher-order truss number is also correctly computed for the following processed edges $e_2, \ldots, e_m$. We prove it by contradiction. Without loss of generality, let e_i be the first edge assigned with wrong higher-order truss number, and the higher-order truss number assigned to e_i by Algorithm 1 is k_1 while the correct higher-order truss number of e_i is k_2. We first consider the case that $k_1 < k_2$. As e_i is the first edge assigned with wrong higher-order truss number and the correct higher-order truss number of e_i is $k_2 > k_1$, then, all the edges whose higher-order truss number is not less than k_2 have not been processed when processing e_i. According to Definition 2.3, these edges together with e_i consist of a (k_2, τ)-truss and $\text{sup}_\tau(e_i, G)$ is at least not less than $k_2 - 2$. Meanwhile Algorithm 1 assigns k_1 to $\phi_\tau(e_i, G)$ in line 6, it means $\text{sup}_\tau(e_i, G)$ is less than $k_1 - 2$ in line 5 before e_i is processed in line 6. It leads to the contradiction against the assumption that $k_1 < k_2$. Therefore, $k_1 < k_2$ is impossible. Similarly, we can prove that $k_1 > k_2$ is also impossible. Therefore, the higher-order truss number is also correctly computed for $e_2, \ldots, e_m$. Combining these two cases together, the theorem holds.□

Theorem 4.2. *Given a graph G and an integer τ, the time complexity of Algorithm 1 is $O(m \cdot m_\tau \cdot (m_\tau + n_\tau))$, where m_τ and n_τ are the maximum numbers of edges and vertices within the τ-hop neighborhood in the graph, respectively.*

Proof. In Algorithm 1, we first compute $\text{sup}_\tau(e, G)$ for each edge in G in line 1-2. To compute $\text{sup}_\tau(e, G)$ for an edge $e = (u, v)$, we first retrieve $N_\tau(u, G)$ and $N_\tau(v, G)$ of u and v by breadth-first search, which costs $O(m_\tau + n_\tau)$ time. Then, the computation of $\Delta_\tau(e, G)$ costs $O(n_\tau)$ time. Therefore, the computation of line 1-2 can be finished in $O(m \cdot (m_\tau + n_\tau))$. For line 3-9, when an edge e is removed in line 7, $O(m_\tau)$ edges in its τ-hop neighborhood need to update their higher-order edge support and each update consumes $O(m_\tau + n_\tau)$ time. Therefore, the time for updating higher-order edge support in line 8-9 can be bounded by $O(m_\tau \cdot (m_\tau + n_\tau))$. In Algorithm 1, each edge is removed once in line 7 and line 4-5 can be finished in const time by using a bin array. As a result, the time complexity of line 3-9 can be bounded by $O(m \cdot m_\tau \cdot (m_\tau + n_\tau))$. Therefore, the overall time complexity of Algorithm 1 is $O(m \cdot m_\tau \cdot (m_\tau + n_\tau))$. □

5 An Improved Decomposition Algorithm

In this section, we aim to improve the performance of the bottom-up decomposition paradigm. According to Theorem 4.2, the most time-consuming part of Algorithm 1 is the higher-order support update in line 8-9. Compared with the time complexity of line 1-2, an additional term m_τ is introduced in that of line 3-9. When an edge e is removed from G in line 7, HOTDecom updates $\text{sup}_\tau(e', G)$ for all the edge e' incident to vertices in $\{u\} \cup \{v\} \cup \Delta_\tau(e, G')$ in line 8-9 immediately. The immediate update strategy adopted by HOTDecom leads to the term m_τ in Theorem 4.2, which makes it prohibitively costly.

To address the performance issue in HOTDecom, we explore three optimization strategies, namely delayed update strategy, early pruning strategy, and unchanged support detection strategy to avoid unnecessary higher-order support update. In this section, we first show these three proposed strategies. Then, we present our improved algorithm for the higher-order truss decomposition.

5.1 A Delayed Update Strategy

Since the immediate update strategy leads to the term m_τ in the time complexity of HOTDecom, our first idea is to explore the opportunities to delay the higher-order support update until necessary. To achieve this goal, we first define:

Definition 5.1. (Truss Number Bounded Edge Set) *Given a graph G, an integer τ, and a condition $f(\phi)$, the truss number bounded edge set, denoted by $\Phi_{\tau, f(\phi)}(G)$, is the set of edges whose higher-order truss number $\phi_\tau(e, G)$ satisfies $f(\phi)$.*

Revisiting Algorithm 1, the key point to guarantee the correctness of the HOTDecom is that the value of $\text{sup}_\tau(e, G)$ for an edge $e \in \Phi_{\tau, \phi=k}(G)$ must be not greater than $k - 2$ when handling a specific k in line 5-6. Following this, HOTDecom updates $\text{sup}_\tau(e', G)$ for the edges incident to vertices in $\{u\} \cup \{v\} \cup \Delta_\tau(e, G')$ to keep the invariant. On the other hand, according to Definition 2.3, when handling a specific k, for an edge $e'' \in \Phi_{\tau, \phi>k}(G)$, it is not necessary to update the value of $\text{sup}_\tau(e'', G)$ immediately based on the following two reasons: (1) for the edge $e \in \Phi_{\tau, \phi=k}(G)$, the value of $\text{sup}_\tau(e'', G)$ does not affect the correctness of computing $\phi_\tau(e, G)$ in Algorithm 1 as $\text{sup}_\tau(e'', G) \geq \phi_\tau(e'', G) - 2 > k - 2$. (2) For the edge e'' itself, we can delay the computation of $\text{sup}_\tau(e'', G)$ until handling k whose value is identical to $\phi_\tau(e'', G)$ as $\phi_\tau(e'', G)$ is determined only by $\Phi_{\tau, \phi \geq k}$.

Following this idea, assume that we have already known the higher-order truss number of edges in prior, then, when an edge e is removed in line 7 of Algorithm 1, instead of updating $\text{sup}_\tau(e', G)$ for the edge e' incident to vertices in $\{u\} \cup \{v\} \cup \Delta_\tau(e, G')$, we only need to update $\text{sup}_\tau(e', G)$ for edges with $\phi_\tau(e') = k$. With this *delayed update strategy*, we can significantly reduce the number of edges updating their values of $\text{sup}_\tau(e', G)$ in line 9, which improves the performance of HOTDecom consequently. *Lower bound of $\phi_\tau(e, G)$.*However, it is intractable to obtain the higher-order truss number directly before the decomposition as it is our goal. Despite the intractability, for an edge e', if we can obtain a lower bound $\underline{\phi}_\tau(e', G)$ of its higher-order truss number rather than the exact value, when handling a specific k, we only update $\text{sup}_\tau(e', G)$ for edges with $\underline{\phi}_\tau(e', G) \leq k$, the above inference still establishes. Therefore, the remaining problem is how to obtain a tight lower bound for an edge efficiently. To achieve this goal, we have the following lemma:

subgraph有k truss那整張圖至少大於等k

Lemma 5.1. *Given a graph G and an integer τ, let G' be a subgraph of G, for any edge $e \in E(G')$, $\phi_\tau(e, G) \geq \phi_\tau(e, G')$.*

Proof. This lemma can be proved directly based on Definition 2.3. □

According to Lemma 5.1, for an edge in a graph G, the higher-order truss number of the edge in any subgraph of G is a lower-bound of the higher-order truss number of the edge in G. Moreover, we have the following lemma:

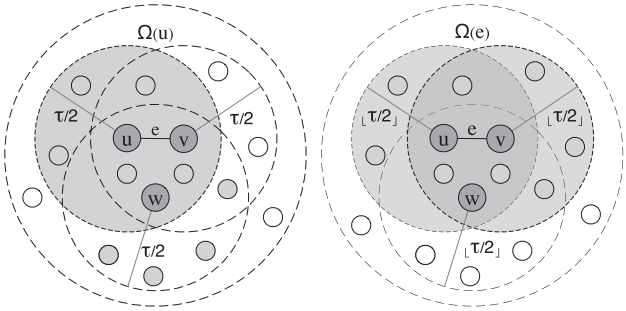

Fig. 4. $\underline{\phi}_\tau(e,G)$ for an edge $e=(u,v)$.

Lemma 5.2. *Given a graph G, for a subgraph G' in G, if the diameter $\omega(G') \le \tau$, then G' is a $(|V(G')|,\tau)$-truss.*

Proof. Since the diameter $\omega(G') \le \tau$, each vertex in G' is τ-hop reachable with each other. It means that for an edge $e=(u,v)$ in G', the τ-hop common neighbors of e in G' are all other vertices in G' except u and v. Therefore, we can drive that $\mathsf{sup}_\tau(e,G') = |V(G')|-2$ for all the edges in G'. According to Definition 2.3, G' is a $(|V(G')|,\tau)$-truss. The lemma holds. □

According to Lemma 5.1 and Lemma 5.2, for a graph G and an integer τ, if we have a subgraph G' of G such that $\omega(G') \le \tau$, then, for any edge $e \in E(G')$, we have $\phi_\tau(e,G) \ge |V(G')|$. Based on this, we define:

Definition 5.2. (Vertex Centric τ-Diameter Subgraph) *Given a graph G and an integer τ, for a vertex $v \in V(G)$, the vertex centric τ-diameter subgraph of v, denoted by $\Omega(v)$, is the subgraph of G induced by the set of vertices in $\{v\} \cup \{w | w \rightarrow_{\lfloor \frac{\tau}{2} \rfloor} v\}$.*

Lemma 5.3. *Given a vertex v in a graph G and an integer τ, for an edge $e=(u,v) \in E(G)$, $\phi_\tau(e,G) \ge |V(\Omega(v))|$.*

Proof. We consider two cases: (1) τ is even. Based on Definition 5.2, $\omega(\Omega(v)) \le \tau$. According to Lemma 5.2, $\Omega(v)$ is a $(|V(\Omega(v)|,\tau)$-truss. Since $\Omega(v)$ is a subgraph of G, according to Lemma 5.1, $\phi_\tau(e,G) \ge |V(\Omega(v))|$. (2) τ is odd. Based on Definition 5.2, $\omega(\Omega(v)) \le \tau-1$. According to Lemma 5.2, $\Omega(v)$ is a $(|V(\Omega(v)|,\tau-1)$-truss. As $\Omega(v)$ is a subgraph of G, according to Lemma 5.1, $\phi_{\tau-1}(e,G) \ge |V(\Omega(v))|$. Following Definition 2.3, $\phi_\tau(e,G) \ge \phi_{\tau-1}(e,G)$. Thus, $\phi_\tau(e,G) \ge |V(\Omega(v))|$. Combining the above two cases, the lemma holds. □

Following Lemma 5.3, for an edge $e=(u,v) \in E(G)$, we can derive that $\phi_\tau(e,G) \ge |V(\Omega(u))|$ as well. Moreover, we can also derive that:

Lemma 5.4. *Given a graph G and an integer τ, for an edge $e=(u,v) \in E(G)$, $\phi_\tau(e,G) \ge \mathsf{max}\{|V(\Omega(w))|\}$, where $w \in \Delta_{\lfloor \frac{\tau}{2} \rfloor}(e,G)$.*

Proof. This lemma can be proved similarly as Lemma 5.3. □

According to Lemma 5.3 and Lemma 5.4, for an edge $e=(u,v)$, a lower bound of e is $\mathsf{max}\{|V(\Omega(u))|, |V(\Omega(v))|, |V(\Omega(w))|\}$, where $w \in \Delta_{\lfloor \frac{\tau}{2} \rfloor}(e,G)$. However, in this case, if τ is odd, the value of $\underline{\phi}_\tau(e,G)$ is identical to that of $\underline{\phi}_{\tau-1}(e,G)$. To address this problem, we define:

Definition 5.3. (Edge Centric τ-Diameter Subgraph) *Given a graph G and an odd integer τ, for an edge $e=(u,v) \in E(G)$, the edge centric τ-diameter subgraph of e, denoted by $\Omega(e)$, is the subgraph of G induced by the set of vertices in $\{u\} \cup \{v\} \cup \{w | w \rightarrow_{\lfloor \frac{\tau}{2} \rfloor} u \vee w \rightarrow_{\lfloor \frac{\tau}{2} \rfloor} v\}$.*

Lemma 5.5. *Given an edge e in a graph G and an odd integer τ, $\phi_\tau(e,G) \ge |V(\Omega(e))|$.*

Proof. The lemma can be prove similarly as Lemma 5.3. □

Therefore, our lower bound is defined as follows:

Definition 5.4. (Lower Bound $\underline{\phi}_\tau(e,G)$) *Given a graph G and an integer τ, for an edge $e=(u,v) \in E(G)$, $\underline{\phi}_\tau(e,G) =$*

$$\begin{cases} \mathsf{max}\{|V(\Omega(u))|, |V(\Omega(v))|, |V(\Omega(w))|\}, & \tau \text{ is even} \\ \mathsf{max}\{|V(\Omega(e))|, |V(\Omega(w))|\}, & \tau \text{ is odd;} \end{cases}$$

where $w \in \Delta_{\lfloor \frac{\tau}{2} \rfloor}(e,G)$.

Example 5.1. Fig. 4 illustrates the idea of $\underline{\phi}_\tau(e,G)$ for an edge $e=(u,v)$. In Fig. 4a, the shadowed dashed circle represents the vertex centric τ-diameter subgraph $\Omega(u)$. In Fig. 4b, the shadowed part represents the edge centric τ-diameter subgraph $\Omega(e)$. When τ is even, we choose $\mathsf{max}\{|V(\Omega(u))|, |V(\Omega(v))|, |V(\Omega(w))|\}$ as $\underline{\phi}_\tau(e,G)$. When τ is odd, we choose $\mathsf{max}\{|V(\Omega(e))|, |V(\Omega(w))|\}$ as $\underline{\phi}_\tau(e,G)$. For a concrete example, consider the graph G shown in Fig. 2, take the edge (v_0,v_1) as an example, $\tau=2$, since τ is even, we just need to consider $\Omega(v_0)=\{v_0,v_1,v_2,v_3,v_7,v_8\}$, $\Omega(v_1)=\{v_0,v_1,v_4,v_{17}\}$. Therefore, $\underline{\phi}_2((v_0,v_1),G) = \mathsf{max}\{|\Omega(v_0)|, |\Omega(v_1)|\} = 6$.

5.2 An Early Pruning Strategy

With the lower bound, we can delay the computation of higher-order support when necessary, which consequently reduces the number of higher-order support update in line 8-9 of Algorithm 1. On the other hand, when handling a specific k during decomposition, if we have some light-weight methods that can determine the higher-order truss number of an edge e cannot be larger than k, then we can directly obtain $\phi_\tau(e,G)=k$, which means we can correctly obtain the higher-order truss number of e without the computation of its higher-order support. As a result, we can still reduce the number of higher-order support update and improve the performance of HOTDecom. Following this idea, we have:

Lemma 5.6. *Given a graph G, two integers k and τ, for an edge $e=(u,v) \in V(G)$, if $d_\tau(u,G) \le k-1 \vee d_\tau(v,G) \le k-1$, $\phi_\tau(e,G) \le k$.*

Proof. According to Property 1, if $d_\tau(u,G) \le k-1$ (resp. $d_\tau(v,G) \le k-1$), u (resp. v) is not contained in a $(k+1,\tau)$-truss. Thus, e is not contained in a $(k+1,\tau)$-truss. Based on Definition 2.4, $\phi_\tau(e,G) \le k$. □

With Lemma 5.6, when handling a specific k, we can prune an edge by computing the τ-hop degree of its two incident vertices. Moreover, if we have computed $d_\tau(v,G)$ for a vertex v and the value of $d_\tau(v,G)$ is not greater than $k-1$, then all the edges incident to v can be pruned. In other words, by computing the τ-hop degree of a single vertex, we can reduce the computation of higher-order support for multiple edges, which achieves the goal of early pruning.

5.3 An Unchanged Support Detection Strategy

In addition to the above discussed optimization strategies, here, we aim to explore the edges whose higher-order support unchanged after the removal of an edge to avoid the update. When an edge $e = (u, v)$ is removed , we have:

Lemma 5.7. *Given an edge $e = (u, v)$ in a graph G, let G' be the graph after the removal of e, for an edge $e' = (u', v')$ in G' where $u', v' \in \Delta_\tau(e, G)$, if the distance between u (resp. v) to u' (resp. v') in G and G' keep the same, then, $\mathsf{sup}_\tau(e', G) = \mathsf{sup}_\tau(e', G')$.*

Proof. According to Definition 2.2, we can prove the lemma if we can prove that $N_\tau(u', G) = N_\tau(u', G')$ and $N_\tau(v', G) = N_\tau(v', G')$. Therefore, we first prove $N_\tau(u', G) = N_\tau(u', G')$. It can be proved by contradiction. Assume that $N_\tau(u', G)$ and $N_\tau(u', G')$ are different due to the removal of e'. Since the removal of an edge only leads to the reduction of τ-hop reachable vertices from a specific vertex, we can derive that $N_\tau(u', G') \subseteq N_\tau(u', G)$. Without lose of generality, let w be a vertex in $N_\tau(u', G) \setminus N_\tau(u', G')$. Then, we can derive $u' \rightarrow_\tau w$ in G but $u' \nrightarrow_\tau w$ in G' according to Definition 2.2. Therefore, the shortest path from u' to w must pass through (u, v) in G. Let the shortest path from u' to w be $p = (u', \ldots, u, v, \ldots, w)$. Since the shortest path from u' to w passes through (u, v), we can derive that the shortest path from u' to v is the sub-path $p' = (u', \ldots, u, v)$ of p and $l(p') < l(p'')$, where p'' is other arbitrary path from u' to v not passing through (u, v) in G. Therefore, after the removal of e, the distance between u' and v must be larger than $l(p')$ in G', which contradicts with the fact that distance between u' and v in G and G' are the same. Therefore, $N_\tau(u', G) = N_\tau(u', G')$. Similarly, we can prove $N_\tau(v', G) = N_\tau(v', G')$. Therefore, $\mathsf{sup}_\tau(e', G) = \mathsf{sup}_\tau(e', G')$. □

According to Lemma 5.7, for an edge (u, v) in G, we can maintain the distance between u (resp. v) and the vertices in $\Delta_\tau(e, G)$ before and after the removal of (u, v). For those vertices $V' \subseteq \Delta_\tau(e, G)$ such that the distance between them and u (resp. v) keep the same, we can guarantee that the higher-order support of the edge e' connecting any two vertices in V' is unchanged after the removal of (u, v). Compared with updating the higher-order support for these edges e' directly, the above distance can be obtained by just two BFS traversals starting from u and v, respectively, which means we can further reduce the number of edges that need to update their higher-order support with little cost.

5.4 The Improved Algorithm

With the delayed update strategy, early pruning strategy and unchanged support detection strategy, we are ready to present our improved algorithm HOTDecom$^+$, which is shown in Algorithm 2.

Algorithm. Algorithm 2 shares a similar framework as Algorithm 1. It first computes the lower bound $\underline{\phi}_\tau(e, G)$ for each edge (line 1-2). Then, it decomposes the graph in the increasing order of k. It first initializes k as the minimum value of $\underline{\phi}_\tau(e, G)$ of all edges in G. For a specific k, Algorithm 2 first computes $\mathsf{sup}_\tau(e, G)$ for each edge with $\underline{\phi}_\tau(e, G) = k$ (line 5-6). Then, for the edge with $\mathsf{sup}_\tau(e, G) \leq k - 2$, it assigns the higher-order truss number to e and removes e from G similarly as Algorithm 1 (line 7-8). After removing e, Algorithm 2 updates the higher-order support for the edges $e' = (u', v')$ incident to vertices in $\{u\} \cup \{v\} \cup \Delta_\tau(e, G')$. It first checks whether $\underline{\phi}_\tau(e', G') > k$. If it is true, Algorithm 2 delays the computation of $\mathsf{sup}_\tau(e', G)$ (line 10-11). Otherwise, Algorithm 2 checks whether vertex u' or v' can be early pruned by pruneVertex (line 12-13). If these two vertices can not be pruned, it further checks whether the distance between u (resp. v) and u' (resp. v') is changed (line 15-16). If the distance is changed and $\mathsf{sup}_\tau(e', G') > k - 2$, Algorithm 2 recomputes $\mathsf{sup}_\tau(e', G)$ (line 18). Algorithm 2 continues the above procedure until G is empty (line 4).

Algorithm 2. HOTDecom$^+(G, \tau)$

```
1: for each e ∈ E(G) do
2:   compute φ_τ(e, G);
3: k ← min_{e∈E(G)}(φ_τ(e, G));
4: while G ≠ ∅ do
5:   for each e ∈ E(G) with φ_τ(e, G) = k do
6:     compute sup_τ(e, G);
7:   while ∃e = (u, v) ∈ E(G) with sup_τ(e, G) ≤ k − 2 do
8:     φ_τ(e, G) ← k; G' ← G; G ← G \ e;
9:     for each e' = (u', v') ∈ E(G) with u', v' ∈ {u} ∪ {v} ∪
       Δ_τ(e, G') do
10:      if φ_τ(e', G') > k then
11:        continue;              // delayed update
12:      if pruneVertex(G, u', k) or pruneVertex(G, v', k) then
13:        continue;              // early pruning
14:      compute dis_G(u, w), dis_G'(u, w), dis_G(v, w), dis_G'(v, w)
         with w ∈ Δ_τ(e, G');
15:      if dis_G'(u, u') = dis_G(u, u') ∧ dis_G'(u, v') =
         dis_G(u, v') ∧ dis_G'(v, u')=dis_G(v, u') ∧ dis_G'(v, v')=
         dis_G(v, v') then
16:        continue;              // unchanged support detection
17:      if sup_τ(e', G') > k − 2 then
18:        compute sup_τ(e', G);
19: k ← k + 1;
```

pruneVertex prunes the vertices based on Lemma 5.6. It uses a set $\mathcal{S}$ to record the vertices that can be pruned. For a vertex v, it first computes $d_\tau(v, G)$ (line 1). If $d_\tau(v, G)$ is not greater than $k - 1$, v is put in $\mathcal{S}$ (line 2-3). Then, it iteratively gets vertex u from $\mathcal{S}$ until $\mathcal{S}$ is empty. For u, since u can be pruned from G, which means the higher-order truss number for edges e incident to u is k, it assigns k to $\phi_\tau(e, G)$ and removes the vertex from G (line 6-8). Due to the removal of u, the τ-hop degree for the vertices in $N_\tau(u, G')$ may be changed. Therefore, it further checks the τ-hop degree of these vertices. If these vertices can be pruned, then they are put in $\mathcal{S}$ and removed from $\mathcal{N}$ (line 8-10). For the edges whose two incident vertices are in the $\mathcal{N}$, it recomputes their higher-order edge support if $\underline{\phi}_\tau(e'', G') \leq k$ (line 11-12). If the procedure prunes any vertices, it returns true (line 13). Otherwise, it returns false (line 15).

Example 5.2. Reconsider the graph G shown in Fig. 2. Fig. 5 shows the procedure of HOTDecom$^+$. It first computes lower bound $\underline{\phi}_2(e, G)$ for each edge. $\underline{\phi}_2(e, G)$ is shown near each edge in Fig. 2. After that, it starts the decomposition with $k = 3$. Since $\underline{\phi}_2((v_8, v_{13}), G) = 3$, it computes $\mathsf{sup}_2((v_8, v_{13}), G) = 3 > k - 2$. Then, k becomes 4 and it computes $\mathsf{sup}_2(e, G)$ for edges with $\underline{\phi}_2(e, G) = 4$. It gets $\mathsf{sup}_2((v_{15}, v_{16}), G) = 2$, $\mathsf{sup}_2((v_{14}, v_{15}), G) = 5$, …. Then, it removes (v_{15}, v_{16}). Due to the removal of (v_{15}, v_{16}) ,

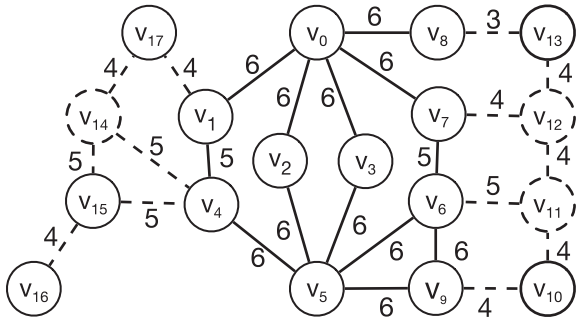

Fig. 5. The procedure of HOTDecom$^+$.

$\text{sup}_2((v_{14}, v_{15}), G)$ is updated from 5 to 4, since $\underline{\phi}_2((v_4, v_{14}), G) = 5 > 4$ and $\underline{\phi}_2((v_4, v_{15}), G) = 5 > 4$, it does not update their higher-order support. Then, k becomes 5, since $\text{sup}_2((v_{14}, v_{17}), G) = 3 \leq 5 - 2$, (v_{14}, v_{17}) is removed. Here, although $v_1, v_4 \in \Delta_2((v_{14}, v_{17}), G)$, it does not update the higher-order support of (v_1, v_4), (v_1, v_{17}) and (v_4, v_{17}) since the distance between v_1 (resp. v_4) and v_{14} (resp. v_{17}) is unchanged after the removal of (v_{14}, v_{17}). Meanwhile, due to the removal of (v_{14}, v_{17}), $d_2(v_{14}, G)$ is updated from 5 to 4, hence v_{14} and its incident edges are removed directly. (v_4, v_{15}) is removed following v_{14}. Here, although $v_4, v_5 \in \Delta_2((v_4, v_{14}) \cap \Delta_2((v_4, v_{15}))$, it does not update the support of (v_4, v_5) as $\underline{\phi}_2((v_4, v_5), G) > 5$. Similarly, edges and vertices are removed until G is empty. As shown in this example, many higher-order support updates are avoided compared with HOTDecom shown in Example 4.1.

Algorithm 3. pruneVertex (G, v, k)

```
1: S ← ∅; compute d_τ(v, G);
2: if d_τ(v, G) ≤ k − 1 then
3:   S.put(v);
4:   while S ≠ ∅ do
5:     u ← S.get();
6:     for each e = (u, w) ∈ E(G) with w ∈ N_1(u, G) do
7:       φ_τ(e, G) ← k;
8:     G' ← G; G ← G \ u; N ← N_τ(u, G');
9:     for each w ∈ N do
10:      if d_τ(w, G) ≤ k − 1 then S.put(w); N ← N \ w;
11:    for each e'' = (x, y) ∈ E(G) with x, y ∈ N do
12:      if φ_τ(e'', G') ≤ k then compute sup_τ(e'', G);
13:  return true;
14: else
15:  return false;
```

Theorem 5.1. *Given a graph G and an integer τ, Algorithm 2 computes the higher-order truss number for each edge correctly.*

Proof. Following Theorem 4.1, we only need to show that the reduced higher-order updates caused by line 11-16 of Algorithm 2 do not affect the correctness of the algorithm. This can be guaranteed by Lemma 5.3, Lemma 5.5, Lemma 5.6, and Lemma 5.7. □

Theorem 5.2. *The time complexity of Algorithm 2 is $O(m \cdot m_\tau \cdot (m_\tau + n_\tau))$.*

Although HOTDecom$^+$ shares the same worst case time complexity as HOTDecom, lots of higher-order edge support updates can be reduced in practice as verified in our experiments, which makes HOTDecom$^+$ significantly outperforms HOTDecom.

6 Top-r Higher-Order Trusses Computation

In the above section, we investigate the higher-order truss decomposition problem that the (k, τ)-trusses with all possible k values are computed. However, in some applications, users are only interested in the (k, τ)-truss with a large k value since the (k, τ)-truss with large k value is generally more cohesive and represents the core part of the graph [37], [38]. Certainly, we can use HOTDecom$^+$ directly to address this problem. Nevertheless, as this approach adopts the bottom-up decomposition paradigm, the (k, τ)-trusses with small k values have to be computed, which leads to lots of unnecessary computation. To address this problem, in this section, we propose a new approach tailored for the top r (k, τ)-trusses computation problem. Formally, given a graph G and two integers r and τ, the top r (k, τ)-trusses computation problem returns the (k, τ)-trusses with k values in the range of $(k_{\max} - r, k_{\max}]$, where $k_{\max}$ is the maximum value of k such that there is a non-empty (k, τ)-truss regarding the corresponding k value in the graph.

An upper-bound integrated approach. To conduct the top r (k, τ)-trusses computation, suppose that we have known the $\phi_\tau(e, G)$ for each edge e in G in prior, we can take the edges $\Phi_{\tau, \phi > k_{\max} - r}$ as the input of HOTDecom$^+$ and conduct the decomposition starting from $k_{\max} - r + 1$ to compute the result. In this way, the unnecessary computation involved in HOTDecom$^+$ can be totally avoided. However, this approach has to know $\phi_\tau(e, G)$ for each edge e in G in prior, which is intractable. On the other hand, if we can obtain an upper bound $\overline{\phi}_\tau(e, G)$ for each edge e in G, we can take the edges with $\overline{\phi}_\tau(e, G) > k_{\max} - r$ as the input of HOTDecom$^+$ and obtain the correct answer for the similar reason as HOTDecom$^+$. Following this idea, we propose an upper bound integrated approach to realize the top r (k, τ)-trusses computation. We first present the upper bound of $\phi_\tau(e, G)$.

Upper bound of $\phi_\tau(e, G)$. For an edge e in a given graph G, a direct upper bound of $\phi_\tau(e, G)$ is $\text{sup}_\tau(e, G)$. However, this bound is too loose. To obtain a tight upper bound, we define:

Definition 6.1. (Support-Vertex Bounded Subgraph) *Given a graph G and an integer τ, for an edge $e = (u, v) \in E(G)$, let G' be the subgraph induced by $\{u\} \cup \{v\} \cup \{\Delta_\tau(e, G)\}$, the support-vertex bounded subgraph of e, denoted by $\Lambda(e)$, is a connected subgraph G'' of G' such that (1) $e \in E(G'')$ (2) for each edge $e \in E(G'')$, $\text{sup}_\tau(e, G) \geq |V(G'')| - 2$ (3) $|V(G'')|$ is maximal.*

Lemma 6.1. *Given a graph G and an integer τ, for an edge $e = (u, v)$, $\phi_\tau(e, G) \leq |V(\Lambda(e))|$.*

Proof. We can prove it by contraction. Assume that there exists an edge $e' = (u', v')$ such that $\phi_\tau(e', G) > |V(\Lambda(e'))|$. Based on Definition 2.4, let $\mathbb{G}$ be the subgraph of G'' induced by $\{u'\} \cup \{v'\} \cup \{\Delta_\tau(e', G)\}$, where G'' is the $(\phi_\tau(e', G), \tau)$-truss in G containing e'. Based on Definition 2.3, or each edge $e'' \in E(\mathbb{G})$, $\text{sup}_\tau(e'', G'') \geq \phi_\tau(e'', G) - 2$. Moreover, $\text{sup}_\tau(e'', G) \geq \text{sup}_\tau(e'', G'')$. Therefore,

$\text{sup}_\tau(e'', G) \geq \phi_\tau(e'', G) - \ 2 \geq \phi_\tau(e', G) - 2 > |V(\Lambda(e'))| - 2$. Meanwhile, $|V(\mathbb{G})| \geq \phi_\tau(e', G) > |V(\Lambda(e'))|$. It means $\mathbb{G}$ satisfies condition (1) and (2) of Definition 6.1 but the number of vertices is bigger than $|V(\Lambda(e'))|$, which contracts with condition (3) of Definition 6.1. Thus, the lemma holds. □

Definition 6.2. (Upper Bound $\overline{\phi}_\tau(e, G)$) *Given a graph G and an integer τ, for an edge $e \in E(G)$, $\overline{\phi}_\tau(e, G) = |V(\Lambda(e))|$.*

Algorithm. With the upper bound, our top r (k, τ)-trusses computation algorithm HOTTopR is shown in Algorithm 4. It first computes $\underline{\phi}_\tau(e, G)$ and $\overline{\phi}_\tau(e, G)$ for each edge in G (line 1-2). Then, it retrieves the maximum value of $\overline{\phi}_\tau(e, G)$ among all the edges (line 3). After that, the graph $\mathcal{G}$ consisting the edges with $\overline{\phi}_\tau(e, G) > k_{\text{max}} - r$ is constructed (line 12-13). Then, Algorithm 4 computes the top r (k, τ)-trusses by utilizing Algorithm 2. Since Algorithm 4 uses the maximum value of $\overline{\phi}_\tau(e, G)$ among all the edges as k_{max} in line 3, it is possible that there exists no such (k_{max}, τ)-truss in G. In this case, Algorithm 4 further explores the possible k_{max} of G. It considers two subcases: (1) there exist some edges whose higher-order truss number have been obtained. In this case, it can be easily derived that k_{max} is the maximum value of $\phi_\tau(e, G)$ among these edges (line 6-7). (2) there exists no edge whose higher-order truss number is in the range of $(k_{\text{max}} - r, k_{\text{max}}]$. In this case, Algorithm 4 progressively decreases the value of k_{max} by r until k_{max} is obtained (line 9). After that, it continuously search search the result with new value of k_{max} and k. Note that for the edges $\phi_\tau(e, G)$ have been obtained, it does not need to recompute $\text{sup}_\tau(e, \mathcal{G})$ and just sets $\text{sup}_\tau(e, \mathcal{G})$ as $\phi_\tau(e, G) - 2$, which can reduce the unnecessary computation (line 15-16). Similarly, the value of $\text{sup}_\tau(e, \mathcal{G})$ for these edges is not necessary to be updated in line 21-22. The algorithm terminates when top r (k, τ)-trusses are found.

Procedure computeUB is used to compute $\overline{\phi}_\tau(e, G)$ for an edge e. It adopts a binary search strategy to find $\Lambda(e)$ and uses l and r to indicates the current search range. In each iteration, it starts two τ-hop BFS traversal from u and v respectively by visiting the edges with $\text{sup}_\tau(e, G) \geq \text{mid} - 2$ to construct G'' (line 28). If there exists G'' such that $|V(G'')| \geq \text{mid}$, $\overline{\phi}$ records current possible upper bound of $\phi_\tau(e, G)$ (line 32). The search continues until $l > r$ and returns $\overline{\phi}$.

Theorem 6.1. *Given a graph G, an integer τ and an integer r, Algorithm 4 computes the top r (k, τ)-trusses in G correctly.*

Proof: This theorem can be proved similarly as Theorem 5.1. □

Theorem 6.2. *The time complexity of Algorithm 4 is $O(r' \cdot m' \cdot m_\tau \cdot (m_\tau + n_\tau) + m \cdot \log n_\tau \cdot (m_\tau + n_\tau))$, where r' is the number of iterations in line 4, m' is the maximum number of edges of $\mathcal{G}$.*

Proof: This theorem can be proved similarly as Theorem 4.2. □

Although the time complexity of Algorithm 4 is not reduced compared with Algorithm 2 theoretically, Algorithm 4 is efficient in terms of the top r (k, τ)-trusses computation in practice. This is because the top r (k, τ)-trusses are generally much smaller than the input graph and the proposed upper bound shown in Definition 6.2 is effective.

Algorithm 4. HOTTopR(G, τ, r)

1: **for each** $e \in E(G)$ **do**
2: compute $\underline{\phi}_\tau(e, G)$; $\overline{\phi}_\tau(e, G) \leftarrow$ computeUB(e, G);
3: $k_{\text{max}} \leftarrow \text{max}_{e \in E(G)}\{\overline{\phi}_\tau(e, G)\}$; $\mathcal{G} \leftarrow \emptyset$; First $\leftarrow$ *true*;
4: **while** $\Phi_{\tau, \phi = k_{\text{max}}} = \emptyset$ **do**
5: **if** !First **then**
6: **if** $\exists e \in E(\mathcal{G})$ with $\phi_\tau(e, G)$ obtained **then**
7: $k_{\text{max}} \leftarrow \text{max}_{e \in E(\mathcal{G})}\{\phi_\tau(e, G)\}$;
8: **else**
9: $k_{\text{max}} \leftarrow k_{\text{max}} - r$;
10: $k \leftarrow k_{\text{max}} - r + 1$;
11: First $\leftarrow$ *false*;
12: **for each** $e \in E(G)$ with $\overline{\phi}_\tau(e, G) > k_{\text{max}} - r$ **do**
13: $\mathcal{G} \leftarrow \mathcal{G} \cup e$;
14: **for each** $e \in E(\mathcal{G})$ **do**
15: **if** $\phi_\tau(e, G)$ has been obtained **then**
16: $\text{sup}_\tau(e, \mathcal{G}) \leftarrow \phi_\tau(e, G) - 2$;
17: **else**
18: compute $\text{sup}_\tau(e, \mathcal{G})$;
19: **while** $k \leq k_{\text{max}}$ **do**
20: line 7-10 of Algorithm 2 replacing G with $\mathcal{G}$;
21: **if** $\phi_\tau(e', G)$ has been obtained **then**
22: **continue**;
23: line 11-19 of Algorithm 2 replacing G with $\mathcal{G}$;
24: **Procedure** computeUB$(e = (u, v), G)$
25: $l \leftarrow 2$; $r \leftarrow \text{sup}_\tau(e, G) + 2$;
26: **while** $l \leq r$ **do**
27: mid $\leftarrow (l + r)/2$;
28: construct G'' by τ-hop BFS traversal starting from u and v through edges with $\text{sup}_\tau(e, G) \geq \text{mid} - 2$;
29: **if** $|V(G'')| <$ mid **then**
30: $r \leftarrow \text{mid} - 1$;
31: **else**
32: $\overline{\phi} \leftarrow \text{mid}$; $l \leftarrow \text{mid} + 1$;
33: **return** $\overline{\phi}$;

7 Performance Studies

In this section, we present our experimental results. All the experiments are conduct on a machine with 4 Intel Xeon 3.0GHz CPUs and 64GB RAM running Linux.

Datasets. We evaluate our algorithms on 12 real world datasets. PT is downloaded from KONECT[1], DB is downloaded from Network Repository[2] and the remaining datasets are downloaded from SNAP[3]. Table 1 shows the details of the datasets, where d_{max} is the maximum degree of vertices in graphs.

Algorithms. We compare the following algorithms:

1) HOTD: HOTDecom algorithm.
2) HOTD+D: HOTDecom algorithm with the delayed update strategy.
3) HOTD+DU: HOTD+D algorithm with the unchanged support detection strategy.
4) HOTD$^+$: HOTDecom$^+$ algorithm.
5) TOP-5: HOTTopR algorithm with $r = 5$.

All algorithms are implemented in C++, using g++ compiler with -O3. Let τ=[2-4], since under large τ ($\tau > 4$), it

1. http://konect.cc/
2. http://networkrepository.com/
3. http://snap.stanford.edu/data/index.html

TABLE 1
Statistic of the Real Datasets

Datasets	Type	$\|V\|$	$\|E\|$	d_{max}
PT	Biography	1,870	2,203	56
CG	Collaboration	5,242	14,496	81
EM	E-Mail	1,005	25,571	345
CH	Collaboration	9,877	25,998	65
FB	Social	4,039	88,234	1,045
CA	Product	334,863	925,872	549
CD	Collaboration	317,080	1,049,866	343
AM	Purchasing	262,111	1,234,877	420
WN	Web	325,729	1,497,134	10,721
WG	Web	875,713	5,105,039	6,332
DB	Collaboration	4,000,150	8,649,005	954
CP	Citation	3,774,768	16,518,948	793

will lead to the loss of cohesiveness of (k, τ)-truss. Thereby, users can choose τ from $[2-4]$ according to their requirement for cohesiveness of result. All reported results were averaged over 5 repeated runs. If an algorithm cannot finish in 12 hours, we denote the processing time as INF.

Exp-1: Efficiency of our proposed algorithms. In this experiment, we compare the running time of five algorithms on all datasets when τ = 2,3,4. The results are shown in Fig. 6.

As τ increases, the running time of the algorithms increases as well. Regarding the algorithms, HOTD is slowest among the five algorithms. HOTD+D is much faster than HOTD benefited from the delayed update strategy. Moreover, with utilizing the unchanged support detection strategy, HOTD+DU is further faster than HOTD+D. Afterwards, HOTD$^+$ is more efficient than HOTD+DU. This is because HOTD$^+$ adopts all three optimization strategies to reduce the number of edges that need to update their higher-order support. Compared with HOTD, HOTD$^+$ achieves up to 4 orders of magnitude speedup. Besides, compared with HOTD$^+$, HOTTopR is suitable to compute the top r results as it avoids lots of unnecessary computation related to the non-top-r results in HOTD$^+$.

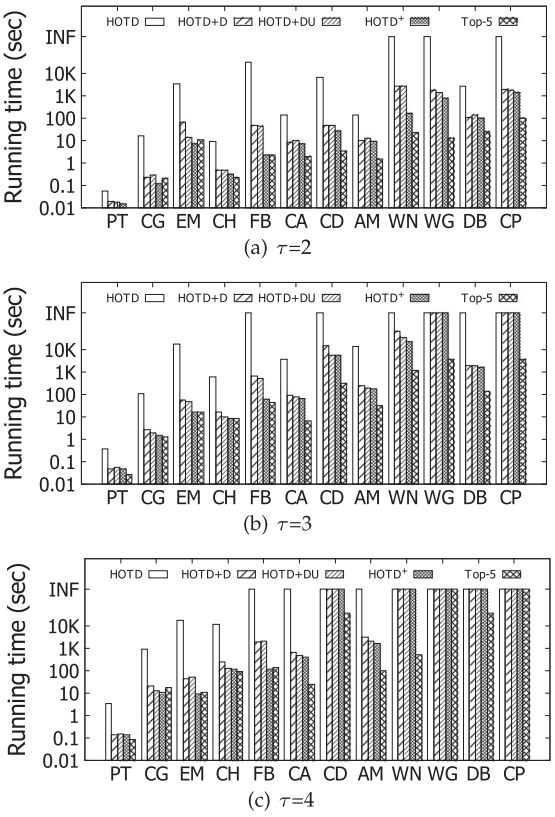

Fig. 6. The running time of algorithms.

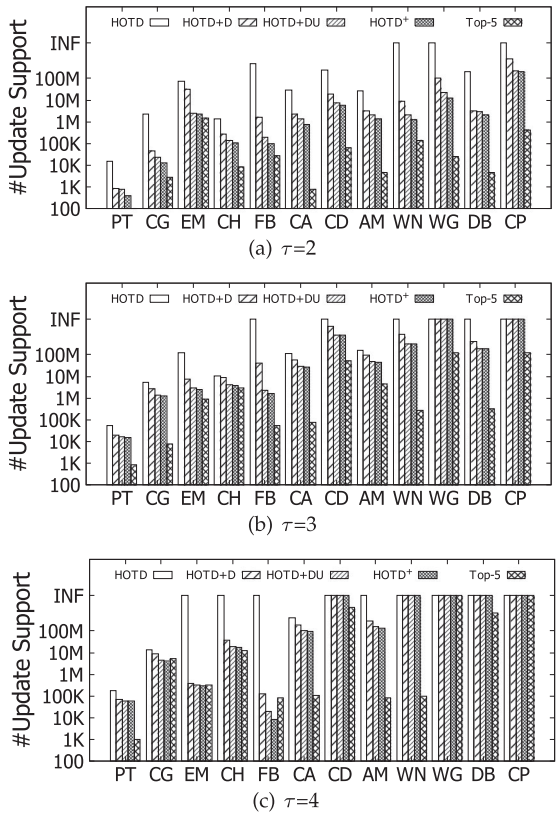

Fig. 7. The number of the support update.

Exp-2: Number of higher-order support update. In this experiment, we report the number of higher-order support update of all algorithms during the decomposition when $\tau = 2, 3, 4$. The results are shown in Fig. 7.

As shown in Fig. 7, on most datasets, the number of higher-order support updates of HOTD increase when τ increases from 2 to 4. This is because as the value of τ increases, the scope of edges that need update enlarges as well. However, the number of higher-order support update is significantly reduced by the optimization strategies. The results also explain the reasons causing different running times of the algorithms shown in Fig. 6.

Exp-3: Tightness of $\underline{\phi}_\tau(e, G)$. In this experiment, we evaluate the tightness of the lower bound $\underline{\phi}_\tau(e, G)$ in HOTD$^+$. We use the widely adopted approximation error (AE) to measure the tightness [39]. The approximation error for $\underline{\phi}_\tau(e, G)$ is defined as follows:

$$AE(\underline{\phi}_\tau(e, G)) = \frac{\sum_{i=1}^{m} \frac{|\phi_\tau(e_i, G) - \underline{\phi}_\tau(e_i, G)|}{\phi(e_i, G)}}{m}; \quad (1)$$

As shown in Table 2, for $\underline{\phi}_\tau(e, G)$, when $\tau = 2$, on most datasets, the approximation error is even less than 0.1, like PT, CG, FB, ... When $\tau = 3$, although its performance is not as good as that of $\tau = 2$, the approximation error is still small on most datasets. This is because the 3-hop neighborhood information is much more complex than the 2-hop neighborhood

TABLE 2
Tightness of $\underline{\phi}_\tau(e, G)$

Dataset	τ=2	τ=3	τ=4	Dataset	τ=2	τ=3	τ=4
PT	0.02	0.16	0.20	CD	0.06	0.60	-
CG	0.03	0.36	0.31	AM	0.11	0.39	0.36
EM	0.34	0.49	0.03	WN	0.03	0.09	-
CH	0.07	0.54	0.47	WG	0.06	-	-
FB	0.002	0.06	0.10	DB	0.01	0.15	-
CA	0.07	0.27	0.28	CP	0.15	-	-

information. When $\tau = 4$, it shows a trend similar to that of $\tau = 3$. Therefore, $\underline{\phi}_\tau(e, G)$ is effective regarding $\phi_\tau(e, G)$.

Exp-4: Scalability. Finally, we evaluate the scalability of the proposed algorithms on two kinds of synthetic datasets (random graph and power-law graph). For the random graph, we generates 8 graphs varying the number of vertices from 2^{17} to 2^{24} using the method in [40], [41]. For the power-law graph, we generates 8 graphs varying the number of vertices from $3M$ to $10M$ using the method in [42] (average degree with 10, other parameters with default setting). Fig. 8 and Fig. 9 show the running time of algorithms on random synthetic graphs and power-law synthetic graphs, respectively.

Fig. 8 and Fig. 9 show that as the graph size increases, the running time of all algorithms increases as well. This is because as the size of graphs increases, more edges have to be processed during the decomposition. Besides, for $\tau = 2, 3, 4$, benefited from the proposed optimization strategies, HOTD$^+$ is the most efficient one among the four algorithms and shows good scalability compared with other three algorithms.

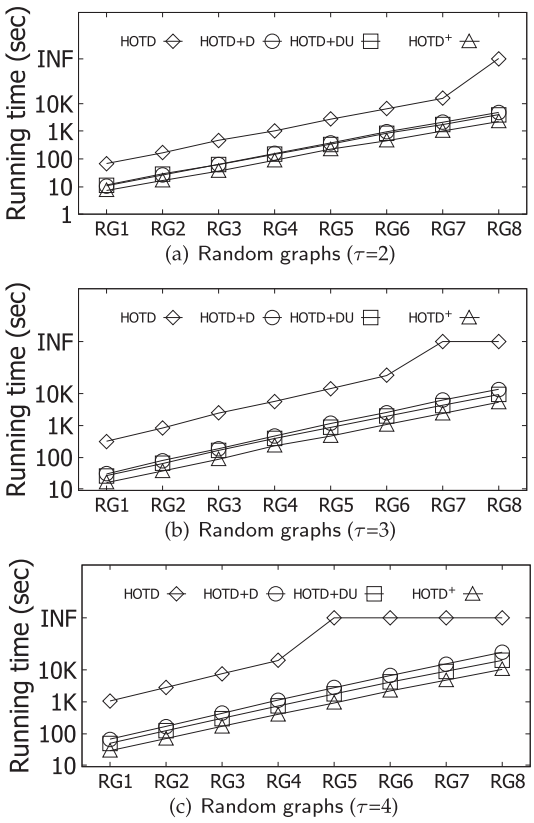

Fig. 8. Scalability on synthetic random graphs.

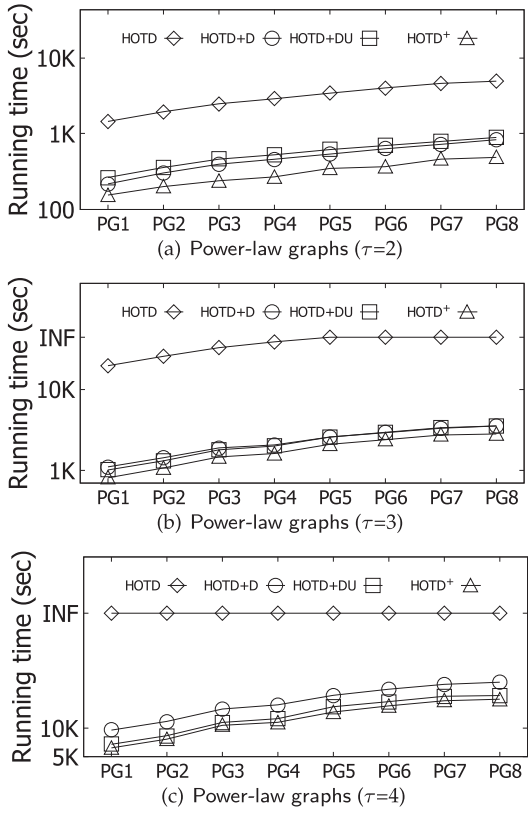

Fig. 9. Scalability on synthetic power-law graphs.

Exp-5: Case study. In this experiment, we compare (k, τ)-truss with (k, h)-core which is the existing most similar cohesive subgraph model with our model. Fig. 10 shows the differences between them on two real-world graphs Highschool and Polotical Books from KONECT. Highschool is a network contains friendships between boys in a small high-school. Polotical Books is a network of books, edges between books represent frequent co-purchasing of books by the same buyers. As shown at Fig. 10a, for (k, h)-core, most vertices are covered by (20,2)-core, it is hard to further distinguish more cohesive structure from it. For our model, (20,2)-core is further decomposed into more fine-grained hierarchy structure, like (16,2)-truss, (18,2)-truss and (20,2)-truss. Obviously, (20,2)-truss is the most cohesive subgraph through graph visualization. Fig. 10b shows the similar phenomenon on Polotical Books dataset. It's because a $(k + 1,\tau)$-truss is a (k, h)-core but not vice versa, $\tau = h$. Therefore, our model can further search "core" of a (k, h)-core benefited from the more rigorous requirement for cohesiveness. In a result, our (k, τ)-truss has higher ability to reveal fine-grained structure information.

8 Related Work

Truss decomposition. In the literature, plenty of research efforts have been devoted to the cohesive subgraph models for graph structure analysis[9], [10], [11], [13], [15], [17], [27], [43], [44]. Among these cohesive subgraph models, the k-truss model has received considerable attention. k-truss model is first introduced in [15]. In [15], an in-memory algorithm to detect the k-truss for a given k with time complexity $O(\sum_{v \in V(G)} d_1^2(v, G))$ is proposed. [32] studies the problem of

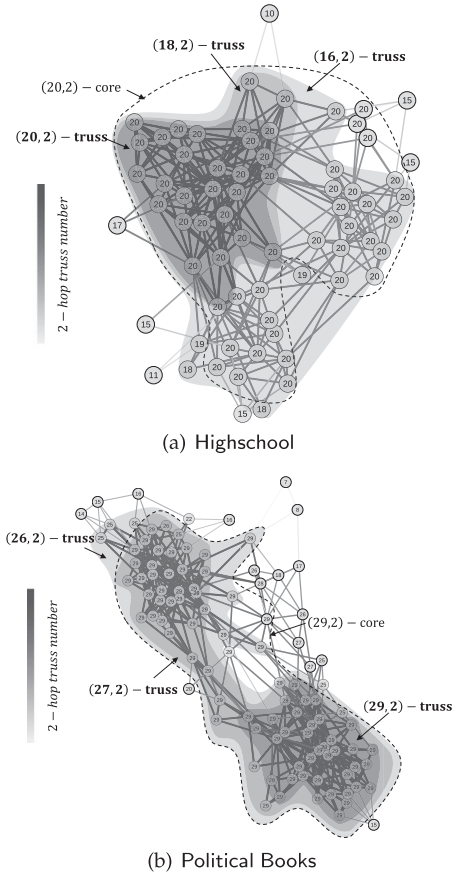

Fig. 10. Compare (k, τ)-truss with (k, h)-core, hop=2.

detecting the k-truss for a given k in distribute environments. By using the triangle counting techniques proposed in [45], [19] proposes an truss decomposition algorithm with time complexity $O(m^{\frac{3}{2}})$. [19] also investigates the external-memory algorithms to conduct the truss decomposition. [46] proposes an efficient algorithm to maintain the truss decomposition results in evolving graphs. [47] investigates the truss decomposition problem in probabilistic graphs. Compared with our (k, τ)-truss model, all these models only consider the direct neighbors of an edge and the higher-order neighborhood information is missed.

Distance-generalized cohesive subgraph models. Clique [9], k-core [13], [27] and k-truss [15] are three most fundamental cohesive subgraph models. For clique model, [48], [49] propose two distance-generalized clique models, named h-club and h-clique. An h-club is a maximal subgraph C, s.t., the length of the shortest path between any two vertices of C is not greater than h. The only difference between h-club and h-clique is that the shcortest path is within an h-club while it is not strictly necessary for h-clique. For k-core model, [30] proposes (k, h)-core model which requires each vertex in it has more than k neighbors within h-hop neighborhood. [27] proposes the (k, h)-core decomposition algorithm with better performance compared with [30]. However, as shown at the experiments part (Exp-5), our model can search "core" of (k, h)-core with higher ability to reveal fine-grained structure information. Besides, there is no distance-generalized model based on k-truss before our model.

9 Conclusion

As a representative cohesive subgraph model, k-truss model has received considerable attention. However, the traditional k-truss model ignores the higher-order neighborhood information of an edge, which limits its ability to reveal fine-grained structure information of the graph. Motivated by this, in this paper, we propose the (k, τ)-truss model and study the higher-order truss decomposition problem. We first propose a bottom-up decomposition paradigm for this problem. Based on the paradigm, we further explore three optimization strategies, namely delayed update strategy, early pruning strategy, and unchanged support detection strategy, to improve the decomposition performance. Moreover, we also devise an efficient algorithm to compute the top r (k, τ)-trusses, which is useful in practical applications. Our experimental results on real datasets and synthetic datasets show the efficiency, effectiveness and scalability of the proposed algorithms.

References

[1] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing," *Proc. Very Large Data Bases Endow.*, vol. 11, no. 4, pp. 420–431, 2017.
[2] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Effective and efficient dynamic graph coloring," *Proc. Very Large Data Bases Endow.*, vol. 11, no. 3, pp. 338–351, 2017.
[3] D. Ouyang, L. Yuan, L. Qin, L. Chang, Y. Zhang, and X. Lin, "Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees," *Proc. Very Large Data Bases Endow.*, vol. 13, no. 5, pp. 602–615, 2020.
[4] L. Chang and L. Qin, *Cohesive Subgraph Computation Over Large Sparse Graphs Algorithms, Data Structures, and Programming Techniques*. Berlin, Germany: Springer, 2018.
[5] R. D. Luce and A. D. Perry, "A method of matrix analysis of group structure," *Psychometrika*, vol. 14, no. 2, pp. 95–116, 1949.
[6] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang, "Index-based densest clique percolation community search in networks," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 5, pp. 922–935, May 2018.
[7] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified top-k clique search," *Very Large Data Bases J.*, vol. 25, no. 2, pp. 171–196, 2016.
[8] Z. Chen, L. Yuan, X. Lin, L. Qin, and J. Yang, "Efficient maximal balanced clique enumeration in signed networks," in *Proc. Int. Conf. World Wide Web*, 2020, pp. 339–349.
[9] R. D. Luce, "Connectivity and generalized cliques in sociometric group structure," *Psychometrika*, vol. 15, no. 2, pp. 169–190, 1950.
[10] J. Abello, M. G. C. Resende, and S. Sudarsky, "Massive quasi-clique detection," in *Latin Proc. Amer. Symp. Theor. Inform.*, 2002, pp. 598–612.
[11] J. Pei, D. Jiang, and A. Zhang, "On mining cross-graph quasi-cliques," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2005, pp. 228–238.
[12] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation: An index-based approach," in *Proc. World Wide Web Conf.*, 2019, pp. 1130–1141.
[13] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269–287, 1983.
[14] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation in bipartite graphs," *Very Large Data Bases J.*, vol. 29, no. 5, pp. 1075–1099, 2020.
[15] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," *Technical Report, National Security Agency*, vol. 16, no. 3.1, 2008.
[16] X. Wu, L. Yuan, X. Lin, S. Yang, and W. Zhang, "Towards efficient k-tripeak decomposition on large graphs," in *Proc. DASFAA*, vol. 11446, 2019, pp. 604–621.
[17] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li, "Finding maximal k-edge-connected subgraphs from a large graph," in *Proc. 15th Int. Conf. Extending Database Technol.*, 2012, pp. 480–491.
[18] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 205–216.

[19] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proc. Very Large Data Bases Endow.*, vol. 5, no. 9, pp. 812–823, 2012.
[20] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1311–1322.
[21] E. Akbas and P. Zhao, "Truss-based community search: A truss-equivalence based indexing approach," *Proc. Very Large Data Bases Endow.*, vol. 10, no. 11, pp. 1298–1309, 2017.
[22] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Truss-based community search over large directed graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2183–2197.
[23] R. F. S. Andrade, J. G. V. Miranda, and T. P. Lob Ao , "Neighborhood properties of complex networks," *Phys. Rev. E*, vol. 73, Apr. 2006, Art. no. 046101.
[24] R. F. Andrade, J. G. Miranda, S. T. Pinho, and T. P. Lobao, "Characterization of complex networks by higher order neighborhood properties," *Eur. Phys. J. B*, vol. 61, no. 2, pp. 247–256, 2008.
[25] S. Abu-El-Haija *et al.*, "MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 21–29.
[26] S. Liu, L. Chen, H. Dong, Z. Wang, D. Wu, and Z. Huang, "Higher-order weighted graph convolutional networks," *CoRR*, vol. abs/1911.04129, 2019.
[27] F. Bonchi, A. Khan, and L. Severini, "Distance-generalized core decomposition," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 1006–1023.
[28] H. Xue, X. Sun, and W. Sun, "Multi-hop hierarchical graph neural networks," in *Proc. IEEE Int. Conf. Big Data Smart Comput.*, 2020, pp. 82–89.
[29] Z. Sun *et al.*, "Knowledge graph alignment network with gated multi-hop neighborhood aggregation," in *Proc. Conf. Assoc. Advancement Artif. Intell.*, 2020, pp. 222–229.
[30] V. Batagelj and M. Zaveršnik,"Fast algorithms for determining (generalized) core groups in social networks," *Adv. Data Anal. Classification*, vol. 5, pp. 129–145, 2011.
[31] C. Orsini, E. Gregori, L. Lenzini, and D. Krioukov, "Evolution of the internet -dense structure," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 1769–1780, Dec. 2014.
[32] Y. Shao, L. Chen, and B. Cui, "Efficient cohesive subgraphs detection in parallel," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 613–624.
[33] E. Mones, L. Vicsek, and T. Vicsek, "Hierarchy measure for complex networks," *PLoS One*, vol. 7, 2012, Art. no. 03.
[34] P. Colomer-de Simón, M. A. Serrano, M. G. Beiró, J. I. Alvarez-Hamelin , and M. Boguná, "Deciphering the global organization of clustering in real complex networks," *Sci. Rep.*, vol. 3, 2013, Art. no. 2517.
[35] P. Eades, S. Hong, A. Nguyen, and K. Klein, "Shape-based quality metrics for large graph visualization," *J. Graph Algorithms Appl.*, vol. 21, no. 1, pp. 29–53, 2017.
[36] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz— open source graph drawing tools," in *Graph Drawing*, P. Mutzel, M. Jünger, and S. Leipert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 483–484.
[37] V. E. Lee, N. Ruan, R. Jin, and C. C. Aggarwal, "A survey of algorithms for dense subgraph discovery," in *Managing and Mining Graph Data*. Berlin, Germany: Springer, 2010, pp. 303–336.
[38] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "I/O efficient ECC graph decomposition via graph reduction," *Proc. Very Large Data Bases Endow.*, vol. 9, no. 7, pp. 516–527, 2016.
[39] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 1996.
[40] M. Holtgrewe, P. Sanders, and C. Schulz, "Engineering a scalable high quality graph partitioner," in *Proc. 24th IEEE Int. Symp. Parallel Distrib. Process*, 2010, pp. 1–12.
[41] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2625–2638, Sep. 2017.
[42] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E*, vol. 78, no. 4, p. 046110, 2008.
[43] N. Wang, J. Zhang, K. Tan, and A. K. H. Tung, "On triangulation-based dense neighborhood graphs discovery," *Proc. Very Large Data Bases Endow.*, vol. 4, no. 2, pp. 58–68, 2010.
[44] D. Wen, L. Qin, Y. Zhang, L. Chang, and L. Chen, "Enumerating k-vertex connected components in large graphs," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 52–63.
[45] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theory Comput. Sci.*, vol. 407, no. 1–3, pp. 458–473, 2008.
[46] Y. Zhang and J. X. Yu, "Unboundedness and efficiency of truss maintenance in evolving graphs," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 1024–1041.
[47] X. Huang, W. Lu, and L. V. S. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 77–90.
[48] R. Luce, "Connectivity and generalized cliques in sociometric group structure," *Psychometrika*, vol. 15, pp. 169–190, 1950.
[49] E. Moradi and B. Balasundaram, "Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts," *Optim. Lett.*, vol. 12, no. 8, pp. 1947–1957, 2018.

Zi Chen received the PhD degree from the Software Engineering Institute, East China Normal University, Shanghai, China, in 2021. She is currently a postdoctoral with ECNU. Her research interest is big graph data query and analysis, including cohesive subgraph search on large-scale social networks and path planning on road networks.

Long Yuan received the BS and MS degrees both from Sichuan University, Chengdu, China, and the PhD degree from the University of New South Wales, Kensington, Australia. He is currently a professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, China. His research interests include graph data management and analysis. He has published papers in conferences and journals including VLDB, ICDE, WWW, *The VLDB Journal*, and *IEEE Transactions on Knowledge and Data Engineering*.

Li Han received the PhD degree from ENS Lyon, Lyon, France and ECNU, Shanghai, China, in 2020. She is currently an associate professor with the Software Engineering Institute at East China Normal University. She is mainly interested in resilience and scheduling problems for large-scale platforms. For more information, please visit https://faculty.ecnu.edu.cn/_s43/hl2/main.psp

Zhengping Qian received the PhD degree from the South China University of Technology, Guangzhou, China, in 2009. He is currently a senior staff engineer and group manager with Damo Academy, Alibaba group. He is doing research and is broadly interested in computation on large-scale graph.