



Towards Efficient k -TriPeak Decomposition on Large Graphs

Xudong Wu¹, Long Yuan²(✉), Xuemin Lin³, Shiyu Yang¹, and Wenjie Zhang³

¹ East China Normal University, Shanghai, China
xdwu@stu.ecnu.edu.cn, syyang@sei.ecnu.edu.cn

² Nanjing University of Science and Technology, Nanjing, China
longyuan@njust.edu.cn

³ The University of New South Wales, Sydney, Australia
lxue@cse.unsw.edu.au, wenjie.zhang@unsw.edu.au

Abstract. Analyzing the structure of real-world networks has attracted much attention over years and cohesive subgraph models are commonly used to characterize the structure of a network. Recently, a model named k -Peak is proposed to address the issue failing to detect sparser regions if the network contains distinct regions of different densities in the cohesive subgraph models. However, k -Peak only considers the edge connection (i.e., degree) in the network and the loose structure restricts the effectiveness of the k -Peak. On the other hand, triangles are fundamental building blocks of a network and are widely used in the literature. Motivated by this, in this paper, we propose the k -TriPeak model based on the triangles and study the problem of k -TriPeak decomposition that computes the k -TriPeak for all possible k values to understand the structure of a network. Through investigating the drawbacks of the baseline algorithm following the idea of k -Peak decomposition, we devise a new efficient algorithm to perform the k -TriPeak decomposition. Our new algorithm adopts a top-down decomposition paradigm and integrates two novel upper bounds with which large unnecessary computation can be pruned. We conduct extensive experiments on several large real-world datasets and the experimental results demonstrate the efficiency and effectiveness of our proposed algorithm.

1 Introduction

Due to the rapid development of information technology, we are witnessing the proliferation of graph data based applications over recent years. This has led to huge research efforts devoted to real-world network analytics [2, 4–6, 10, 16, 23, 27]. Among them, identifying cohesive subgraphs to characterize the structure of real-world networks has been extensively studied. Observing that the cohesive subgraph models are often computed globally and fail to detect sparser regions if the network contains distinct regions of different densities, Govindan et al. proposed a new model named k -Peak recently [7]. By conducting the k -Peak decomposition (compute the k -Peak for all possible k values in the graph), [7]

can divide the graph into separate 'mountains' and can find the centers of distinct regions in the graph.

Motivation. The k -Peak decomposition addresses the issue of neglecting sparser regions in the existing cohesive subgraph model [7], however, since k -Peak only considers the edge connection (i.e., degree) between nodes in the subgraph, the returned results are often not that cohesive [5, 14, 28]. The returned loose structure restricts the effectiveness of the k -Peak decomposition. On the other hand, triangles are higher-order connectivity structure than degree [3, 19] and are known as fundamental building blocks of a network [12, 17, 18]. Therefore, triangles are commonly treated as the building blocks for the cohesive subgraph model in the literature [5, 8, 16, 27]. Motivated by this, in this paper, we propose a new model named k -TriPeak based on the triangles and study the k -TriPeak decomposition problem. The model inherits the ability to find the centers of distinct regions of k -Peak model and avoids the problem of incohesiveness for the returned result. Formally, given a graph G , the support of an edge is the number of triangles containing it. A k -TriContour of G is the largest subgraph of G such that (i) the support of edges in it is at least $k - 2$; (ii) the k -TriContour does not include edges from a higher TriContour. The k -TriPeak of G is the union of j -TriContours, where $j \geq k$. And k -TriPeak decomposition computes the k -TriPeak for all possible k values.

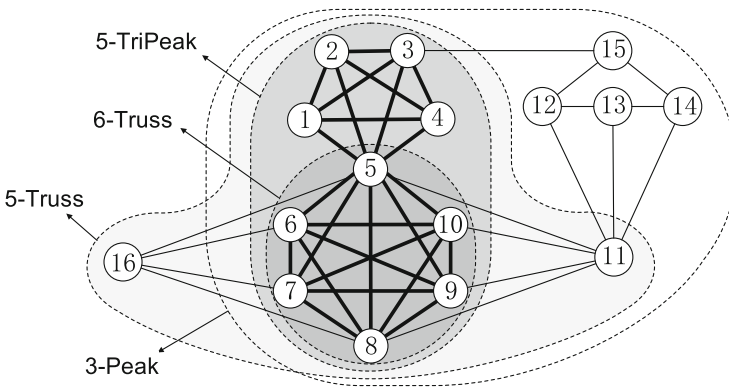


Fig. 1. k -Peak vs k -TriPeak

Example 1. Consider the graph in Fig. 1, we show the 3-Peak and 5-TriPeak of G (The counterpart of 3-Peak is 5-TriPeak since the support of edges is at least $k - 2$ in the definition of k -TriContour). As shown in Fig. 1, since 3-Peak only considers the edge connection between nodes, the incohesive subgraph induced by v_{11}, \dots, v_{15} is also returned. On the contrary, this incohesive subgraph is filtered out by the 5-TriPeak. In the literature, another similar cohesive subgraph model defined based triangles, k -truss, is also studied [5]. It is defined as the largest

subgraph of G in which every edge is contained in at least $(k-2)$ triangles within the subgraph. The problem of k -truss is that it is unable to find the centers of distinct regions in the graph. As shown in Fig. 1, two centers of regions exist in G , namely $\{v_1, \dots, v_5\}$, $\{v_5, \dots, v_{10}\}$. For k -truss model, if we return the 6-truss, the center $\{v_1, \dots, v_5\}$ will be missed; if we return 5-truss, the two centers are returned but the returned result also contains node v_{11} and v_{16} and these two nodes are loosely connected with each other and are not what we want. On the contrary, 5-TriPeak can find these two centers.

Applications. k -TriPeak decomposition can be used in many applications. For example, in the community detection, since k -TriPeak model can find central regions with different densities, those sparser communities in the graph will not be missed if the k -TriPeak model is adopted [8, 14]. Similar to k -Peak decomposition, k -TriPeak decomposition can also be used to visualize the graph through the mountain plot technique presented in [7]. Moreover, understanding the hierarchical structure facilitates graph-topology analysis [1, 22]. The k -TriPeaks of a graph for all k values form a hierarchical structure. It is clear that k -TriPeak decomposition is helpful for understanding the hierarchical structure in a graph.

Our Approach. To perform the k -TriPeak decomposition, a direct approach is following the idea of k -Peak decomposition in [7]. In [7], the k -Peak decomposition is achieved by iteratively computing the k -core with maximum k value in the graph through the k -core decomposition algorithm [2] and removing the computed k -core until the graph is empty. Following the idea, we can implement the k -TriPeak decomposition through k -truss decomposition algorithm [15] in a similar way. However, as analysed in Sect. 3, lots of unnecessary edges will involve in the expensive k -truss decomposition procedure, which leads to the inefficiency of this direct approach. To address the drawback of this approach, in this paper, we propose a new algorithm for the k -TriPeak decomposition. Our new algorithm adopts a top-down decomposition paradigm in which k is explored in decreasing order. Based on this top-down decomposition paradigm, we design two effective but lightweight upper bounding techniques. Using these two upper bounding techniques, we can prune the unpromising edges involving in the expensive k -truss decomposition procedure and the unnecessary computation in the direct approach can be significantly reduced.

Contribution. In this paper, we make the following contributions:

- (1) *The k -TriPeak model to find the centers of distinct regions in the graph.* We investigate the drawbacks of existing k -Peak model and propose a new model, namely k -TriPeak. Based on the k -TriPeak model, we study the problem of k -TriPeak decomposition. To the best of our knowledge, this is the first work to study the problem of efficient k -TriPeak decomposition.
- (2) *An efficient algorithm for k -TriPeak decomposition.* We present an efficient algorithm to perform the k -TriPeak decomposition. In our algorithm, we adopt a top-down decomposition paradigm and devise a static upper bound

and a dynamic upper bound to reduce the unnecessary computation. Moreover, we also explore efficient techniques with which we can maintain the dynamic upper bound in $O(1)$ time for each update during the decomposition process.

(3) *Extensive performance studies on large real-world datasets.* We conduct extensive performance studies using large real-world datasets. The experimental results demonstrate the effectiveness of our proposed model and the efficiency and scalability of the devised decomposition algorithm.

2 Preliminaries

We model a undirected graph as $G(V, E)$, where $V(G)$ represents the set of nodes and $E(G)$ represents the set of edges in G . We denote the number of nodes as n and the number of edges as m , i.e., $n = |V(G)|$ and $m = |E(G)|$. We define the size of G , denoted by $|G|$, as $|G| = m + n$. For a node $u \in V(G)$, we use $\text{nbr}(u, G)$ to denote the neighbor set of u in G , i.e., $\text{nbr}(u, G) = \{v \in V(G) | (u, v) \in E(G)\}$. The degree of a node $u \in V(G)$, denoted by $\text{deg}(u, G)$, is the number of neighbors of u , i.e., $\text{deg}(u, G) = |\text{nbr}(u, G)|$. A triangle in G is a cycle of length 3. In this paper, we omit G in the notations when it is explicit in context.

Definition 1 (Support). *Given a graph G , the support of an edge $e \in E(G)$, denoted by $\text{sup}(e, G)$, is the number of triangles that contain e in G .*

Definition 2 (k -TriContour). *Given a graph G , a subgraph S is the k -TriContour of G , denoted by $C_k(G)$, if (i) $\text{sup}(e, S) \geq k - 2$ for every edge $e \in S$; (ii) the k -TriContour does not include edges from a higher TriContour; (iii) S is maximal, i.e., any subgraph $S' \supset S$ is not a k -TriContour.*

Definition 3 (k -TriPeak). *Given a graph G , a k -TriPeak, denoted by $P_k(G)$, is the union of j -TriContours, where $j \geq k$.*

Definition 4 (TriPeak Number). *The TriPeak number of an edge e in G , denoted by $\kappa(e, G)$, is the value k such that e is contained in the k -TriContour.*

Problem Statement. In this paper, we study the problem of k -TriPeak decomposition that computes the k -TriPeak for all possible k values in the given graph. Since the k -TriPeak consists of the edges with TriPeak number at least k . The k -TriPeak decomposition problem equals to compute the TriPeak number for each edge in the given graph. Therefore, in this paper, we aim to design an efficient algorithm to perform the assignment of TriPeak number to each edge.

Example 2. Consider the graph G illustrated in Fig. 2, we also show its corresponding k -TriContour and k -TriPeak in Fig. 2. For example, for the edge (v_8, v_9) , its support is 3 since it is contained in triangles $\{v_8, v_9, v_6\}, \{v_8, v_9, v_{10}\}, \{v_8, v_9, v_{11}\}$. The 5-TriContour of G is the subgraph induced by nodes $\{v_5, v_6, v_8 \dots v_{15}\}$ except edge (v_5, v_6) and (v_{10}, v_{13}) as this is the maximal subgraph such that the support for each edge in it is 3 and it does not contain any

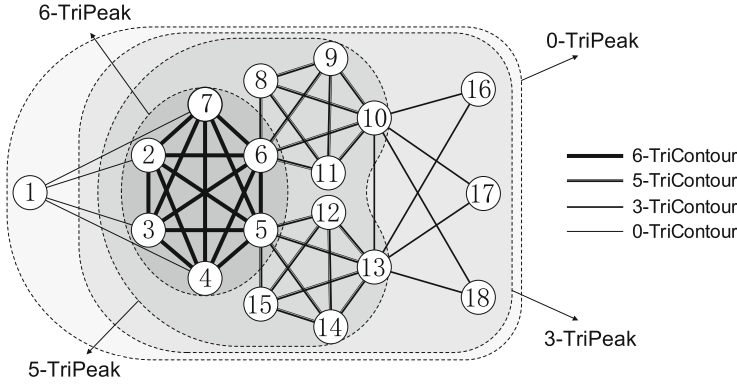


Fig. 2. An example graph G

edges from a k -TriContour with $k > 5$. Note that although (v_1, v_7) has support 3, it is not in the 5-TriContour of G . This is because (v_1, v_7) is contained in $\{v_1, v_7, v_2\}, \{v_1, v_7, v_3\}, \{v_1, v_7, v_4\}$ but edges $(v_7, v_2), (v_7, v_3)$ and (v_7, v_4) are in the 6-TriContour of G . After the TriPeak decomposition, all the TriPeak number of edge can be obtained. For example, $\kappa((v_6, v_7)) = 6$ as it is in the 6-TriContour and $\kappa((v_{10}, v_{13}))$ is 3 as it is in the 3-TriContour.

3 Baseline k -TriPeak Decomposition Algorithm

Inspired by the solution proposed in [7] to perform the Peak decomposition, we present a baseline solution for the TriPeak decomposition problem in this section. In [7], to conduct the Peak decomposition, it iteratively computes the k -core with maximum k value in the graph by the k -core decomposition algorithm [2] and removing the computed k -core until the graph is empty. Following the same idea, we can perform the TriPeak decomposition through the k -truss decomposition [15] based on the following lemma:

Lemma 1. *Given a graph G , let k_{max} be the maximum value such that the corresponding k -truss in G , denoted by $T_{k_{max}}(G)$, is not empty, then $T_{k_{max}}(G) = P_{k_{max}}(G)$.*

Proof. According to Definition 2, $C_{k_{max}}(G) = T_{k_{max}}(G)$ since there doesn't exist $C_k(G)$ with $k > k_{max}$, otherwise such $C_k(G)$ is also a $T_k(G)$ that $k > k_{max}$. And by Definition 3, $P_{k_{max}}(G)$ is the union of $C_j(G)$ where $j \geq k_{max}$. Thus $P_{k_{max}}(G) = C_{k_{max}}(G) = T_{k_{max}}(G)$.

Based on Lemma 1, for a given graph G , the k_{max} -truss and k_{max} -TriPeak in G are the same, which means the TriPeak number of edges in $T_{k_{max}}(G)$ equals to k_{max} exactly. Moreover, based on Definitions 2 and 4, the edges with TriPeak number k have no impact on the TriPeak number of edges whose TriPeak number

Algorithm 1. Baseline(Graph G)

```

1: while not all edges in  $G$  are removed do
2:    $C_{k_{max}} \leftarrow \text{maxTruss}(G)$ ;
3:   for each edge  $e \in C_{k_{max}}$  do
4:      $\kappa(e) \leftarrow k_{max}$ ;
5:     remove  $e$  from  $G$ ;
6: procedure maxTruss(Graph  $G$ )
7:   compute  $\text{sup}(e)$  for each edge  $e \in E(G)$  using the triangle counting algorithm [9];
8:   sort all edges in ascending order of their support;
9:    $k_{max} \leftarrow 2$ ;
10:  while not all edges in  $G$  are removed do
11:    let  $e = (u, v)$  be the edge with the lowest support in  $G$ ; (assume  $\text{deg}(u) \leq \text{deg}(v)$ )
12:     $k \leftarrow \text{sup}(e) + 2$ ;
13:     $k_{max} \leftarrow \text{max}(k_{max}, k)$ ;
14:     $\Phi_{k_{max}} \leftarrow \Phi_{k_{max}} \cup \{e\}$ ;
15:    for each  $w \in \text{nbr}(u)$  do
16:      if  $(v, w) \in G_k$  then
17:         $\text{sup}((u, w)) \leftarrow \text{sup}((u, w)) - 1$ ;
18:         $\text{sup}((v, w)) \leftarrow \text{sup}((v, w)) - 1$ ;
19:        update the new positions of  $(u, w)$  and  $(v, w)$  in the sorted edge array;
20:      remove  $e$  from  $G$ ;
21:  return  $\Phi_{k_{max}}$ ;

```

is k' , where $k' < k$. In other words, for a graph G , if we remove the edges with TriPeak number k from G , the edges with TriPeak number k' in G and the new generated graph G' after the edge removal are the same. Therefore, we can conduct the TriPeak decomposition by iteratively computing the k -truss with the maximum k value and removing the edges in the graph until the graph is empty.

Algorithm. Based on the above analysis, the baseline algorithm, Baseline, is shown in Algorithm 1. The baseline algorithm iteratively computes and removes the k -truss with the maximum k value of G at each iteration (line 2–5). If an edge is contained in k -truss with the maximum k value of G at current iteration, the TriPeak number of it will be assigned (line 4) and it will be removed from G then (line 5). This process is carried out until all edges in G are removed (line 1).

Procedure `maxTruss` computes the k -truss with the maximum k value in G . It first computes the support of each edge in G by the triangle counting algorithm [9] (line 7). Then it sorts all the edges in ascending order of their supports and keep them in an array (line 8). After that, the algorithm iteratively removes the edge e with the lowest support, which is the first edge in the sorted edge array, and add e into the result set of $\Phi_{k_{max}}$ (line 11–14). When removing e , the supports of all other edges that form a triangle with e should be decreased, and their new positions in the sorted edge array should be updated (line 15–20). This algorithm terminates after all edges in G are removed (line 10) and returns the k -truss with the maximum k value (line 21). [15] shows that the time complexity of procedure `maxTruss` is $O(m^{1.5})$.

Drawbacks of the Baseline Solution. In the baseline algorithm, we conduct the TriPeak decomposition through k -truss decomposition iteratively. In each iteration, the edges with $\kappa(e) = k_{max}$ are assigned by computing the k -truss of G with the maximum k value in current iteration (line 4). For a specific iteration, an ideal algorithm is that the computation in this iteration only involves the edges with $\kappa(e) = k_{max}$. However, in the baseline algorithm, all the edges in the remaining graph are taken as the input for the k -truss decomposition algorithm (line 2). Assume that there is an edge e with a small TriPeak number k' in G , it will participate in all iterations computing k -truss where $k \geq k'$ in the baseline algorithm. Therefore, lots of redundant computation exist in the baseline algorithm and it is inefficient to conduct TriPeak decomposition considering the time complexity of k -truss decomposition is $O(m^{1.5})$.

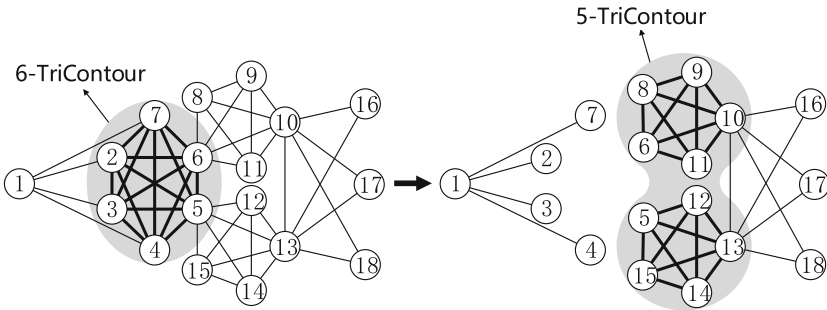


Fig. 3. A running example of Algorithm 1

Example 3. Figure 3 shows a running example of Algorithm 1 on the graph G in Fig. 2. It first performs k -truss decomposition on the whole graph and finds 6-TriPeak. Then, it removes edges in the 6-TriPeak and performs k -truss decomposition on all remaining edges to find the 5-TriContour on the remaining graph. The procedure terminates when all the edges are removed. As shown in Fig. 3, although the TriPeak number of edges incident to $v_1, v_{16}, v_{17}, v_{18}$ and (v_{10}, v_{13}) is not 5, all of these edges involve the k -truss decomposition to compute 5-TriContour on the remaining graph, which leads to the inefficiency of Algorithm 1.

4 Our New Approach

To overcome the drawbacks of the baseline solution, we propose a new paradigm for the TriPeak decomposition problem. In this section, we first present an overview of the new paradigm in Sect. 4.1. Then, we show our concrete techniques in Sects. 4.2 and 4.3, respectively.

4.1 A New Top-Down Decomposition Paradigm

In the baseline algorithm (Algorithm 1), in a specific iteration, we compute the TriPeak number for the edges with $\kappa(e) = k_{max}$, where k_{max} is the maximum k value such that the corresponding k -truss exists in the remaining graph of current iteration. Since the k_{max} for current iteration cannot be determined in advance, it has to conduct the truss decomposition on the graphs consisting of the edges with $\kappa(e) \leq k_{max}$, which leads to the inefficiency of the baseline algorithm for the TriPeak decomposition problem. On the other hand, based on Lemma 1, we know the maximum TriPeak number for all edges in G equals to the maximum k value such that the k -truss exists in the original input graph. For brevity, we denote it as κ_{max} . In other words, we know the TriPeak number for all edges of G is in the range from 1 to κ_{max} . According to the definition of TriPeak decomposition, the essence of the problem is to determine the TriPeak number for each edge. Therefore, to perform the TriPeak decomposition, we can iterate all the possible TriPeak number of the graph in decreasing order based on their values and compute the edges whose TriPeak number equals to the specific TriPeak number. The benefit of this paradigm is that it is possible to prune the edge with $\kappa(e) < k$ in a specific iteration as we know the TriPeak number k to be handled in each iteration in advance. In this way, we can reduce the redundant computation in the baseline algorithm caused by edges with small TriPeak number involving truss decomposition many times.

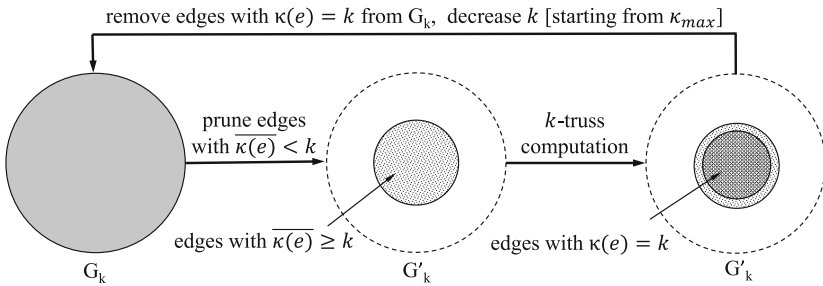


Fig. 4. The new paradigm

Algorithm Framework. Following the above analysis, the new TriPeak decomposition paradigm is illustrated in Fig. 4. Staring from κ_{max} , the paradigm computes the TriPeak number for the edges in decreasing order of k . For a specific k , the edges with $\kappa(e) = k$ are computed. As analysed above, we aim to limit the edges involving the truss computation in this step to the edges with $\kappa(e) = k$. However, this goal is hard to achieve. Therefore, we compute upper bounds of TriPeak number for the edges and use these upper bounds to prune the useless edges. Specifically, when processing a specific k (without loss of generality, we denote the input graph regarding k as G_k), it first prunes the edges whose upper

bound of TriPeak number is less than k , i.e., $\overline{\kappa(e)} < k$. We denote the pruned graph as G'_k . Then, we determine the edges with $\kappa(e) = k$ by computing the k -truss on G'_k . After that, the edges with $\kappa(e) = k$ are removed from G_k and the remaining graph are treated as the input graph for the next iteration. The process terminates when G_k is empty.

4.2 Upper Bounding Techniques

As analysed in Sect. 4.1, the key point for the efficiency of the new paradigm is tight upper bounds of $\kappa(e)$. To achieve this goal, in this part, we will introduce two kinds of upper bounding techniques for $\kappa(e)$.

A Static Upper Bound. Based on the definition of TriPeak number, a direct upper bound of the TriPeak number of the edges in G can be obtained by the truss number:

Definition 5 (Truss Number). *Given a graph G , the truss number of an edge e in G , denoted by $\phi(e)$, is the maximal number of k such that e is contained in a k -truss.*

Lemma 2. *Let e be an edge in G , $\phi(e)$ is the truss number of e , and $\kappa(e)$ is the TriPeak number of e , then $\phi(e) \geq \kappa(e)$.*

Proof. By the definition of k -TriContour, every edge in k -TriContour has no less than $k - 2$ triangles inside k -TriContour, which indicates that the k -TriContour is a part of k -truss. Thus if e is in k -TriContour, it must also be in k -truss. Hence $\phi(e) \geq \kappa(e)$ holds.

The truss number for each edge can be easily obtained through the k -truss decomposition algorithm. However, since k -TriContour does not consider the support from triangles in higher TriContours, as our decomposition paradigm progresses, the pruning power of truss number weakens and the edges with $\phi(e) > \kappa(e)$ accumulate more and more. Therefore, we propose another tight but lightweight upper bound for $\kappa(e)$. The upper bound is defined based on G_k (the input graph of our paradigm when processing a specific k , i.e., the graph after removing all the edges with $\kappa(e) > k$) and is dynamically maintained as our decomposition paradigm progresses.

A Dynamic Upper Bound. Given an edge $(u, v) \in E(G_k)$, $\text{sup}((u, v), G_k)$ be the support of $e = (u, v)$ in G_k . For a node $u \in V(G_k)$, let $h(u, G_k)$ returns the maximum value h such that there exist at least h neighbours v of u with $\text{sup}((u, v), G_k) \geq h$. We define $\lambda(e, G_k) = \min\{\text{sup}(e, G_k), h(u, G_k), h(v, G_k)\} + 2$. And we can prove that for any arbitrary valid k in our paradigm, $\lambda(e, G_k)$ is an upper bound of $\kappa(e, G)$, which is shown in the following lemma:

Lemma 3. *Let e be an edge in G , then $\lambda(e, G_k) \geq \kappa(e, G)$.*

Algorithm 2. TriPeakDecom(Graph G)

```

1: compute  $\kappa_{max}$  and  $\sup(e)$ ,  $\phi(e)$  for all edges by maxTruss in Algorithm 1;
2: compute  $h(u)$  for each node  $u \in V(G)$ ;
3: for each  $e = (u, v) \in E(G)$  do
4:    $\lambda(e) \leftarrow \min\{\sup(e), h(u), h(v)\} + 2$ ;
5:    $\kappa(e) \leftarrow \min\{\lambda(e), \phi(e)\}$ ;
6:  $k \leftarrow \kappa_{max}$ ;  $G_k \leftarrow G$ ;
7: while  $G_k \neq \emptyset$  do
8:    $G'_k \leftarrow \{e | e \in E(G_k), \overline{\kappa(e)} \geq k\}$ ;
9:    $C_{k'} \leftarrow \mathbf{maxTruss}(G'_k)$ 
10:  if  $k' = k$  then
11:     $S \leftarrow \emptyset$ ;
12:    for each edge  $e = (u, v) \in C_{k'}$  (assume  $\deg(u) \leq \deg(v)$ ) do
13:       $\kappa(e) \leftarrow k$ ; remove  $e$  from  $G_k$ ;
14:      update  $h(u)$ ,  $h(v)$ ;
15:      add  $u$  (or  $v$ ) into  $S$  if  $h(u)$  (or  $h(v)$ ) is changed;
16:      for each  $w \in \text{nbr}(u)$  do
17:        if  $(v, w) \in G_k$  then
18:           $\sup((u, w)) \leftarrow \sup((u, w)) - 1$ ;  $\sup((v, w)) \leftarrow \sup((v, w)) - 1$ ;
19:          update  $h(w)$ ,  $h(u)$ ,  $h(v)$  and  $\lambda((u, w))$  and  $\lambda((v, w))$ ;
20:          add  $u$  (or  $w$ ) into  $S$  if  $h(u)$  ( $h(v)$  or  $h(w)$ ) is changed;
21:      for each node  $w$  in  $S$  do
22:        update  $\overline{\kappa(e)}$  for each edge  $e$  incident to  $w$  as line 4-5;
23:   $G_{k-1} \leftarrow G_k$ ;  $k \leftarrow k - 1$ ;

```

Proof. Since $e = (u, v)$ is still remained in G_k , we know $C_{\kappa(e, G)}(G) \subseteq G_k$. And within $C_{\kappa(e, G)}(G)$, we know that $\sup(e, C_{\kappa(e, G)}(G)) + 2 \geq \kappa(e, G)$, $h(u, C_{\kappa(e, G)}(G)) + 2 \geq \kappa(e, G)$ and $h(v, C_{\kappa(e, G)}(G)) + 2 \geq \kappa(e, G)$. Thus $\lambda(e, G_k) = \min\{\sup(e, G_k), h(u, G_k), h(v, G_k)\} + 2 \geq \min\{\sup(e, C_{\kappa(e, G)}(G)), h(u, C_{\kappa(e, G)}(G)), h(v, C_{\kappa(e, G)}(G))\} + 2 \geq \kappa(e, G)$.

4.3 Our k -TriPeak Decomposition Algorithm

In this part, we present our algorithm to conduct the TriPeak decomposition. With the new decomposition paradigm and upper bounding techniques, the only challenge is integrating the upper bounding techniques into decomposition paradigm efficiently, especially the maintenance of upper bound $\lambda(e, G_k)$. This part addresses this challenge.

Algorithm. Our algorithm, TriPeakDecom, is shown in Algorithm 2. It first computes the κ_{max} and initializes the auxiliary information for the upper bounds (line 1–5). Then, it conduct the TriPeak decomposition following the new top-down paradigm until the graph is empty (line 6–23).

Specifically, it first invokes procedure **maxTruss** in Algorithm 1 to compute κ_{max} , $\sup(e)$ and $\phi(e)$ for each edge e ($\phi(e)$ equals to k_{max} when e is removed from G in line 20 of Algorithm 1). Then it computes $h(u)$ for each node u based

on the supports of edges incident to u (line 2). At last, $\lambda(e)$ for each edge e is assigned according to its definition and $\overline{\kappa}(e)$ of each edge e is initialized as $\min\{\lambda(e), \phi(e)\}$ (line 4–5).

Then, it conducts the TriPeak decomposition iteratively starting with $k = \kappa_{max}$ and $G_k = G$ (line 6) and the decomposition terminates when G_k is empty (line 7). In a specific iteration processing k , it first extracts G'_k from G_k with edges of which the upper bound $\overline{\kappa}(e)$ is no less than k (line 8). This step filters out the unpromising edges. Then it computes the $C_{k'}$ of G'_k by the `maxTruss` procedure (line 9). If $k' = k$, for each edge $e = (u, v) \in C'_k$, it assigns $\kappa(e) = k$ and removes the edge from G_k (line 13). The remaining work is to maintain the incorrect $\overline{\kappa}(e)$ caused by the removal of edges. As $\phi(e)$ is fixed in the whole process, we only need to find the edges whose $\lambda(e)$ changes after the edge removal. To achieve this goal, `TriPeakDecom` uses a set S to store the nodes u that $h(u)$ has changed in the iteration since this change may influence $\lambda(e)$ of any edges incident to u (line 11). Regarding a removed edge (u, v) , for u, v and each common neighbor w of u and v , it decreases the support of (u, w) and (v, w) (line 18), updates $h(u)$, $h(v)$ and $h(w)$ (line 14, 19) and $\lambda((u, w))$ and $\lambda((v, w))$ (line 19). If a node u whose $h(u)$ is changed, adds u into S (line 15, 20). At the end of iteration, for each node $w \in S$, it updates $\overline{\kappa}(e)$ for all edges incident to w , since the change of $h(w)$ may change $\overline{\kappa}(e)$ (line 21–22). When an iteration finishes, k is decreased and the remaining edges are taken as the input graph for the next iteration (line 23).

Efficient Maintenance of $\lambda(e, G_k)$. In Algorithm 2, we maintain $\lambda(e, G_k)$ dynamically as the decomposition processes (line 19, 22). Based on the definition of $\lambda(e, G_k)$, for an edge $e = (u, v)$, the key to obtain $\lambda(e, G_k)$ is to compute $h(u, G_k)$ and $h(v, G_k)$. However, the time complexity to compute $h(u, G_k)$ and $h(v, G_k)$ on the fly based on the edge support maintained in Algorithm 2 is at least $O(\max\{\deg(u, G_k), \deg(v, G_k)\})$. Since $h(u, G_k)$ is recomputed frequently in Algorithm 2 (line 14, 19), this approach is inefficient. To improve the efficiency to maintain $h(u)$, for each node u , besides $h(u)$, we maintain the number of edges incident to u with different support values, respectively, i.e., cnt_i^u , which represents the number of edges incident to u with support equals i . Moreover, we also maintain $cnt_{\geq h(u)}^u$ that stores the number of edges incident to u with support not less than $h(u)$. During the decomposition, when the support of an edge $e = (u, v)$ decrease from i to j , we just decrease cnt_i^u and cnt_i^v by 1 and increase cnt_j^u and cnt_j^v by 1. And for the node u (the same as v), if $i \geq h(u)$, $j < h(u)$ and if $cnt_{\geq h(u)}^u > h(u)$, we decrease $cnt_{\geq h(u)}^u$ by 1; and if $cnt_{\geq h(u)}^u = h(u)$, we decrease $h(u)$ by 1 and update $cnt_{\geq h(u)}^u$ as $cnt_{\geq h(u)}^u + cnt_{h(u)}^u - 1$. Otherwise, we just keep $h(u)$ and $cnt_{\geq h(u)}^u$ unchanged. In this way, for the operation updating $h(u)$ regarding a node u in line 14 and 19 in Algorithm 2, we can finish it in $O(1)$ time. As a result, for each edge e , the $\lambda(e, G_k)$ are maintained in $O(1)$ in the decomposition procedure.

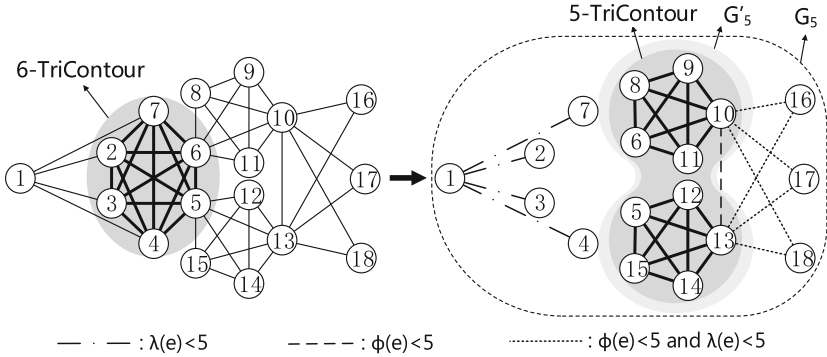


Fig. 5. A running example of Algorithm 2

Example 4. Figure 5 shows a running example of Algorithm 2 on the graph G in Fig. 2. Similar to Algorithm 1, it first performs truss decomposition and finds 6-TriPeak. Then, it removes edges in the 6-TriPeak and performs truss decomposition to find the 5-TriContour. Different from Algorithm 1, when performing the k -truss decomposition to find 5-TriContour, it first prunes the edges with $\overline{\kappa}(e) < 5$. For example, the edge (v_{10}, v_{13}) is pruned by static upper bound as $\phi((v_{10}, v_{13})) < 5$ though $\lambda((v_{10}, v_{13})) \geq 5$; edge (v_1, v_7) is pruned by dynamic upper bound as $\lambda((v_1, v_7)) < 5$ though $\phi((v_1, v_7)) \geq 5$. As illustrated in Fig. 5, Algorithm 2 significantly reduces the number of unnecessary edges involving the procedure of truss decomposition compared with Algorithm 1.

Theorem 1. *Given a graph G , the running time of Algorithm 2 can be bounded by $O(\kappa_{max} \cdot m^{1.5})$.*

Proof. The whole algorithm can be divided into two stages, the initialization stage (line 1–6) and the main iteration stage (line 7–23). Line 1 invokes `maxTruss` procedure using $O(m^{1.5})$ time. Line 2–5 can be done in $O(m)$ time. In the main iteration stage, line 9 takes $O(m^{1.5})$ time. Line 13–15 and line 18–20 can be done in constant time, and line 21–22 requires $O(m)$ time. Now the only question left is what is the number of loops in line 12 and 16. For a certain node u , line 16 is bounded by $\text{deg}(u)$, and line 12 is bounded by $|\text{nbr}_{\geq u}|$, which is the number of neighbors of u whose degree is not smaller than u . Thus line 12–20 can be done in $O(\sum_{u \in G_k} (\text{deg}(u) \cdot |\text{nbr}_{\geq u}|))$ time, which is bounded by $O(m^{1.5})$. This is because if $\text{deg}(u) \leq \sqrt{m}$, $|\text{nbr}_{\geq u}| \leq \text{deg}(u) \leq \sqrt{m}$ and $\sum_{u \in G_k} (\text{deg}(u) \cdot |\text{nbr}_{\geq u}|) \leq m^{1.5}$. If $\text{deg}(u) > \sqrt{m}$, $|\text{nbr}_{\geq u}| \leq \sqrt{m}$ as well for $\text{deg}(u) \cdot |\text{nbr}_{\geq u}| \leq \sum_{v \in |\text{nbr}_{\geq u}|} \text{deg}(v) < 2m$, and $\sum_{u \in G_k} (\text{deg}(u) \cdot |\text{nbr}_{\geq u}|) \leq m^{1.5}$. The number of iterations is bounded by κ_{max} . Thus, the the running time of Algorithm 2 can be bounded by $O(\kappa_{max} \cdot m^{1.5})$.

5 Performance Studies

In this section, we evaluate the effectiveness of our model and the efficiency and scalability of our proposed algorithm. The experiments are conducted on a machine with an Intel Xeon 2.20 GHz CPU and 128 GB memory running Red Hat Linux 4.8.5, 64 bit.

Table 1. Datasets used in experiments

Datasets	Type	Number of nodes	Number of edges	Average degree	κ_{max}
DBLP	Citation	317,080	1,049,866	6.62	114
Livemocha	Social	104,103	2,193,083	42.13	27
Flickr	Misc	105,938	2,316,948	43.74	574
Flixster	Social	2,523,386	7,918,801	6.28	47
Skitter	Computer	1,696,415	11,095,298	13.08	68
LiveJournal	Social	3,997,962	34,681,189	17.35	352

Datasets. In our experiments, we evaluate the algorithms on six publicly available real-world datasets as listed in Table 1. Of these, DBLP and LiveJournal are downloaded from *SNAP*¹, and the others are downloaded from *KONECT*².

Algorithms. We implement and compare the following four algorithms:

- Baseline: Algorithm 1
- TriPeakDecom_s: TriPeak decomposition algorithm with static upper bound only.
- TriPeakDecom_d: TriPeak decomposition algorithm with dynamic upper bound only.
- TriPeakDecom: Algorithm 2

All algorithms are implemented in C++ and compiled with GNU GCC 4.8.5 using optimization level 2. The time cost of the algorithm is measured as the amount of elapsed wall-clock time during the program execution.

Exp-1: Effectiveness. We evaluate the effectiveness of k -TriPeak and k -Peak by examining the quality of detected subgraph via the clustering coefficient [18] metric. Clustering coefficient (CC) indicates the tendency of nodes in a subgraph to cluster together. Thus, high clustering coefficient means high probability that the connections inside the detected subgraph are dense. In this experiment, we find all k -TriPeaks and k -Peaks with different k values and compute the clustering coefficient of them. Since the distributions of k in findings of k -TriPeak and k -Peak are quite different, here we compare the clustering coefficient of k_1 -TriPeak and k_2 -Peak of similar size even if $k_1 \neq k_2$. The results are shown in

¹ <http://snap.stanford.edu/>.

² <http://konect.uni-koblenz.de/>.

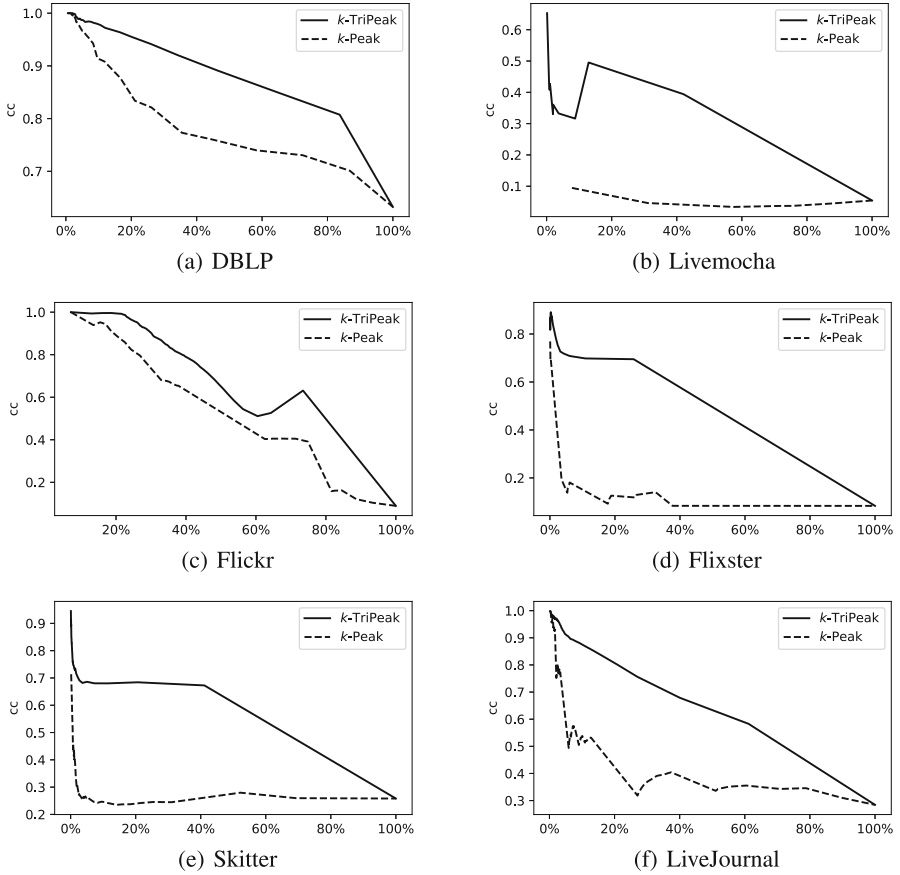


Fig. 6. Effectiveness of k -TriPeak and k -Peak

Fig. 6. In Fig. 6, the horizontal coordinate denotes the size of k -TriPeak (k -Peak) as a percentage of total graph.

As shown in Fig. 6, as the size of k -TriPeak (k -Peak) increases, the value of clustering coefficient for both of them generally decreases. However, it can be observed that, of similar size, the k -TriPeak is much denser than k -Peak. For example, in *Flixster* dataset, the k -TriPeak of which the size is around 30% of the total graph has clustering coefficient of 0.7, while the k -Peak of similar size only have 0.1. This is because k -TriPeak takes a high-order connectivity structure as the building blocks while k -Peak only considers degree. These results indicate that compared with the k -Peak model, the returned result of k -TriPeak are more cohesive and k -TriPeak is better cohesive subgraph model compared with k -Peak.

Table 2. Running time on real-world datasets

Alg	Dataset					
	DBLP	Livemocha	Flickr	Flixster	Skitter	LiveJournal
Baseline	81.85s	178.5 s	3538.5 s	1017.25 s	1993.32 s	29768.37 s
TriPeakDecom _s	19.69 s	62.1 s	2020.35 s	194.41 s	545.78 s	5530.65 s
TriPeakDecom _d	15.56 s	98.67 s	1228.3 s	285.02 s	615.49 s	5038.35 s
TriPeakDecom	14.75 s	58.29 s	834.79 s	150.9 s	471.26 s	2653.01 s

Exp-2: Efficiency. In this experiment, we compare the total processing time of those four algorithms on six real-world datasets. The results are reported in Table 2.

Generally, the processing time increases as the size of the graph increases. Baseline takes the most time on all six datasets. It spends more than 8 hours to perform the TriPeak decomposition on LiveJournal dataset. The reason for Baseline’s long running time is that edges with small TriPeak number, which make up a large portion of the whole graph, take participate in the k -truss decomposition for the big TriPeak number many times. The algorithms solely adopting static or dynamic upper bounding technique run much faster than Baseline. TriPeakDecom_s is faster than TriPeakDecom_d on Livemocha, Flixster and Skitter and the opposite on DBLP, Flickr and LiveJournal, for they play to their strength on different stages in a decomposition. TriPeakDecom algorithm, which adopts both upper bounding techniques, achieves the best performance on all six datasets. For example, on LiveJournal, it achieves an order of magnitude faster than Baseline.

Exp-3: Scalability. We study the scalability of the four algorithms in this experiment. To test the scalability, we randomly sample the nodes and edges respectively of two largest datasets *Skitter* and *LiveJournal* from 20% to 100% and take the induced subgraph as the input graph. The results are shown in Fig. 7.

As shown in Fig. 7, as the size of the graph increases, the processing times of four algorithms increase due to the increasing of the number of iterations and the number of involved edges in each iteration. Moreover, as the size of the graph increases, the gap in processing times between Baseline and other three algorithms increases. This is because the unnecessary computation on edges, which are reduced by other three algorithms but remained in Baseline, make up larger portion of computation when the size of the graph grows. The gap in processing times between TriPeakDecom_s and TriPeakDecom_d remains small on both datasets. The TriPeakDecom algorithm consumes the least time and its processing time grows the most stably on all datasets. The results show the good scalability of our proposed algorithm.

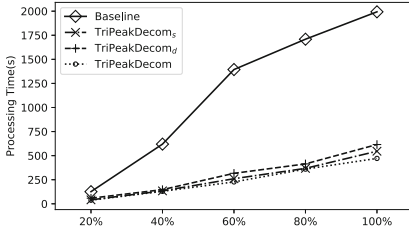
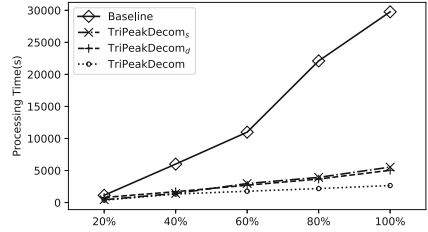
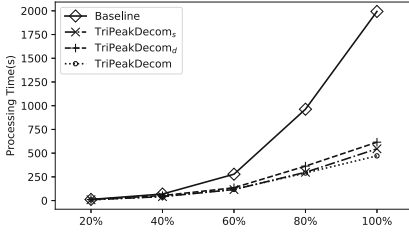
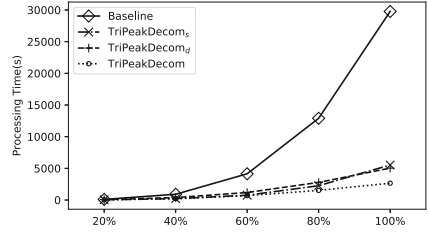
(a) Skitter (Vary $|V|$)(b) LiveJournal (Vary $|V|$)(c) Skitter (Vary $|E|$)(d) LiveJournal (Vary $|E|$)

Fig. 7. Scalability testing

6 Related Work

The most related works to k -TriPeak are k -Peak [7] and k -truss [5], which have been introduced in Sect. 1. Since k -Peak is defined based on degree, the returned results of k -Peak are often not that cohesive [5, 14, 28] compared with the k -TriPeak [14]. The difference between k -TriPeak and k -truss is that in a k -truss, triangles containing edges from the higher k -truss are taken into consideration while in a k -TriPeak, all these edges are filtered out. This difference leads to it that k -TriPeak is able to find the centers of distinct regions in a graph as k -Peak [7].

Besides k -Peak and k -truss, there are many different models proposed in the literature. One of the most intuitive cohesive subgraph models is the clique model in which each node is adjacent to every other node [4]. More complex models based on the clique model [20, 21, 25] are also studied in the literature. However, clique is often too restrictive for many applications, thus, more clique relaxation models have been proposed, such as the k -plex [13], n -clan and n -club [11]. Nevertheless, these models always face the problem of computational intractability. To address this problem, more polynomial time solvable cohesive subgraph models are proposed recently, such as k -core [2], triangle k -core [27], (k, s) -core [26], DN-Graph [16], k -edge connected component [22, 24] and k -mutual-friend subgraph model [28].

7 Conclusion

Motivated by the recent proposed k -Peak model, in this paper, we propose a new k -TriPeak model based on the triangles in the graph and study the k -TriPeak decomposition problem. To perform the k -TriPeak decomposition, we first present an approach following the idea of k -Peak decomposition. However, this approach involves lots of unnecessary computation. Therefore, we propose a new top-down paradigm to conduct the decomposition. Based on the new paradigm, we devise two effective upper bounds to prune the unnecessary edges involving computation in the baseline approach. Moreover, we explore efficient techniques to maintain the upper bounds during the decomposition. We conduct experiments on large real-world datasets and the experimental results demonstrate the efficiency and effectiveness of our proposed algorithm.

References

1. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *NHM* **3**(2), 371–393 (2008)
2. Batagelj, V., Zaveršnik, M.: An $o(m)$ algorithm for cores decomposition of networks. *Comput. Sci.* **1**(6), 34–37 (2003)
3. Benson, A.R., Gleich, D.F., Leskovec, J.: Higher-order organization of complex networks. *Science* **353**(6295), 163–166 (2016)
4. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* **16**(9), 575–576 (1973)
5. Cohen, J.: Trusses: cohesive subgraphs for social network analysis. National Security Agency Technical Report (2008)
6. Feng, X., Chang, L., Lin, X., Qin, L., Zhang, W., Yuan, L.: Distributed computing connected components with linear communication cost. *Distrib. Parallel Databases* **36**(3), 555–592 (2018)
7. Govindan, P., Wang, C., Xu, C., Duan, H., Soundarajan, S.: The k -peak decomposition: mapping the global structure of graphs. In: *Proceedings of WWW*, pp. 1441–1450 (2017)
8. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k -truss community in large and dynamic graphs. In: *Proceedings of SIGMOD*, pp. 1311–1322 (2014)
9. Latapy, M.: Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.* **407**(1), 458–473 (2008)
10. Luce, R.D., Perry, A.D.: A method of matrix analysis of group structure. *Psychometrika* **14**(2), 95–116 (1949)
11. Mokken, R.J.: Cliques, clubs and clans. *Qual. Quant.* **13**(2), 161–173 (1979)
12. Newman, M.E.J., Watts, D.J., Strogatz, S.H.: Random graph models of social networks. *PNAS* **99**, 2566–2572 (2002)
13. Seidman, S.B., Foster, B.L.: A graph-theoretic generalization of the clique concept. *J. Math. Sociol.* **6**(1), 139–154 (1978)
14. Shao, Y., Chen, L., Cui, B.: Efficient cohesive subgraphs detection in parallel. In: *Proceedings of SIGMOD*, pp. 613–624 (2014)
15. Wang, J., Cheng, J.: Truss decomposition in massive networks. *PVLDB* **5**(9), 812–823 (2012)

16. Wang, N., Zhang, J., Tan, K., Tung, A.K.H.: On triangulation-based dense neighborhood graphs discovery. *PVLDB* **4**(2), 58–68 (2010)
17. Wasserman, S., Faust, K.: *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge (1994)
18. Watts, D.J., Strogatz, S.H.: Collective dynamics of ‘small-world’ networks. *Nature* **393**(6684), 440 (1998)
19. Yin, H., Benson, A.R., Leskovec, J., Gleich, D.F.: Local higher-order graph clustering. In: *Proceedings of SIGKDD*, pp. 555–564 (2017)
20. Yuan, L., Qin, L., Lin, X., Chang, L., Zhang, W.: Diversified top-k clique search. In: *Proceedings of ICDE*, pp. 387–398 (2015)
21. Yuan, L., Qin, L., Lin, X., Chang, L., Zhang, W.: Diversified top-k clique search. *VLDB J.* **25**(2), 171–196 (2016)
22. Yuan, L., Qin, L., Lin, X., Chang, L., Zhang, W.: I/O efficient ECC graph decomposition via graph reduction. *PVLDB* **9**(7), 516–527 (2016)
23. Yuan, L., Qin, L., Lin, X., Chang, L., Zhang, W.: Effective and efficient dynamic graph coloring. *PVLDB* **11**(3), 338–351 (2017)
24. Yuan, L., Qin, L., Lin, X., Chang, L., Zhang, W.: I/O efficient ECC graph decomposition via graph reduction. *VLDB J.* **26**(2), 275–300 (2017)
25. Yuan, L., Qin, L., Zhang, W., Chang, L., Yang, J.: Index-based densest clique percolation community search in networks. *IEEE TKDE* **30**(5), 922–935 (2018)
26. Zhang, F., Yuan, L., Zhang, Y., Qin, L., Lin, X., Zhou, A.: Discovering strong communities with user engagement and tie strength. In: Pei, J., Manolopoulos, Y., Sadiq, S., Li, J. (eds.) *DASFAA 2018*. LNCS, vol. 10827, pp. 425–441. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91452-7_28
27. Zhang, Y., Parthasarathy, S.: Extracting analyzing and visualizing triangle k-core motifs within networks. In: *Proceedings of ICDE* (2012)
28. Zhao, F., Tung, A.K.: Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB* **6**(2), 85–96 (2012)