Higher-Order Neighborhood Truss Decomposition

Zi Chen\textsuperscript{1} , Long Yuan\textsuperscript{2} , Li Han\textsuperscript{1} , Zhengping Qian\textsuperscript{3}

\textsuperscript{1}East China Normal University, \textsuperscript{2}Nanjing University of Science and Technology, \textsuperscript{3}Alibaba Group

zchen.hanli@sei.ecnu.edu.cn, longyuan@njust.edu.cn, zhengping.qzp@alibaba-inc.com

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 $(k, \tau)$-truss that considers the higher-order neighborhood $(\tau$ hop) information of an edge. Based on the $(k, \tau)$-truss model, we study the higher-order truss decomposition problem which computes the $(k, \tau)$-trusses for all possible $k$ values regarding a given $\tau$. 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 $(k, \tau)$-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.

1 Introduction

Graphs have been widely used to represent the relationships of entities in real-world applications [Sahu \textit{et al.}, 2017, Yuan \textit{et al.}, 2017, Ouyang \textit{et al.}, 2020]. With the proliferation of graph applications, plenty of research efforts have been devoted to cohesive subgraph models for graph structure analysis [Chang and Qin, 2018]. Typical cohesive subgraph models include clique [Luce and Perry, 1949, Yuan \textit{et al.}, 2018, Yuan \textit{et al.}, 2016a, Chen \textit{et al.}, 2020], $k$-clique [Luce, 1950a, quasi-clique [Abello \textit{et al.}, 2002, Pei \textit{et al.}, 2005], $k$-core [Liu \textit{et al.}, 2019a, Seidman, 1983, Liu \textit{et al.}, 2020a], $k$-truss [Cohen, 2008, Wu \textit{et al.}, 2019], and $k$-ECC [Zhou \textit{et al.}, 2012, Chang \textit{et al.}, 2013].

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 [Wang and Cheng, 2012, Huang \textit{et al.}, 2014, Akbas and Zhao, 2017, Liu \textit{et al.}, 2020a]. 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 $k-2$ [Cohen, 2008]. 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 Figure 1. Figure 1 shows part of the collaboration network in 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 Figure 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 [Andrade \textit{et al.}, 2006, Andrade \textit{et al.}, 2008], and remarkable results have been obtained in the network science due to the introduction of higher-order neighborhood [Abu-El-Haija \textit{et al.}, 2019, Liu \textit{et al.}, 2019b, Bonchi \textit{et al.}, 2019, Xue \textit{et al.}, 2020, Sun \textit{et al.}, 2020]. Motivated by this, we propose the $(k, \tau)$-truss model by incorpo-
rating the higher-order neighborhood into the \( k \)-truss model. Formally, given a graph \( G \) and an integer \( \tau \), for an edge \( e = (u, v) \), the higher-order support of \( e \) is the number of \( \tau \)-hop common neighbors of \( u \) and \( v \). \((k, \tau)\)-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 \((k, \tau)\)-truss model, we study the higher-order truss decomposition problem that computes the \((k, \tau)\)-truss for all possible \( k \) values in \( G \) regarding a given \( \tau \).

The benefits of the \((k, \tau)\)-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 Figure 1, Figure 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 \((k, h)\)-core that also considers higher-order neighborhood information is studied in [Batagelj and Zaversnik, 2011; Bonchi et al., 2019]. However, the definition of our model is more rigorous, which makes our model can search “core” of a \((k, h)\)-core (e.g. a \((k+1, \tau)\)-truss is a \((k, h)\)-core but not vice versa, \( \tau = h \)). 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 \((k, \tau)\)-truss model when \( \tau = 1 \), namely \((k, 1)\)-truss. These applications include community detection and search [Huang et al., 2014; Akbas and Zhao, 2017]. Moreover, as the higher-order truss decomposition can reveal more fine-grained structure of graphs compared with the traditional truss decomposition as shown in Figure 1, it can also be applied in the applications which focus on the hierarchical structure of graph, such like hierarchical structure analysis [Orsini et al., 2013] and graph visualization [Colomer-de Simón et al., 2013; Eades et al., 2017; Ellson et al., 2002]. Besides, due to the consideration of the higher-order neighborhood with parameter \( \tau \), 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 [Wang and Cheng, 2012]. 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 \( u \) and \( v \) and their \( \tau \)-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 undetermination 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 undetermination of the decreased higher-order support value, it seems insoluble 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 \((k, \tau)\)-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 \((k, \tau)\)-truss model. Motivated by the traditional \(k\)-truss model ignores the higher-order neighborhood information of an edge, we propose the \((k, \tau)\)-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 \((k, \tau)\)-truss model.

- Efficient algorithms for the higher-order truss decomposition problem. 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 \((k, \tau)\)-trusses with large \(k\) values, we study the top \(r\) \((k, \tau)\)-trusses computation problem which returns the \((k, \tau)\)-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 \((k, \tau)\)-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 \(r\) \((k, \tau)\)-trusses computation problem which returns the \((k, \tau)\)-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 \(p = (v_1, v_2, \cdots, v_j)\) where \((v_i, v_{i+1}) \in E\) for each \(1 \leq i < j\) 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 \(v\) the distance of \(u\) and \(v\), and denote it as \(d_G(u, v)\). 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 \(\tau\)-hop reachable from \(v\), denoted by \(v \rightarrow_{\tau} u\), if there is a path \(p\) between \(u\) and \(v\) with \(l(p) \leq \tau\). 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 \(\tau\)-hop neighbors of \(v\), denoted by \(N_\tau(v, G)\), is the set of vertices \(u\) such that \(u\) is \(\tau\)-hop reachable from \(v\). Similarly, the \(\tau\)-hop degree of \(v\), denoted by \(d_\tau(v, G)\), is the number of \(\tau\)-hop neighbors of \(v\), i.e., \(d_\tau(v, G) = |N_\tau(v, G)|\).

Definition 2.1: \((\tau\)-Hop Common Neighbor) Given a graph \(G\) and an integer \(\tau\), for an edge \(e = (u, v)\) in \(G\), \(w\) is a \(\tau\)-hop common neighbor of \(e\), i.e., \(\Delta_\tau(e, G) = N_\tau(u, G) \cap N_\tau(v, G)\).

Definition 2.2: \((\tau\)-Hop Edge Support) Given a graph \(G\) and an integer \(\tau\), for an edge \(e\), the \(\tau\)-hop support of \(e\), denoted by \(\sup_\tau(e, G)\), is the number of \(\tau\)-hop common neighbors of \(e\), i.e., \(\sup_\tau(e, G) = |\Delta_\tau(e, G)|\).

Definition 2.3: \((k, \tau)\)-Truss. Given a graph \(G\) and an integer \(\tau\), a \((k, \tau)\)-truss is a maximal subgraph \(G'\) of \(G\) such that \(\sup_\tau(e, G') \geq k - 2\) for all \(e \in E(G')\) and no more edges can be added into \(G'\).

Definition 2.4: \((\tau\)-Hop Truss Number) Given a graph \(G\) and an integer \(\tau\), for an edge \(e\), the \(\tau\)-hop truss number of \(e\), denoted by \(\phi_\tau(e, G)\), is the maximum value of \(k\) such that \(e\) is contained in the corresponding \((k, \tau)\)-truss.

Problem Statement. In the applications, the higher-order truss number of each edge is required. Given a graph \(G\) and an integer \(\tau\), in this paper, we study the higher-order truss decomposition problem which aims to compute the \((k, \tau)\)-trusses of \(G\) for all possible \(k\) values regarding \(\tau\). Straightforwardly, the \((k, \tau)\)-truss of \(G\) consists of the set of edges with higher-order truss number at least \(k\), i.e., \(\cup_{\phi_\tau(e, G) \geq k} E(G)\). 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 Figure \[2\] and 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, \(\Delta_2((v_0, v_1), G) = N_2(v_0, G) \cap N_2(v_1, G) = \{v_2, \ldots, v_7, v_8, v_{14}, v_{15}, v_{17}\}\), \(\sup_2((v_0, v_1), G) = 7\). \(\sup_2(e, G)\) for each edge \(e\) can be computed similarly and we can
find that \( \sup_\tau(e, G) \geq 2 \). 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 \( \sup_\tau(e, G') \geq 3 \) for each edge \( e \) in \( G' \) and no more edges can be added to \( G' \) to make it 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 Figure 2. The hierarchy structure of \( G \) is clearly illustrated by the value of \( \phi_2(e, G) \).

3 Theoretical Properties of \((k, \tau)\)-truss

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

**Property 1: (Minimum Degree)** Given a \((k, \tau)\)-truss \( G' \) of \( G \), for each vertex \( v \in V(G') \), \( d_r(v, G') \geq k - 1 \).

*Proof.* For a vertex \( v \), let \( e = (u, v) \) be an edge incident to \( v \) in \( G' \). Based on Definition 2.3, we have \( \sup_\tau(e, G') \geq k - 2 \). According to Definition 2.2, \( v \) has at least \( k - 2 \) \( \tau \)-hop neighbors in \( \Delta_\tau(e, G) \). Moreover, \( u \) is a 1-hop neighbor of \( v \). Thus, \( d_r(v, G') \geq k - 1 \). 

**Property 2: (Bounded Diameter)** Given a \((k, \tau)\)-truss \( G' \) of \( G \), for each connected component \( G'' \) of \( G' \), the diameter of \( G'' \) \( \omega(G'') \leq 2r |V(G'')| - 1 \).

*Proof.* Without loss of generality, let \( p = \langle v_1, v_2, \ldots, v_d \rangle \) 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 \( e_i = \langle v_i, v_{i+1} \rangle \) on \( p \), we use \( \Gamma_i \) to denote the set of vertices in \( \Delta_\tau(e_i, G'') \cap V(p) \), according to Definition 2.1, it is clear that \( |\Delta_\tau(e_i, G'') \cap V(p)| \leq 2r - 2 \). Since \( \Gamma_i = \Delta_\tau(e_i, G'') \cap V(p) \), \( |\Gamma_i| = |\Delta_\tau(e_i, G'')| - |\Delta_\tau(e, G'') \cap V(p)| \). According to Definition 2.3, \( |\Delta_\tau(e, G'') \cap V(p)| \leq 2r - 2 \). Together with \( |\Delta_\tau(e_i, G'') \cap V(p)| \leq 2r - 2 \), we have \( |\Gamma_i| \geq 2r - 2 \). Meanwhile, for a vertex \( w \in V(G'') \setminus V(p) \), we have \( \{\Gamma_i : w \in \Gamma_i, 1 \leq i \leq d - 1\} \leq 2r \). This can be proved by contradiction. Assume that \( \{\Gamma_i : w \in \Gamma_i, 1 \leq i \leq d - 1\} \geq 2r + 1 \), let \( E_{1r} \) be the set of edges on \( p \) such that \( w \) is a \( \tau \)-hop common neighbor. Assume that \( |E_{1r}| \geq 2r + 1 \). Let \( e_j = \langle v_j, v_{j+1} \rangle \) and \( e_k = \langle v_k, v_{k+1} \rangle \) be two edges in \( E_{1r} \) 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 \( 2r + 1 \) due to \( |E_{1r}| \geq 2r + 1 \). On the other hand, since \( w \) is a \( \tau \)-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 \( 2r \). It contradicts with the assumption that the shortest path from \( v_j \) to \( v_{k+1} \) on \( p \) with \( p' \) to obtain a shorter path. Therefore, \( \{\Gamma_i : w \in \Gamma_i, 1 \leq i \leq d - 1\} \leq 2r \). Based on this, we can derive that \( |\Gamma_i| \geq |\Gamma_i \cup \Gamma_{i+1} | \geq 2r - (d-1)(2r-2) \). Since \( V(G'') = \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_{d-1} \cup V(p) \), \( |V(G'')| = |\Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_{d-1} | + |V(p)| \), we can derive that \( |V(G'')| \geq 2r - (d-1)(2r-2) + d \). As \( \omega(G'') = d - 1, \omega(G'') \leq 2r |V(G'')| - 1 \), the property holds.

Algorithm 1 HOTDecom \((G, \tau)\)

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

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 \( \tau \), a \((k + 1, \tau)\)-truss is contained by a \((k, \tau)\)-truss.

*Proof.* According to Definition 2.3, for each edge \( e \) in a \((k + 1, \tau)\)-truss \( G' \), \( \sup_\tau(e, G') \geq k - 1 \geq k - 2 \). It is clear that \( G' \) is also a \((k, \tau)\)-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 \((k, \tau)\)-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 have been 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 \( \tau \), 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 \((k, \tau)\)-truss. Therefore, the higher-order truss number for the edges \( e \) in the remaining graph with \( \sup_\tau(e, G) \leq k - 2 \) 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 \( \tau \)-hop common neighbors of \( e \) 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 \( \sup_\tau(e', G) > k - 2 \) (line 8-9). Algorithm 1 continues until \( G \) is empty (line 3).

**Example 4.1:** Reconsider the graph \( G \) shown in Figure 2. Figure 3 shows the procedure of HOTDecom to conduct the decomposition. It first computes \( \sup_\tau(e, G) \) for each edge, which is shown near each edge. Since the minimum value of \( \sup_\tau(e, G) \) among all the edges in \( G \) is 2, then \( k \) is assigned as 4 and \( \phi_\tau((v_{15}, v_{16}), G) = 4 \). After that, \((v_{15}, v_{16}) \) is removed.
and HOTDecom needs to update the higher-order support of 
\((v_{14}, v_{132})\). \(\text{sup}_2((v_{14}, v_{132}), G)\) decreases from 5 to 4, following \((v_4, v_{14})\) and \((v_4, v_{132})\). 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 \(\tau\), Algorithm \[\text{HOTDecom}\] computes the higher-order truss number for each edge correctly.

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

Based on the procedure of Algorithm \[\text{HOTDecom}\] 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, \(\text{sup}_\tau(e, G)\) is correctly computed for every edge of \(G\) in line 1-2 and the value of \(k\) is the minimum value among all the computed \(\text{sup}_\tau(e, G)\) plus 2. Based on Definition 2.3, the graph \(G\) itself is a \((k, \tau)\)-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 \[\text{HOTDecom}\] is \(k_1\) while the correct higher-order truss number of \(e_i\) is \(k_2\). We first consider the case that \(k_1 \leq k_2\). 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 \((k_2, \tau)\)-truss and \(\text{sup}_\tau(e_i, G)\) is at least not less than \(k_2 - 2\). Meanwhile Algorithm \[\text{HOTDecom}\] 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 \(\tau\), the time complexity of Algorithm \[\text{HOTDecom}\] is \(O(m \cdot m_\tau \cdot (m_\tau + n_\tau))\), where \(m_\tau\) and \(n_\tau\) are the maximum numbers of edges and vertices within the \(\tau\)-hop neighborhood in the graph, respectively.

**Proof.** In Algorithm \[\text{HOTDecom}\] 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 \(O(m_\tau \cdot (m_\tau + n_\tau))\). For line 3-9, when an edge \(e\) is removed in line 7, \(O(m_\tau)\) edges in its \(\tau\)-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 \[\text{HOTDecom}\] each edge is removed once in line 7 and line 4-5 can be finished in constant 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 \[\text{HOTDecom}\] 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 \[\text{HOTDecom}\] is the higher-order support update in line 8-9. Compared with the time complexity of line 1-2, an additional term \(m_\tau\) 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 \(m_\tau\) 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 \(m_\tau\) 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 \(\tau\), 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 \[\text{HOTDecom}\] the key point to guarantee the correctness of the HOTDecom is that the value of \(\text{sup}_\tau(e, G)\) for an edge \(e \in \Phi_{\tau, f(\phi)}(G)\) 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 \(e'' \in \Phi_{\tau, f(\phi)}(G)\), it is not necessary to update...
the value of $\sup_{\tau}(e'', G)$ immediately based on the following two reasons: (1) for the edge $e \in \Phi_{\tau, \phi_{\geq k}}(G)$, the value of $\sup_{\tau}(e'', G)$ does not affect the correctness of computing $\phi_{\tau}(e, G)$ in Algorithm 1 as $\sup_{\tau}(e'', G) \geq \phi_{\tau}(e'', G) - 2 \geq k - 2$. (2) For the edge $e''$ itself, we can delay the computation of $\sup_{\tau}(e'', G)$ 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 $\sup_{\tau}(e', G)$ for the edge $e'$ incident to vertices in $\{u\} \cup \{v\} \cup \Delta_1(e, G')$, we only need to update $\sup_{\tau}(e', G)$ for edges with $\phi_{\tau}(e') = k$. With this delayed update strategy, we can significantly reduce the number of edges updating their values of $\sup_{\tau}(e', G)$ 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 $\phi_{\tau}(e', G)$ of its higher-order truss number rather than the exact value, when handling a specific $k$, we only need to update $\sup_{\tau}(e', G)$ for edges with $\phi_{\tau}(e', G) \leq k$, 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:

**Lemma 5.1**: Given a graph $G$ and an integer $\tau$, 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:

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

**Proof**. Since the diameter $\omega(G') \leq \tau$, each vertex in $G'$ is $\tau$-hop reachable with each other. It means that for an edge $e = (u, v)$ in $G'$, the $\tau$-hop common neighbors of $e$ in $G'$ are all other vertices in $G'$ except $u$ and $v$. Therefore, we can drive that $\sup_{\tau}(e, G') = |V(G')| - 2$ 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 $\tau$, if we have a subgraph $G'$ of $G$ such that $\omega(G') \leq \tau$, then, for any edge $e \in E(G')$, we have $\phi_{\tau}(e, G') \geq |V(G')|$. Based on this, we define:

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

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

**Proof**. We consider two cases: (1) $\tau$ is even. Based on Definition 5.2, $\omega(\Omega(v)) \leq \tau$. 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, $\phi_{\tau-1}(e, G) > |V(\Omega(v))|$. (2) $\tau$ is odd. Based on Definition 5.2, $\omega(\Omega(v)) \leq \tau - 1$. 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, $\phi_{\tau-1}(e, G) > |V(\Omega(v))|$. Following Definition 5.2, $\phi_{\tau-1}(e, G) > \phi_{\tau-1}(e, G)$. Thus, $\phi_{\tau}(e, G) \geq |V(\Omega(v))|$. 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 $\phi_{\tau}(e, G) \geq |V(\Omega(u))|$ as well. Moreover, we can also derive that:

**Lemma 5.4**: Given a graph $G$ and an integer $\tau$, for an edge $e = (u, v) \in E(G)$, $\phi_{\tau}(e, G) \geq \max\{|V(\Omega(w))|\}, w \in \Delta_{\Omega(v)}(e, G)$.

**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 $\max\{|V(\Omega(u))|, |V(\Omega(v))|\}$, where $w \in \Delta_{\Omega(v)}(e, G)$. However, in this case, if $\tau$ is odd, the value of $\phi_{\tau}(e, G)$ is identical to that of $\phi_{\tau-1}(e, G)$. To address this problem, we define:

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

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

**Proof**. The lemma can be proved similarly as Lemma 5.3. □

Therefore, our lower bound is defined as follows:

**Definition 5.4**: (Lower Bound $\phi_{\tau}(e, G)$) Given a graph $G$ and an integer $\tau$, for an edge $e = (u, v) \in E(G)$, $\phi_{\tau}(e, G) = \min\{|V(\Omega(u))|, |V(\Omega(v))|\}$. $\tau$ is even, $\phi_{\tau}(e, G) = \min\{|V(\Omega(u))|, |V(\Omega(v))|\}$. $\tau$ is odd, $\phi_{\tau}(e, G) = \min\{|V(\Omega(u))|, |V(\Omega(v))|\}$.

**Example 5.1**: Figure 4 illustrates the idea of $\phi_{\tau}(e, G)$ for an edge $e = (u, v)$. In Figure 4 (a), the shadowed dashed circle represents the vertex centric $\tau$-diameter subgraph $\Omega(u)$. In Figure 4 (b), the shadowed part represents the edge centric $\tau$-diameter subgraph $\Omega(e)$. When $\tau$ is even, we choose $\max\{|V(\Omega(u))|, |V(\Omega(v))|\}$ as $\phi_{\tau}(e, G)$. When $\tau$ is odd, we choose $\max\{|V(\Omega(u))|, |V(\Omega(v))|\}$ as $\phi_{\tau}(e, G)$. For a concrete example, consider the graph $G$ shown in Figure 2, take the edge $(v_0, v_1)$ as an example,
Given a graph \(e\) performance of HOTDecom

\[
\phi_e = \frac{\tau}{2}
\]

With the lower bound, we can delay the computation of an edge by computing the distance of \(\tau\) to \(e\). Definition 2.4, the \(\phi_e\) of \(e\) is unchanged after the removal of an edge to avoid the update. In addition to the above discussed optimization strategies, the computation of higher-order support for multiple edges, we can derive that 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 \(p' = (u', \ldots, u, v, \ldots, w)\) of \(p\) and \(l(p') < l(p)\), where \(p'\) is another 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 \(G\) are the same. Therefore, \(N_e(u', G) = N_e(u', G')\). Similarly, we can prove \(N_e(u', G) = N_e(v', G')\). Therefore, \(\tau\) is even.

\[
\phi_e = \frac{\tau}{2}
\]

Following this idea, we have:

Lemma 5.6: Given a graph \(G\), two integers \(k\) and \(\tau\), for an edge \(e = (u, v) \in V(G)\), if \(d_{\tau}(u, G) \leq k - 1\) and \(d_{\tau}(v, G) \leq k - 1\), \(\phi_e(u, G) \leq k\).

Proof. According to Property \(\tau\) if \(d_{\tau}(u, G) \leq k - 1\) (resp. \(d_{\tau}(v, G) \leq k - 1\)), \(u\) (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, \(\phi_e(u, G) \leq k\).

With Lemma 5.6, when handling a specific \(k\), we can prune an edge by computing the \(\tau\)-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 \(\tau\)-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.

5.4 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\(\star\), which is shown in Algorithm 2.

Algorithm. Algorithm 2 shares a similar framework as Algorithm 1. It first computes the lower bound \(\phi_e(u, G)\) 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 \(\phi_e(u, G)\) of all edges in \(G\). For a specific \(k\), Algorithm 2 first computes \(sup_{\tau}(e, G)\) for each edge with \(\phi_e(u, G) = k\) (line 5-6). Then, for the edge with \(sup_{\tau}(e, G) \leq k - 2\), 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 \(\phi_e(u', G') > k\). If it is true, Algorithm 2 delays the computation of \(sup_{\tau}(e', G)\) (line 10-11).
Algorithm 2 HOTDecom\(^+\)(\(G, \tau\))

1: for each \(e \in E(G)\) do
2: compute \(\phi_\tau(e, G)\);
3: \(k \leftarrow \min_{e \in E(G)} \phi_\tau(e, G)\);
4: while \(G \neq \emptyset\) do
5: for each \(e \in E(G)\) with \(\phi_\tau(e, G) = k\) do
6: compute \(\operatorname{sup}_\tau(e, G)\);
7: while \(\exists e = (u, v) \in E(G)\) with \(\operatorname{sup}_\tau(e, G) \leq k - 2\) do
8: \(\phi_\tau(e, G) \leftarrow k; G' \leftarrow G \setminus e;\)
9: for each \(e' = (u', v') \in E(G)\) with \(u', v' \in \{u\} \cup \{v\} \cup \Delta_\tau(e, G')\) do
10: if \(\phi_\tau(e', G') > k\) then
11: continue; // delayed update
12: if \(\text{pruneVertex}(G, u', k)\) or \(\text{pruneVertex}(G, v', k)\) then
13: continue; // early pruning
14: compute \(\operatorname{dis}_G(u, w), \operatorname{dis}_{G'}(u, w), \operatorname{dis}_G(v, w), \operatorname{dis}_{G'}(v, w)\) with \(w \in \Delta_\tau(e, G')\);
15: if \(\operatorname{dis}_G(u, u') = \operatorname{dis}_G(u, u') \land \operatorname{dis}_{G'}(u, v') = \operatorname{dis}_G(u, v') \land \operatorname{dis}_{G'}(v, v') = \operatorname{dis}_G(v, v')\) then
16: continue; // unchanged support detection
17: if \(\operatorname{sup}_\tau(e', G') > k - 2\) then
18: compute \(\operatorname{sup}_\tau(e', G)\);
19: \(k \leftarrow k + 1;\)

Otherwise, Algorithm 2 checks whether vertex \(u'\) or \(v'\) can be early pruned by \(\text{pruneVertex}\) (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 \(\operatorname{sup}_\tau(e', G') > k - 2\), Algorithm 2 recomputes \(\operatorname{sup}_\tau(e', G)\) (line 18). Algorithm 2 continues the above procedure until \(G\) is empty (line 4).

\(\text{pruneVertex}\) prunes the vertices based on Lemma 5.6. It uses a set \(S\) to record the vertices that can be pruned. For a vertex \(v\), it first computes \(d_\tau(v, G)\) (line 1). If \(d_\tau(v, G) < k - 1\) and \(v\) is put in \(S\) (line 2-3). Then, it iteratively gets vertex \(u\) from \(S\) until \(S\) 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 \(\tau\)-hop degree for the vertices in \(N_\tau(u, G')\) may be changed. Therefore, it further checks the \(\tau\)-hop degree of these vertices. If these vertices can be pruned, then they are put in \(S\) and removed from \(N'\) (line 8-10). For the edges whose two incident vertices are in \(N'\), it recomputes their higher-order edge support if \(\phi_\tau(e', G') \leq k\) (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 Figure 2. Figure 5 shows the procedure of HOTDecom\(^+\). It first computes lower bound \(\phi_\tau(e, G)\) for each edge. \(\phi_\tau(e, G)\) is shown near each edge in Figure 2. After that, it starts the decomposition with \(k = 3\). Since \(\phi_\tau((v_8, v_{13}), G) = 3\), it computes \(\operatorname{sup}_\tau((v_8, v_{13}), G) = 3 > k - 2\). Then, \(k\) becomes 4 and it computes \(d_\tau(e, G)\) for edges with \(\phi_\tau(e, G) = 4\). It gets \(\operatorname{sup}_\tau((v_{15}, v_{16}), G) = 2, \operatorname{sup}_\tau((v_{14}, v_{15}), G) = 5, \ldots\). Then, it removes \((v_{15}, v_{16})\). Due to the removal of \((v_{15}, v_{16})\), \(\operatorname{sup}_\tau((v_{14}, v_{15}), G)\) is updated from 5 to 4, since \(\phi_\tau((v_4, v_{14}), G) = 5 > 4\) and \(\phi_\tau((v_4, v_{15}), G) = 5 > 4\), it does not update their higher-order support. Then, \(k\) becomes 5, since \(\operatorname{sup}_\tau((v_{14}, v_{17}), G) = 5 \leq 5 - 2 = 3\). \((v_{14}, v_{17})\) is removed. Here, although \((v_1, v_4, v_5) \in \Delta_\tau((v_{14}, v_{17}), G)\), it does not update the higher-order support of \((v_1, v_4), (v_1, v_5)\) and \((v_4, v_5)\) 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 \((v_{14}, v_{17})\), \(d_\tau(v_{14}, G)\) 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 \((v_4, v_5) \in \Delta_\tau((v_{14}, v_{14}) \cap \Delta_\tau((v_4, v_{15}), G)\), it does not update the support of \((v_4, v_5)\) as \(\phi_\tau((v_4, v_5), G) > 5\). 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.

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

**Proof.** Following Theorem 4.1, we only need to show that
Theorem 5.2: The time complexity of Algorithm 2 is $O(m \cdot m^r \cdot (m^r + n^r))$.

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 $(k,\tau)$-trusses with all possible $k$ values are computed. However, in some applications, users are only interested in the $(k,\tau)$-trusses with large $k$ value since the $(k,\tau)$-truss with large $k$ value is generally more cohesive and represents the core part of the graph [Lee et al., 2010] [Yuan et al., 2016b]. Certainly, we can use HOTDecom$^+$ directly to address this problem. Nevertheless, as this approach adopts the bottom-up decomposition paradigm, the $(k,\tau)$-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 $r$ $(k,\tau)$-trusses computation problem. Formally, given a graph $G$ and two integers $r$ and $\tau$, the top $r$ $(k,\tau)$-trusses computation problem returns the $(k,\tau)$-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 $(k,\tau)$-truss regarding the corresponding $k$ value in the graph.

An upper-bound integrated approach. To conduct the top $r$ $(k,\tau)$-trusses computation, suppose that we have known the $\phi_r(e,G)$ for each edge $e$ in $G$ in prior, we can take the edges $\Phi_{\tau,\phi_{\max}-r}$ 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 $\phi_r(e,G)$ for each edge $e$ in $G$ in prior, which is intractable. On the other hand, if we can obtain an upper bound $\overline{\phi}_r(e,G)$ for each edge $e$ in $G$, we can take the edges with $\overline{\phi}_r(e,G) > k_{max} - r$ 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 $r$ $(k,\tau)$-trusses computation. We first present the upper bound of $\phi_r(e,G)$.

Upper bound of $\phi_r(e,G)$. For an edge $e$ in a given graph $G$, a direct upper bound of $\phi_r(e,G)$ is $sup_r(e,G)$. 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 $\tau$, 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 $e \in E(G'')$, $sup_r(e,G) \geq |V(G'')| - 2$ (3) $|V(G'')|$ is maximal. \[ \]

Lemma 6.1: Given a graph $G$ and an integer $\tau$, for an edge $e = (u,v)$, $\phi_r(e,G) \leq |V(\Lambda(e))|$. \[ \]

Proof. We can prove it by contradiction. Assume that there exists an edge $e' = (u',v')$ such that $\phi_r(e',G) > |V(\Lambda(e'))|$. Based on Definition 2.4, let $G$ be the subgraph of $G''$ induced by \{u'\} $\cup$ \{v'\} $\cup$ \{$\Delta_\tau(e',G)$\}, where $G''$ is the $(\phi_r(e',G),\tau)$-truss in $G$ containing $e'$. Based on Definition 2.3, for each edge $e'' \in E(G)$, $sup_r(e'',G'') \geq \phi_r(e'',G') - 2$. Moreover, $sup_r(e'',G') \geq sup_r(e'',G'')$. Therefore, $sup_r(e'',G') \geq \phi_r(e'',G') - 2 \geq \phi_r(e',G) - 2 > |V(\Lambda(e'))| - 2$. Meanwhile, $|V(G)| \geq \phi_r(e',G) > |V(\Lambda(e'))|$. It means $G$ 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 $\overline{\phi}_r(e,G)$) Given a graph $G$ and an integer $\tau$, for an edge $e \in E(G)$, $\overline{\phi}_r(e,G) = |V(\Lambda(e'))|$.

Algorithm. With the upper bound, our top $r$ $(k,\tau)$-trusses computation algorithm HOTTopR is shown in Algorithm 4.

It first computes $\phi_r(e,G)$ and $\overline{\phi}_r(e,G)$ for each edge in $G$ (line 1-2). Then, it retrieves the maximum value of $\overline{\phi}_r(e,G)$ among the all the edges (line 3). After that, the graph $G$ consisting the edges with $\overline{\phi}_r(e,G) > k_{max} - r$ is constructed (line 12-13). Then, Algorithm 4 computes the top $r$ $(k,\tau)$-trusses by utilizing Algorithm 2. Since Algorithm 4 uses the maximum value of $\overline{\phi}_r(e,G)$ among all the edges as $k_{max}$, in line 3, it is possible that there exists no such $(k_{max},\tau)$-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 $\phi_r(e,G)$ among these edges (line 6-7). (2) there exists no edge whose higher-order truss number is in the range of $(k_{max} - r, k_{max}]$. 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 the result with new value of $k_{max}$ and $k$. Note that for the edges $\phi_r(e,G)$ have been obtained, it does not need to recompute $sup_r(e,G)$ and just sets $sup_r(e,G)$ as $\phi_r(e,G) - 2$, which can reduce the unnecessary computation (line 15-16). Similarly, the value of $sup_r(e,G)$ for these edges is not necessary to be updated in line 21-22. The algorithm terminates when top $r$ $(k,\tau)$-trusses are found.

Procedure computeUB is used to compute $\overline{\phi}_r(e,G)$ 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 $\tau$-hop BFS traversal from $u$ and $v$ respectively by visiting the edges with $sup_r(e,G) \geq mid - 2$ to construct $G''$ (line 28). If there exists $G''$ such that $|V(G'')| \geq mid$, $\overline{\phi}$ records current possible upper bound of $\phi_r(e,G)$ (line 32). The search continues until $l > r$ and returns $\overline{\phi}$.

Theorem 6.1: Given a graph $G$, an integer $\tau$ and an integer
Algorithm 4 HOTTopR($G, \tau, r$)

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

$r$, Algorithm 4 computes the top $r$ $(k, \tau)$-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 $O(r' \cdot \log E \cdot \max\{\tau + n_\tau (m_\tau + n_\tau) + m \cdot \log n_\tau (m_\tau + n_\tau)\})$, where $r'$ is the number of iterations in line 4, $m' \cdot \tau$ is the maximum number of edges of $G$.

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 $r$ $(k, \tau)$-trusses computation in practice. This is because the top $r$ $(k, \tau)$-trusses are generally much smaller than the input graph and the proposed upper bound shown in Definition 6.2 is effective.

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.

Table 1: Statistic of the real datasets

| Datasets | Type     | $|V|$ | $|E|$ | $d_{\text{max}}$ |
|----------|----------|------|------|----------------|
| 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           |

Figure 6: The running time of algorithms

Datasets. We evaluate our algorithms on 12 real world datasets. PT is downloaded from KONECT, DB is downloaded from Network Repository and the remaining datasets are downloaded from SNAP. Table 1 shows the details of

http://konect.uni-koblenz.de/
http://networkrepository.com/
http://snap.stanford.edu/data/index.html
The number of the support update

the datasets, where \( d_{\text{max}} \) 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 \( \tau = [2-4] \), since under large \( \tau (\tau > 4) \), it will lead to the loss of cohesiveness of \((k,\tau)\)-truss. Thereby, users can choose \( \tau \) 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 \( \tau = 2, 3, 4 \). The results are shown in Figure 8.

As \( \tau \) 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+.

**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 Table 2.

As shown in Figure 7, on most datasets, the number of higher-order support update of HOTD increase when \( \tau \) increases from 2 to 4. This is because as the value of \( \tau \) 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 Figure 6.

### Table 2: Tightness of \( \phi_r(e, G) \)

| Dataset | \( \tau = 2 \) | \( \tau = 3 \) | \( \tau = 4 \) |
|---------|------------|------------|------------|
| PT      | 0.02       | 0.16       | 0.20       |
| CG      | 0.03       | 0.36       | 0.31       |
| EM      | 0.34       | 0.49       | 0.03       |
| CH      | 0.07       | 0.54       | 0.47       |
| FB      | 0.002      | 0.06       | 0.10       |
| CA      | 0.07       | 0.27       | 0.28       |

Figure 7: The number of the support update

Figure 8: Scalability on synthetic random graphs
**Exp-3: Tightness of \( \phi_{\tau}(e,G) \).** In this experiment, we evaluate the tightness of the lower bound \( \phi_{\tau}(e,G) \) in HOTD\(^+\). We use the widely adopted approximation error (AE) to measure the tightness [Golub and Loan, 1996]. The approximation error for \( \phi_{\tau}(e,G) \) is defined as follows:

\[
AE(\phi_{\tau}(e,G)) = \frac{\sum_{i=1}^{m} | \phi_{i}(e,G) - \phi_{\tau}(e,G) |}{\phi_{\tau}(e,G)},
\]

(1)

As shown in Table 2 for \( \phi_{\tau}(e,G) \), 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 information. When \( \tau = 4 \), it shows a trend similar to that of \( \tau = 3 \). Therefore, \( \phi_{\tau}(e,G) \) 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 [Holtgrewe et al., 2010; Meyerhenke et al., 2017]. For the power-law graph, we generates 8 graphs varying the number of vertices from \( 3M \) to \( 10M \) using the method in [Lancichinetti et al., 2008] (average degree with 10, other parameters with default setting). Figure 8 and Figure 9 show the running time of algorithms on random synthetic graphs and power-law synthetic graphs, respectively.

Figure 8 and Figure 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.

**Exp-5: Case study.** In this experiment, we compare \((k,\tau)\)-truss with \((k,h)\)-core which is the existing most similar cohesive subgraph model with our model. Figure 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 highschool. Polotical Books is a network of books, edges between books represent frequent co-purchasing of books by the same buyers. As shown at Figure 10(a) 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. Figure 10(b) shows the similar phenomenon on Polotical Books dataset. It’s because a
(k+1,τ)-truss is a (k,τ)-core but not vice versa, τ=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 [Luce, 1950a; Seidman, 1983; Bonchi et al., 2019; Cohen, 2008; Abello et al., 2002; Pei et al., 2005; Wang et al., 2010; Zhou et al., 2012; Wen et al., 2019]. Among these cohesive subgraph models, the k-truss model has received considerable attention. k-truss model is first introduced in [Cohen, 2008]. In [Cohen, 2008], an in-memory algorithm to detect the k-truss for a given k with time complexity $O(\sum_{v \in Y(G)} d^2_t(v,G))$ is proposed. [Shao et al., 2014] studies the problem of detecting the k-truss for a given k in distribute environments. By using the triangle counting techniques proposed in [Lattapy, 2008], [Wang and Cheng, 2012] proposes a truss decomposition algorithm with time complexity $O(m^{3/2})$. [Wang and Cheng, 2012] also investigates the external-memory algorithms to conduct the truss decomposition. [Zhang and Yu, 2019] proposes an efficient algorithm to maintain the truss decomposition results in evolving graphs. [Huang et al., 2016] 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 [Luce, 1950a], k-core [Seidman, 1983; Bonchi et al., 2019] and k-truss [Cohen, 2008] are three most fundamental cohesive subgraph models. For clique model, [Luce, 1950b; Moradi and Balasundaram, 2018] propose two distance-generalized clique models, named h-clique and h-club. 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 shortest path is within an h-club while it is not strictly necessary for h-clique. For k-core model, [Batagelj and Zaveršnik, 2011] proposes (k,h)-core model which requires each vertex in it has more than k neighbors within h-hop neighborhood. [Bonchi et al., 2019] proposes the (k,h)-core decomposition algorithm with better performance compared with [Batagelj and Zaveršnik, 2011]. 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

[Abello et al., 2002] James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In LATIN 2002: Theoretical Informatics, pages 598–612. 2002.

[Abu-El-Haija et al., 2019] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In Proceedings of ICML, pages 21–29, 2019.

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

[Andrade et al., 2006] Roberto F. S. Andrade, José G. V. Miranda, and Thierry Petit Lobão. Neighborhood properties of complex networks. Phys. Rev. E, 73:046101, Apr 2006.

[Andrade et al., 2008] Roberto FS Andrade, José GV Miranda, Suani TR Pinho, and Thierry Petit Lobao. Characterization of complex networks by higher order neighborhood properties. The European Physical Journal B, 61(2):247–256, 2008.

[Batagelj and Zaveršnik, 2011] V. Batagelj and M.Zaveršnik. Fast algorithms for determining (generalized) core groups in social networks. Adv Data Anal Classif, 5:129–145, 2011.

[Bonchi et al., 2019] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. Distance-generalized core decomposition. In Proceedings of SIGMOD, pages 1006–1023, 2019.

[Chang and Qin, 2018] Lijun Chang and Lu Qin. Cohesive Subgraph Computation over Large Sparse Graphs Algorithms, Data Structures, and Programming Techniques. Springer, 2018.

[Chang et al., 2013] Lijun Chang, Jeffrey Xu Yu, Lu Qin, Xuemin Lin, Chengfei Liu, and Weifa Liang. Efficiently computing k-edge connected components via graph decomposition. In Proceedings of the SIGMOD, pages 205–216, 2013.

[Chen et al., 2020] Zi Chen, Long Yuan, Xuemin Lin, Lu Qin, and Jianye Yang. Efficient maximal balanced clique enumeration in signed networks. In WWW, pages 339–349. ACM / IW3C2, 2020.
[Cohen, 2008] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. In Technical report, National Security Agency, 2008.

[Colomer-de Simón et al., 2013] Pol Colomer-de Simón, M Angeles Serrano, Mariano G Beiró, J Ignacio Alvarez-Hamelin, and Marián Boguná. Deciphering the global organization of clustering in real complex networks. Scientific reports, 3:2517, 2013.

[Eades et al., 2017] Peter Eades, Seok-Hee Hong, An Nguyen, and Karsten Klein. Shape-based quality metrics for large graph visualization. Journal of Graph Algorithms and Applications, 21(1):29–53, 2017.

[Ellson et al., 2002] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, Graph Drawing, pages 483–484. Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Golub and Loan, 1996] Gene H. Golub and Charles F. Van Loan. Matrix Computations. The Johns Hopkins University Press, 1996.

[Holtgrewe et al., 2010] Manuel Holtgrewe, Peter Sanders, and Christian Schulz. Engineering a scalable high quality graph partitioner. In 24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, Georgia, USA, 19-23 April 2010 - Conference Proceedings, pages 1–12. IEEE, 2010.

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

[Huang et al., 2016] Xin Huang, Wei Lu, and Laks V. S. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In Proceedings of SIGMOD, pages 77–90, 2016.

[Lancichinetti et al., 2008] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. In Phys. Rev. E 78, 2008.

[Latapy, 2008] Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. Theory Computer Science, 407(1-3):458–473, 2008.

[Lee et al., 2010] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu C. Aggarwal. A survey of algorithms for dense subgraph discovery. In Managing and Mining Graph Data, pages 303–336. 2010.

[Liu et al., 2019a] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. Efficient (α, β)-core computation: an index-based approach. In WWW, pages 1130–1141. ACM, 2019.

[Liu et al., 2019b] Songtao Liu, Lingwei Chen, Hanze Dong, Zihao Wang, Dinghao Wu, and Zengfeng Huang. Higher-order weighted graph convolutional networks. CoRR, abs/1911.04129, 2019.

[Liu et al., 2020a] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. Efficient (α, β)-core computation in bipartite graphs. VLDB J., 29(5):1075–1099, 2020.

[Liu et al., 2020b] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. Truss-based community search over large directed graphs. In Proceedings of SIGMOD, pages 2183–2197, 2020.

[Luce and Perry, 1949] R Duncan Luce and Albert D Perry. A method of matrix analysis of group structure. Psychometrika, 14(2):95–116, 1949.

[Luce, 1950a] R Duncan Luce. Connectivity and generalized cliques in sociometric group structure. Psychometrika, 15(2):169–190, 1950.

[Luce, 1950b] R.D Luce. Connectivity and generalized cliques in sociometric group structure. Psychometrika, 15:169–190, 1950.

[Meyerhenke et al., 2017] Henning Meyerhenke, Peter Sanders, and Christian Schulz. Parallel graph partitioning for complex networks. IEEE Trans. Parallel Distributed Syst., 28(9):2625–2638, 2017.

[Mones et al., 2012] Enys Mones, Lilla Vicsék, and Tamás Vicsék. Hierarchy measure for complex networks. PLOS ONE, 7, 03 2012.

[Moradi and Balasundaram, 2018] Esmaeel Moradi and Balabbashkar Balasundaram. Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts. Optim. Lett., 12(8):1947–1957, 2018.

[Orsini et al., 2013] Chiara Orsini, Enrico Gregori, Luciano Lenzini, and Dmitri Krioukov. Evolution of the internet k-dense structure. IEEE/ACM Transactions on Networking, 22(6):1769–1780, 2013.

[Ouyang et al., 2020] Dian Ouyang, Long Yuan, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees. Proc. VLDB Endow., 13(5):602–615, 2020.

[Pei et al., 2005] Jian Pei, Daxin Jiang, and Aidong Zhang. On mining cross-graph quasi-cliques. In Proceedings of SIGKDD, pages 228–238, 2005.

[Sahu et al., 2017] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing. Proc. VLDB Endow., 11(4):420–431, 2017.

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

[Shao et al., 2014] Yingxia Shao, Lei Chen, and Bin Cui. Efficient cohesive subgraphs detection in parallel. In Proceedings of SIGMOD, pages 613–624, 2014.

[Sun et al., 2020] Zequn Sun, Chengming Wang, Wei Hu, Muhao Chen, Jian Dai, Wei Zhang, and Yuzhong Qu. Knowledge graph alignment network with gated multi-hop neighborhood aggregation. In Proceedings of AAAI, pages 222–229, 2020.
[Wang and Cheng, 2012] Jia Wang and James Cheng. Truss decomposition in massive networks. *Proceedings of VLDB Endow.*, 5(9):812–823, 2012.

[Wang et al., 2010] Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony K. H. Tung. On triangulation-based dense neighborhood graphs discovery. *Proceedings of VLDB Endow.*, 4(2):58–68, 2010.

[Wen et al., 2019] Dong Wen, Lu Qin, Ying Zhang, Lijun Chang, and Ling Chen. Enumerating k-vertex connected components in large graphs. In *Proceedings of ICDE*, pages 52–63, 2019.

[Wu et al., ] Xudong Wu, Long Yuan, Xuemin Lin, Shiyu Yang, and Wenjie Zhang. Towards efficient k-tripeak decomposition on large graphs. In *DASFAA*.

[Xue et al., 2020] Hui Xue, Xin-Kai Sun, and Wei-Xiang Sun. Multi-hop hierarchical graph neural networks. In *Proceedings of BigComp*, pages 82–89, 2020.

[Yuan et al., 2016a] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Diversified top-k clique search. *VLDB J.*, 25(2):171–196, 2016.

[Yuan et al., 2016b] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. I/O efficient ECC graph decomposition via graph reduction. *Proc. VLDB Endow.*, 9(7):516–527, 2016.

[Yuan et al., 2017] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Effective and efficient dynamic graph coloring. *Proc. VLDB Endow.*, 11(3):338–351, 2017.

[Yuan et al., 2018] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. Index-based densest clique percolation community search in networks. *IEEE Trans. Knowl. Data Eng.*, 30(5):922–935, 2018.

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

[Zhou et al., 2012] Rui Zhou, Chengfei Liu, Jeffrey Xu Yu, Weifa Liang, Baichen Chen, and Jianxin Li. Finding maximal k-edge-connected subgraphs from a large graph. In *Proceedings of EDBT*, pages 480–491, 2012.