World Wide Web (2020) 23:799–830
https://doi.org/10.1007/s11280-019-00725-6

Towards *k*-vertex connected component discovery from large networks

Yuan Li[1] · **Guoren Wang**[2] · **Yuhai Zhao**[3] · **Feida Zhu**[4] · **Yubao Wu**[5]

Received: 30 May 2019 / Revised: 22 August 2019 / Accepted: 27 August 2019 /
Published online: 7 November 2019

Abstract

In many real life network-based applications such as social relation analysis, Web analysis, collaborative network, road network and bioinformatics, the discovery of components with high connectivity is an important problem. In particular, k-edge connected component (k-ECC) has recently been extensively studied to discover disjoint components. Yet many real scenarios present more needs and challenges for overlapping components. In this paper, we propose a k-vertex connected component (k-VCC) model, which is much more cohesive, and thus supports overlapping between components very well. To discover k-VCCs, we propose three frameworks including top-down, bottom-up and hybrid frameworks. The top-down framework is first developed to find the exact k-VCCs by dividing the whole network. To further reduce the high computational cost for input networks of large sizes, a bottom-up framework is then proposed to locally identify the seed subgraphs, and obtain the heuristic k-VCCs by expanding and merging these seed subgraphs. Finally, the hybrid framework takes advantages of the above two frameworks. It exploits the results of bottom-up framework to construct the well-designed mixed graph and then discover the exact k-VCCs by contracting the mixed graph in a top-down way. Because the size of mixed graph is smaller than the original network, the hybrid framework runs much faster than the top-down framework. Comprehensive experimental are conducted on large real and synthetic networks and demonstrate the efficiency and effectiveness of the proposed exact and heuristic approaches.

Keywords k-vertex connected component(k-VCC) · Component detection · Large network

This article belongs to the Topical Collection: *Special Issue on Graph Data Management in Online Social Networks*
Guest Editors: Kai Zheng, Guanfeng Liu, Mehmet A. Orgun, and Junping Du

✉ Yuan Li
neu_liyuan@126.com

Extended author information available on the last page of the article.

1 Introduction

Component detection is a fundamental problem [18, 36] in the analysis of large-scale networks. Vertices in one component are more highly connected with each other than the rest of the network. Thus, finding highly connected components can benefit many real life applications. For example, groups of intimate entities discovered in social networks can be exploited to analyze their social behaviors [18, 32]; a set of Web pages with similar contents in Web data network can be indexed together to facilitate Web search [10]; clusters of interactive genetic markers discovered in genetic interaction networks can be utilized to infer the corresponding cause of diseases [46].

The existing methods of component detection can be roughly divided into two main categories, i.e. clique-based methods and clique-relaxed methods. According to differently relaxed constraints, clique-relaxed methods can be further divided into degree-relaxed [4, 9, 49], distance-relaxed [25, 33] and triangulation-relaxed [11, 22, 23, 42, 43] methods. Although succeeding to some extent, these methods still have respective drawbacks.

A toy co-friendship network is considered in Figure 1a. Degree-relaxed and distance-relaxed methods often consider the network as an indivisible whole, saying a k-core with $k = 3$ and a n-club with $n = 3$, since the degree of every vertex is no less than 3 and the maximum length of shortest pathes of all pairs of vertices is no larger than 3, respectively. This contradicts with the intuition that Figure 1a should be disconnected into two components by deleting edge e. The reason is that these two methods only concern about the degree and distance but ignore the connectivity of any pair of vertices. Although triangulation-relaxed methods can detect these two components, they do not work when there are no triangles in graph, e.g. bipartite graphs.

Connectivity is measured by the number of disjoint paths between vertices. Intuitively, high connectivity would contribute to the robustness and steadiness of the component. For example, in the collaborative network, the vertices represent experts and the edges represent the relationships among them. The connectivity of a group of experts often measures their ability to finish a task together [26, 39]. Thus, the high connectivity of a component in this network guarantees its robustness because even if a few experts quit or relationships disappear, the task could still be completed. Similarly, in the road network, the vertices represent places and the edges represent the roads between places. The high connectivity among a set of places increases the intensity of road network. Even if some of the roads are congested, it can still be reached from other routes [29, 30, 50–53]. Also, in the genetic interaction network, the connectivity of a set of genetic markers ensures the successful expression of special gene functions. Therefore, the component of genetic markers with higher connectivity possess more stable functional characteristic [46].

A component with high connectivity could still be connected, even if losing a few relationships or entities. Therefore, in recent years, connectivity-based component detection

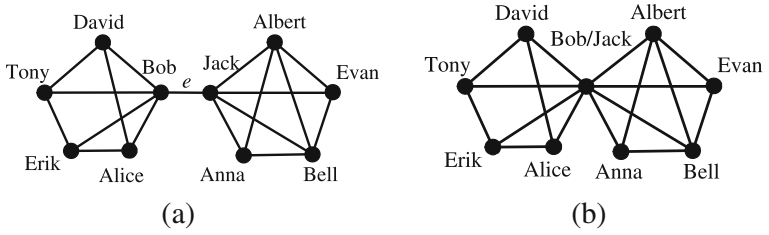

Figure 1 A toy co-friendship network

such as k-edge connected components (k-ECC) [2, 6, 7, 21, 41, 54] has drawn a lot of attention. A k-ECC refers to a maximal subgraph, the remaining subgraph of which is still connected after any $k-1$ edges are removed from it. A toy co-friendship network is considered in Figure 1a as an example. The network in Figure 1a is a 1-ECC. Since the low edge connectivity, it is naturally divided into two separate 3-ECCs, {*Bob, David, Tony, Erik, Alice*} and {*Jack, Anna, Bell, Evan, Albert*}.

However, k-ECC still has its own limitation. For example, if *Bob* is an alias of *Jack*, Figure 1a is equivalent to Figure 1b. In this case, Figure 1b is identified as a whole 3-ECC, although there are practically two separate components, since once we remove the vertex *Bob*/*Jack*, the network becomes disconnected. Thus, high edge connectivity does not necessarily indicate a component of strong connectivity.

In this paper, we study the *k-vertex connected component (k-VCC) detection problem*, which focuses on vertex connectivity instead of edge connectivity, of networks. Given a graph G, the goal is to find all such induced subgraphs, g', of G that g' is still connected after removing any $k-1$ vertices from it and no supergraph of g' has the same property. According to this informal definition, the network in Figure 1b can be identified as two 3-VCCs, {*Bob/Jack, David, Tony, Erik, Alice*} and {*Bob/Jack, Anna, Bell, Evan, Albert*}. Unlike k-ECC, k-VCC has three unique advantages: (1) k-VCC captures more connectivity of networks than k-ECC. According to [15], a component of high vertex connectivity must be of high edge connectivity, but not vice versa; (2) k-VCC allows overlap among different components, say Bob/Jack appears in both the components shown in Figures 1b, which is more natural and reasonable for real-world networks [35]; (3) k-VCC can prevent the detected communities from including irrelevant subgraphs, i.e. free rider effect [45].

Unfortunately, the methods for k-ECC [2, 6] cannot be directly utilized to find k-VCCs. Because each vertex only belongs to at most one k-ECC [6], these methods could obtain k-ECCs by vertex contraction [40]. In our case, however, each vertex can be in more than one k-VCCs, which makes the former trick not work. To this end, we propose three novel frameworks to tackle the k-VCC detection problem, namely top-down, bottom-up and hybrid frameworks for k-VCC detection, respectively. The top-down iteratively divides the networks by finding minimum vertex cut set, which could find all exact k-VCCs. The bottom-up framework, instead of using the entire network structure, locally identifies the seed subgraphs, and obtains the heuristic k-VCCs by expanding and merging these seed subgraphs. Inspired by the above two frameworks, the hybrid framework further considers the group relationships of seed subgraphs, and thus can utilize the elaborate mixed graph structure to discover the exact k-VCCs by contracting the mixed graph in a top-down way. To the best of our knowledge, our conference paper [28] is the first work which discusses k-VCC detection problem. And this work is an extended version of it. All in all, our contributions are summarized as below:

- A novel k-VCC detection problem is proposed from the perspective of vertex connectivity.
- The top-down, bottom-up and hybrid frameworks are respectively developed to solve the k-VCC detection problem. The top-down and hybrid are exact approaches, while the bottom-up framework is heuristic.
- Especially, in the bottom-up framework, a concept of local k-vertex connected subgraph is proposed to enable locally identifying seed subgraphs, which accelerates k-VCC detection. Further, in the hybrid framework, an upper bound of vertex connectivity is designed to speed up the contraction of the mixed graph. In addition, several optimization techniques are proposed to further reduce the search space;

– Extensive experiments on both real and synthetic datasets demonstrate the efficiency and effectiveness of our frameworks and theories.

The rest of our paper is organized as follows. In Section 2 we discuss the related work. We give the notions and problem statement in Section 3. In Sections 4, 5 and 6, we present the top-down, bottom-up and hybrid k-VCC detection frameworks, respectively. Extensive experiments are reported in Section 7. Section 8 concludes this work.

2 Related works

The related work is categorized into three main research lines, including graph connectivity, component detection, and overlapping component detection (OCD) respectively.

Graph connectivity Graph connectivity has an extricably bound with minimum cut, since the minimum cut contributes to the graph connectivity. A large number of algorithms have been designed for computing the global minimum connectivity of the whole graphs [16, 17, 20, 40]. Recently, there exists several research on finding k-edge connected subgraphs, which concern about the local edge connectivity in the subgraphs [2, 6, 54]. Whereas, in this paper, we focus on the k-vertex connectivity of subgraphs.

Component detection The existing component detection methods can be roughly divided into clique-based [36] and clique-relaxed-based methods. Since the definition of clique is too strict, the clique-relaxed based methods have recently drawn a great deal of attentions. It can be classified into the following three categories: (1) Distance-based relaxed methods. *n-clique* is defined a maximal subgraph such that the distance of each pair of its vertices is not larger than n in the whole network [25]. *n-clan* is an n-clique whose diameter is no larger than n [33]. *n-club* is a maximal subgraph whose diameter is no larger than n [33]. (2) Degree-based relaxed methods. *k-plex* is defined as a maximal subgraph in which each vertex is adjacent to all other vertices except at most k of them. Recently, efficient approaches are proposed to enumerate k-plexes in large networks [4, 13]. Similarly, *k-core* is a maximal subgraph in which each vertex is adjacent to at least k other vertices of the subgraph. Efficient global search methods [9, 39] and local search method [14] have been developed to discover the k-core communities or the community k-core containing given entities. *Quasi-clique* with a parameter γ is a subgraph with n vertices and $\binom{\gamma * n}{2}$ edges [49]. (3) Triangulation-based relaxed methods. The definition of DN-graph is proposed in [43], which is a connected subgraph in which the lower bound of shared neighborhood between any connected vertices is locally maximized. k-truss is first proposed by J Cohen [11], and defined as the largest subgraph in which every edge is contained in at least $(k-2)$ triangles within the subgraph. It has received extensive research attentions [22, 23, 42]. Different from aforementioned models, in this paper, we propose the k-VCC model, which possesses high vertex connectivity.

Overlapping community detection Overlapping community structure exists in most real networks, which means vertices could belong to different communities [37]. Moreover, many methods have been proposed to solve the OCD problem. The clique percolation based method [1] defines a community as a k-clique component to detect the overlapping communities. The link-partition based method [31] first transforms the original graph G into line graph $L(G)$, in which the edges in G correspond to vertices in $L(G)$. Then, the node partition of $L(G)$ can be converted to overlapping vertex communities of G. The label

propagation based method [19] utilizes propagating labels between neighbouring vertices to detect communities that overlap. The k-VCC cannot only model community structure but is more aware of the connectivity of the communities. Furthermore, it naturally quantify the overlapping extent between two communities, such that two k-VCCs could at most overlap on $k-1$ vertices.

The differences of this extended paper from our conference version [28] are as follows:

- We give more analysis on complexity and property of this problem and present the paper with more clarity.
- We propose the hybrid framework to discover exact k-VCCs by contracting groups of obtained k-vertex connected subgraphs in a top-down manner. An upper bound of vertex connectivity and other optimization techniques are designed to accelerate the efficiency of this framework. Section 6 is newly added.
- We conduct more experiments on both real and synthetic datasets to evaluate the three proposed frameworks. Besides, we add another case study on protein interaction network to further determine the effectiveness of k-VCCs.

3 Notions and problem statement

In this paper, we focus on an *unweighted undirected* graph $G(V, E)$, where $V(G)$ is the set of vertices and $E(G)$ is the set of edges. The number of vertices, $|V| = n$, and the number of edges, $|E| = m$. We use $nb_G(v)$ to denote the set of neighbors of a vertex v in G, i.e., $nb_G(v) = \{u|(u, v) \in E\}$. The degree of a vertex $v \in V(G)$, denoted by $deg_G(v) = |nb_G(v)|$ is the number of neighbors of v in G. d_{max} denotes the maximum vertex degree of G. If there is no ambiguity, we denote them as $nb(v)$ and $deg(v)$. Paths joining the same pair of vertices are called *vertex disjoint* paths if they have no other vertices in common. A graph G' is a subgraph of G, denoted by $G' \subseteq G$ if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. Given a set of vertices $S \subseteq V$, the *induced* subgraph of S, denoted by $G[S]$, is a subgraph of G such that $G[S] = (S, E(S))$ where $E(S) = \{(u, v) \in E(G)|u, v \in S\}$.

Definition 1 (Vertex connectivity of two vertices) Let u and v be two vertices in graph G. If $(u, v) \notin E$, we define the vertex connectivity between u and v, denoted by $\kappa(u, v)$, as the least number of vertices chosen from $V - \{u, v\}$, whose deletion from G will disconnect u and v (destroy every vertex disjoint path between u and v), and for $(u, v) \in E$, $\kappa(u, v) = n - 1$, because they can only be separated by deleting $n - 1$ vertices.

Next, we define the connectivity of a graph.

Definition 2 (Vertex connectivity of a graph) The vertex connectivity of a graph G denoted as $\kappa(G)$ is the least cardinality $|S|$ of a vertex set $S \subseteq V$ such that $G[V \setminus S]$ is either disconnected or trivial (graph with only one vertex). Such a vertex set S is called a minimum vertex cut set.

Obviously, $\kappa(G)$ can be expressed in terms of $\kappa(v, w)$ as follows: $\kappa(G) = \min\{\kappa(v, w)| \textit{unordered pair of vertices } v, w \textit{ in } V\}$.

Definition 3 (k-vertex connected graph) A graph $G(V, E)$ is k-vertex connected if the remaining graph is still connected after the removal of any $(k - 1)$ vertices from G, i.e., $\kappa(G) \geq k$.

Specially, we define the graph with only one vertex is trivial and the vertex connectivity of a complete graph K_n is $(n-1)$. In other words, if a graph G is k-vertex connected, there are at least $(k+1)$ vertices in it.

Definition 4 (k-vertex connected component) Given a graph $G(V, E)$, a subgraph $G[S]$ ($S \subseteq V$) of G is a k-vertex connected component (k-VCC) if (1) $G[S]$ is k-vertex connected, and (2) any supergraph $G[S']$ ($S \subset S' \subseteq V$) is not k-vertex connected.

For example, in Figure 2, graph G_1, G_2, G_3 and G_4 are all 3-vertex connected subgraphs, while only G_3 and G_4 are 3-VCCs, because G_1 and G_2 are contained in G_4.

Problem statement Here, we give the formal problem statement.

Problem 1 (k-VCC detection problem) Given a graph $G(V, E)$ and an integer k, we study the problem of discovering all k-VCCs of G.

In theory, the value of k in k-VCC ranges from 1 to $n-1$, however, it is unlikely to reach $n-1$ in practice, because if k is large enough, the final result of k-VCCs is probably an empty set. Here, we give the upper bound of parameter k. It is highly related with k-core, denoted as $G[C_k]$, which is a maximal induced subgraph of G, such that $\forall v \in C_k$, $deg_{G[C_k]}(v) \geq k$. The core number of a vertex $v \in V$, denoted as $\psi(v)$, is the largest k such that v is in $G[C_k]$. In other words, $\psi(v) = k$ means that $v \in C_k$ and $v \notin C_{k+1}$.

Lemma 1 *All the k-VCCs in graph G are included in the k-core subgraph of G.*

Definition 5 (Degeneracy of G) The degeneracy $\mathcal{D}$ of $G(V, E)$ is the largest k for which G has a non-empty k-core, i.e., $\mathcal{D} = \max_{v \in V} \psi(v)$.

Theorem 1 *The value of k in k-VCC is upper bounded by the degeneracy $\mathcal{D}$ of G.*

Theorem 1 could be directly induced from Lemma 1. Next, we analyze the complexity of the k-VCC detection problem shown in Theorem 2.

Theorem 2 *The k-VCC detection problem can be solved in $O((n-k) \cdot (n-\delta-1+\delta(\delta-1)/2) \cdot mn^{2/3})$ polynomial time complexity, where $\delta = d_{max}$.*

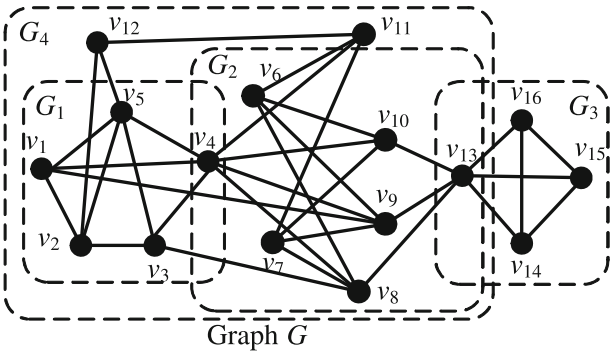

Figure 2 An example of k-VCCs

Proof As detailed in Section 4, we can compute the vertex cut V_{cut} of graph G in $O((n - \delta - 1 + \delta(\delta - 1)/2) \cdot mn^{2/3})$, where $\delta = d_{max}$ denoting the maximum vertex degree of G. If $|V_{cut}| < k$, in the extreme case, graph G could be divided into two subgraphs, i.e., g_1 and g_2 with size $|V_{cut}| + 1$ and $n - 1$, respectively. g_1 and g_2 overlaps on these V_{cut} vertices. Then, the larger subgraph g_2 can be further divided into two subgraphs. In the following separations, we can see that the size of the larger subgraph at least decreases 1 in each iteration. Because the size of k-VCC is larger than k, the separating process conducts at most $n - k$ times and it will stop when the size of the larger subgraph is k. Therefore, the k-VCC detection problem can be solved in $O((n - k) \cdot (n - \delta - 1 + \delta(\delta - 1)/2) \cdot mn^{2/3})$ time complexity, which is polynomial. □

Discussion Next, we discuss why k-VCC is a nice definition for components. We illustrate the reasons from the following three perspectives.

(1) *Robustness*. Based on the definition of k-VCC, the components can only be disconnected by removing at least k vertices, which indicates k-VCC possesses high vertex connectivity and it is uneasy to destruct the structure (connectivity) of k-VCCs. Oppositely, like k-core or k-ECC, even if they inhere high vertex degree or edge connectivity, only deleting one vertex will disconnect the components. Therefore, high vertex connectivity contributes more to the robustness of components.

(2) *Cohesiveness*. k-VCCs allow overlap between components. This is very natural and reasonable for component detection in large real-world networks, because a vertex can participate in a variety of components composed of different types of relationships. However, k-core and k-ECC only discover disjoint components (i.e., a vertex can only be contained in one component) and ignore the relationships that a component represents. This will bring in irrelevant subgraphs, called free rider effect [45]. On the contrary, due to allowing overlapping, k-VCCs could effectively prevent the free rider effect. Furthermore, the diameter of a k-VCC, $G[V_k]$, denoted as $diam(G[V_k])$, is upper bounded by $\lfloor(|V_k| - 2)/\kappa(G[V_k])\rfloor + 1$ [24]. Thus, when the connectivity of $G[V_k]$ is high, it has a low diameter, which also verifies the cohesiveness of k-VCCs.

(3) *Diversity*. k-VCC could guarantee the diversity of components. Although k-VCCs allow overlap, any two k-VCCs can only overlap on less than k vertices; otherwise they can be merge into a new larger one. Thus, k-VCCs could both discover diversified overlapping components and decrease the redundancy among k-VCCs.

4 Top-down framework for *k*-VCCs detection

In this section, we detail the top-down framework for k-VCCs detection. The main idea behind this framework is to iteratively compute the minimum vertex cut set V_{cut} of the current subgraph $G[C]$, if $|V_{cut}| \geq k$, then $G[C]$ is a k-VCC; otherwise V_{cut} and their incident edges are copied to each remaining connected subgraph after deleting V_{cut} and their induced edges from $G[C]$ and these newly constructed subgraphs are saved in a queue structure Q for further consideration.

Algorithm 1 summarizes this process. First, Lemma 1 enable us to exploit k-core to shrink the scale of G, which is possible to divide the original big graph G into several subgraphs of much smaller scale (line 1). In this way, the computation cost of the minimum vertex cut set V_{cut} of G is largely reduced. Now, the important problem is how to find the minimum vertex cut set (line 9). Unlike the min-cut [40] and Gomory–Hu tree [20] methods,

which can efficiently find the minimum edge cut set in an undirected graph, the problem of computing $\kappa(G)$ have to be reduced into a maximum flow problem in directed graph. Even and Tarjan [17] prove that $\kappa(s,t)$ in an undirected graph G is equivalent to the maximal value of flow from s to t in the corresponding constructed directed graph G'. And, $\kappa(G)$ can be calculated in $O(n-\delta-1+\delta(\delta-1)/2)$ calls to maximum flow algorithm where $\delta = d_{max}$ [16]. The detail of constructing the directed graph can be found in [28].

Algorithm 1 Top-down framework.

```
   Input: G(V, E), k
   Output: The set of k-VCCs G[R]
1  Find the k-core G[V_k] of G;
2  Initialize queue Q ← ∅;
3  Q.enqueue(all the connected subgraph of G[V_k]);
4  while Q ≠ ∅ do
5      G[C] ← Q.dequeue();
6      if |G[C]| > k then
7          Find the minimum vertex cut set V_cut of G[C];
8          if |V_cut| < k then
9              Find all the connected subgraphs of G[C \ V_cut], denoted as G[C_i];
10             Add V_cut and the induced edges into each G[C_i];
11             Q.enqueue(all G[C_i]);
12         else
13             Put G[C] into G[R];
14 Return G[R];
```

Actually, we do not need to find the minimum vertex cut set every time. For the current subgraph $G[C]$, once we discover a $\kappa(u,v) < k$ where $u, v \in G[C]$, $G[C]$ can not be a k-VCC, we can safely terminate this process and use the vertex cut set corresponding to u, v to separate $G[C]$. Theorem 3 guarantees the top-down framework is correct.

Theorem 3 *Given a graph G and a value k, the top-down framework for k-VCC detection can correctly find all the k-VCCs.*

Complexity analysis Computing $\kappa(v,w)$ on graph $G[C]$ needs $O(m'n'^{2/3})$ time where n' is the average size of C and m' is the average size of $E(G[C])$. In the worst case, it needs to be invoked $O(n'-\delta-1+\delta(\delta-1)/2)$ times. Let L represent the total number of $G[C]$ detected in the algorithm. The overall running time of the top-down framework is $O((n'-\delta-1+\delta(\delta-1)/2)\cdot m'n'^{2/3}\cdot L)$.

5 Bottom-up framework for *k*-VCCs detection

In this section, we develop a bottom-up framework for k-VCCs detection. Instead of depending on the global structure of graph, it focuses on the microscopic structure when dealing with large networks and thus is able to find target components with computational cost proportional to their size. The idea is to locally find seed subgraphs around the neighborhood of vertices and obtain the k-VCCs heuristically by expanding and merging these subgraphs. The overall framework is summarized in Algorithm 2. Although this framework is heuristic, the experiment in Section 7.3 shows that the result is comparable to the real.

Algorithm 2 Bottom-up framework for k-VCC detection.

Input: $G(V, E)$, k
Output: The set of k-VCCs, $\mathcal{G}[\mathcal{R}]$
1 $\mathcal{G}[\mathcal{R}] \leftarrow \emptyset$; $\mathcal{G}[\mathcal{S}] \leftarrow \emptyset$; $\mathcal{G}[\mathcal{S}'] \leftarrow \emptyset$;
2 Find the k-core $G[V_k]$ of G;
3 $\mathcal{G}[\mathcal{S}] \leftarrow$Seeding($G[V_k]$, k); //detailed in Section 5.1;
4 **while** $\mathcal{G}[\mathcal{S}'] \neq \mathcal{G}[\mathcal{S}]$ **do**
5 $\quad \mathcal{G}[\mathcal{S}'] \leftarrow \mathcal{G}[\mathcal{S}]$;
6 $\quad \mathcal{G}[\mathcal{S}] \leftarrow$Expanding($G[V_k]$, k, $\mathcal{G}[\mathcal{S}]$); //detailed in Section 5.2;
7 $\quad \mathcal{G}[\mathcal{S}] \leftarrow$Merging($G[V_k]$, k, $\mathcal{G}[\mathcal{S}]$); //detailed in Section 5.2;
8 $\mathcal{G}[\mathcal{R}] \leftarrow \mathcal{G}[\mathcal{R}] \cup \mathcal{G}[\mathcal{S}]$;
9 **return** $\mathcal{G}[\mathcal{R}]$;

5.1 Identifying seed subgraphs

We propose the local k-vertex connected subgraph as seed subgraph, which only considers the neighborhood structure of a vertex. Moreover, unlike maximal clique, which is adopted as seed subgraph in [27], the local k-vertex connected subgraph is more generalized as seed than clique, whose structure is too strict [36].

In this section, Seeding() is developed to identify such subgraphs. And there exists two important problems: (1) how to find the seed subgraph for a start vertex; (2) how to efficiently identify seed subgraphs for the whole network. Correspondingly, we first give the formal definition of seed subgraph and propose the LkVCS method for its discovery, and then we devise two optimization strategies to elaborately select the start vertices in which we do not have to identify seed subgraphs for all the vertices, and thus to accelerate the process of identifying seed subgraphs.

Identifying seed subgraph for a start vertex Here, we call the vertices used to obtain the seed subgraphs as start vertices. Assume u is a known start vertex, we study how to define and find the seed subgraph for it. We first give the definition of 2-ego neighborhood. We then define the local k-vertex connected subgraph as seed subgraph by exploiting 2-ego neighborhood to add additional constraint on the path length between vertices in the defined local k-vertex connected subgraph. This makes it possible to find the seed subgraph within the neighborhood of the start vertex.

Definition 6 (2-ego neighborhood) Given a graph $G(V, E)$ and a vertex u in G, the 2-ego neighborhood of u, $N_2(u)$, denotes the set of vertices in G whose distance to u is no more than 2, i.e., $\{v|dist(u, v) \leq 2\}$. Specially, u also belongs to $N_2(u)$.

Definition 7 (Local k-vertex connected subgraph) Given a graph $G(V, E)$ and a start vertex $u \in V$, an induced subgraph $G[S]$ is a local k-vertex connected subgraph if and only if (1) $\forall v, w \in S$, if $(v, w) \notin E(S)$, $|nb_{G[S]}(v) \cap nb_{G[S]}(w)| \geq k$; (2) $|S| > k$; (3) $u \in S$.

From Definition 7, we can see the diameter of the local k-vertex connected subgraph is no more than 2 proved in Theorem 4. And the diameter of a graph is defined as the longest distance among all the distances between any pair of vertices in G, i.e., $diam(G) = \max_{u,v \in G}\{dist_G(u, v)\}$.

Theorem 4 *The local k-vertex connected subgraph $G[S]$ is k-vertex connected and its diameter $diam(G[S]) \leq 2$.*

Proof In Definition 7, condition (1) guarantees there are at least k independent paths and the path length is 2 for each pair of vertices that are not directly connected. Also, based on Definition 1, the connectivity between the pair of vertices that are connected in the network is $|S| - 1$ and the path length is 1. So, the path length between any pair of vertices in $G[S]$ is no more than 2. In addition, condition (2) makes sure that $G[S]$ has enough vertices to be a k-vertex connected subgraph. □

Based on [14], in real networks, community is usually existed in the neighborhood of vertices, hence it is rational to define the local k-vertex connected subgraph with path length constraint as seed subgraph. Furthermore, a k-VCC is usually composed of many adjacent local k-vertex connected subgraphs. Thus, if we can obtain these local subgraphs in advance, we probably retrieve the original k-VCC from them.

Algorithm 3 LkVCS.

Input: $G(V, E)$, k, u
Output: Local k-vertex connected subgraph $G[R]$ of u

$P \leftarrow N_2(u)$;
$P' \leftarrow$ the vertex set of k-core of $G[P]$;
if $u \notin P'$ **then**
  return $\emptyset$;
for *each subset* R *of* $nb_{G[P']}(u)$ *with size* k **do**
  $R \leftarrow R \cup u$;
  boolean $direct \leftarrow ture$;
  while $direct == true$ **do**
    if $\forall x, y \in R, (x, y) \notin E$ *and* $|nb_{G[R]}(x) \cap nb_{G[R]}(y)| \geq k$ **then**
      return $G[R]$;
    else
      if $\exists x, y \in R, (x, y) \notin E$ *and* $|nb_{G[P']}(x) \cap nb_{G[P']}(y)| < k$ **then**
        $direct \leftarrow false$;
      else
        Find a pair of vertices $x, y \in R$ such that $(x, y) \notin E$ and $|nb_{G[R]}(x) \cap nb_{G[R]}(y)| < k$, and randomly choose $k - |nb_{G[R]}(x) \cap nb_{G[R]}(y)|$ vertices from $(nb_{G[P']}(x) \cap nb_{G[P']}(y)) \setminus (nb_{G[R]}(x) \cap nb_{G[R]}(y))$ adding into R;
return $\emptyset$;

Further, we propose the LkVCS algorithm shown in Algorithm 3, which finds one of the local k-vertex connected subgraphs from the induced subgraph $G[N_2(u)]$ as the seed subgraph for a start vertex u. The LkVCS adopts the idea of locally enumerating and starts with every different subset of vertices of size k from the neighborhood of u, denoted as R and then continue bring vertices from $P' \setminus R$ into R until $G[R]$ is a local k-vertex connected subgraph or there exists $x, y \in R$ and $(x, y) \notin E$ such that $|nb_{G[P']}(x) \cap nb_{G[P']}(y)| < k$ where P' is the vertex set of k-core of $G[N_2(u)]$. When the combination number $\binom{|nb_{G[P']}(u)|}{k}$ is larger than a threshold α, α subsets are randomly tested. The pseudo code of the LkVCS algorithm is in the supplementary materials. Next, Example 1 illustrates how the LkVCS algorithm works.

Example 1 We apply LkVCS on the graph G in Figure 2 for $u = v_1$. We set $k = 3$. First, we obtain the 3-core of $G[N_2(v_1)]$ is G_4. We arbitrarily select 3 vertices from $nb_{G_4}(v_1)$, that is $\{v_2, v_3, v_4\}$ and $R = \{v_2, v_3, v_4\} \cup \{v_1\}$. Because $G[R]$ is not a 3-vertex connected subgraph, we continue to add the common neighbor v_5 of v_1, v_3 and v_2, v_4 into R. Now,

$G[R]$ is a 3-vertex connected subgraph. We output $G[R]$ as the local 3-vertex connected subgraph of v_1.

Identifying seed subgraphs for the whole network A naive way is to view every vertex in the network as start vertex and compute the corresponding local k-vertex connected subgraph for them by LkVCS algorithm. However, it is very inefficient. In this part, we carefully devise two optimization strategies to select some of the vertices as start vertices to identify seed subgraphs, and thus further reduce the computational cost largely.

Optimization 1: Vertex order priority based strategy. In vertex order priority strategy, we assign the vertices with smaller degree have higher priority than that with larger degree. We observe that the value of $\binom{deg(u)}{k}$ is not large for a vertex u with small degree. Hence, if a vertex u having smaller degree, we can take less time to detect whether there exists a local k-vertex connected subgraph for u.

Theorem 5 *Given a vertex u in $G(V, E)$, it can be contained in at most 0 k-VCCs when $deg(u) < k$ or $1 + \lceil \frac{deg(u)-k}{k-1} \rceil$ k-VCCs when $deg(u) \geq k$, simultaneously.*

Proof Based on Definition 3, if the vertex u is in a k-vertex connected subgraph, it must have at least k neighbors. Thus, for u with $deg(u) < k$, it cannot be contained in any k-vertex connected subgraphs. Next, recall the property of the k-VCCs that given any two k-VCCs G_i and G_j, $|V_i \cap V_j| < k$. Based on this property and Definition 3, for the vertex u with $deg(u) \geq k$, besides u, there are at least k neighbors of u belonging to the same k-vertex connected subgraph. At the same time, any two k-vertex connected subgraph including u can overlap at most $k - 1$ vertices. Thus, u can be contained in at most $1 + \lceil \frac{deg(u)-k}{k-1} \rceil$ k-VCCs. □

Theorem 5 gives the upper bound of number of k-VCCs, that a vertex could be contained in at the same time. We can see that the vertices with larger degree can be contained in more different k-VCCs and even larger amount of k-vertex connected subgraphs. In particular, for a specific vertex u with $deg(u) = k$, it can only be contained in at most 1 k-VCC. Recall that the LkVCS method will take more computational time for the input vertex u with larger vertex degree, because it will enumerate much more combinations of vertices than that of small degree to find the local k-vertex connected subgraphs. To avoid visiting the vertices with larger degree first, we set the vertices with larger degree having lower priority.

Optimization 2: Non-redundancy based strategy. We observe that if we find the seed subgraph for every vertex in the network, we will acquire many duplicate subgraphs or highly overlapping subgraphs. In order to reduce redundance, we design the non-redundancy based strategy such that for a given vertex, if it has already been contained in the discovered seed subgraphs of other vertices, there is no need to find its own seed subgraph. Note that a vertex can be included in different seed subgraphs, even if it is not processed.

Together with the vertex order priority strategy, we can find seed subgraphs for the uncovered vertices with smaller degree as soon as possible. Besides, based on the long-tail theory, the vertices with larger vertex degree are probably included in the discovered seed subgraphs, which means we do not need to detect seed subgraphs for these vertices. Thus, making use of these two strategies, we can find constraint number of seed subgraphs with higher efficiency.

In Seeding() shown in Algorithm 4, we visit all the vertices in the graph according to the non-decreasing order of their vertex degree. If a vertex v has not been visited, we find its local k-vertex connected subgraph using the LkVCS algorithm as its seed subgraph; otherwise we will continue. Finally, we obtain the set of seed subgraphs.

Algorithm 4 Seeding.

```
   Input: G(V, E), k
   Output: The set of seed subgraphs 𝒢[𝒮]
1  𝒢[𝒮] ← ∅;
2  foreach v ∈ V do
3      CandMaintain[v] ← 0;
4  foreach v ∈ V in non-decreasing order w.r.t. deg(v) do
5      if CandMaintain[v] == 0 then
6          G[C] ← LkVCS(G, k, v);
7          if C ≠ ∅ then
8              𝒢[𝒮] ← 𝒢[𝒮] ∪ G[C];
9              foreach u ∈ C do
10                 CandMaintain[u] ← 1;
11 return 𝒢[𝒮];
```

Example 2 We apply the above two strategies in seeding() on the graph shown in Figure 2 to identify the seed subgraphs. We rank all the vertices in G according to their non-decreasing order of their vertex degree denoted as, $v_{12}(3) \preceq v_{14}(3) \preceq \ldots \preceq v_9(5) \preceq v_{13}(6) \preceq v_4(7)$. The numbers in the brackets are their vertex degree. We first find the seed subgraph for v_{12}, which is $\emptyset$. Then, we successively visit the vertex according to the vertex priority. For example, we find the seed subgraph G_3 for v_{14}. As G_3 contains $\{v_{13}, v_{15}, v_{16}\}$, we do not need to detect the seed subgraphs for these vertices. At last, we identify three seed subgraphs including G_3 for v_{14}, G_1 for v_1 and G_2 for v_6.

5.2 Expanding and merging seed subgraphs

In this section, we focus on solving the problem of detecting k-VCCs from the discovered seed subgraphs. We observe that the k-VCCs do not satisfy the property of *downward closeness*. That is for a k-VCC denoted as $G[S]$, $\exists S' \subseteq S$, the induced subgraph $G[S']$ is not a k-vertex connected subgraph. Thus, we cannot simply expand the discovered seed subgraphs by adding a series of vertices adjacent to their neighborhood to obtain the target k-VCCs.

Menger's Theorem [15] indicates that a graph is k-vertex connected if and only if it contains k vertex independent paths between any two non-adjacent vertices. Based on this relationship between independent path and vertex connectivity, we devise two algorithmic approaches, Expanding() and Merging(), in which, we could safely add vertices into the current subgraph and combine different subgraphs to form a bigger k-vertex connected subgraph, respectively. Next, we detail these two approaches.

Expanding We first give some explanations of the notions used here. Given a graph $G(V, E)$ and a vertex set $S \subseteq V$, $\overline{S}$ represents the complement of S in G, i.e., $\overline{S} = V \setminus S$, and δS denotes the *boundary* of the induced subgraph $G[S]$, which means for any vertex $v \in \delta S$, there exists a vertex $u \in nb(v)$ and $u \notin S$.

In Expanding() shown in Algorithm 5, we add vertices that are connected to at least k vertices in $G[S]$ into the current subgraph. The specific process is as follows. For each k-vertex connected subgraph $G[S]$ now we have, if there exists vertex u in $\delta\overline{S}$ that is adjacent to at least k vertices in δS, we add every vertex like this into the current S and update $S \cup u$ as the new S. We iteratively conduct this procedure until there is no such vertex in $\delta\overline{S}$ that can be added into S. Theorem 6 guarantees the correctness of the Expanding() approach.

Algorithm 5 Expanding.

Input: $G(V, E)$, k, the set of k-vertex connected subgraphs $\mathcal{G}[\mathcal{S}]$
Output: The set of updated k-vertex connected subgraphs $\mathcal{G}[\mathcal{S}]$
1 **foreach** $G[S] \in \mathcal{G}[\mathcal{S}]$ **do**
2 **while** $\exists u \in \delta\overline{S}$, $|nb(u) \cap \delta S| \geq k$ **do**
3 Update $G[S] \leftarrow G[S \cup u]$;
4 **return** $\mathcal{G}[\mathcal{S}]$;

Theorem 6 *Suppose $G[S]$ is a k-vertex connected subgraph. If vertex u is adjacent to at least k vertices in δS, $G[S \cup u]$ is also a k-vertex connected subgraph.*

Proof We need to prove that for every pair of vertices a, b in $G[S \cup u]$, $\kappa(a, b) \geq k$. Since $G[S]$ is a k-vertex connected subgraph, we only need to prove for $\forall a \in S$, $\kappa(u, a) \geq k$. We assume that there is a vertex $c \in S$ making $\kappa(u, c) < k$. Based on the Menger's Theorem, we know that there are k' ($k' < k$) independent paths between u and c. So, once we delete one vertex from each of the k' paths, u will be disconnected from c. However, u is adjacent to at least k vertices in δS, even if we delete any k' vertices from them, the subgraph is still connected, which is controversial to the assumption. Thus, $G[S \cup u]$ is a k-vertex connected subgraph. □

Example 3 We use the graph in Figure 2 as an example to illustrate Expanding(). Assume that we have already obtained the 3-vertex connected subgraph $G[S]$ where $G[S] = G_2$. We find that v_{11} is adjacent to three boundary vertices including v_4, v_6 and v_7 in $G[S]$. Thus, we add v_{11} into $G[S]$ and $G[S \cup v_{11}]$ is also a 3-vertex connected subgraph.

Merging When the obtained k-vertex connected subgraphs cannot be further expanded by Expanding(), we expect to combine some of the adjacent subgraphs together to acquire larger subgraphs with k-vertex connectivity. Here, we develop Merging() shown in Algorithm 6 to integrate different k-vertex connected subgraphs with at least k direct independent paths into a new one. Specifically, we first detect whether the input subgraphs are k-VCCs. If so, we directly output them as k-VCCs; otherwise, we iteratively merge the subgraphs satisfying the condition in Theorem 7 that will be detailed later until no subgraphs can be merged. If meeting the conditions in Corollary 1, the subgraph after combined is already a k-VCC, otherwise it will be saved for further processing. In the implementation, we use the disjoint sets structure to accelerate the merging operation.

Formally, Theorem 7 provides the sufficient condition to guarantee Merging() is correct. If the sum of the number of overlapping vertices and length-1 disjoint paths between two k-vertex connected subgraphs is more than or equal to k, they can be combined together.

Algorithm 6 Merging.

Input: $G(V, E)$, k, the set of k-vertex connected subgraphs $\mathcal{G}[\mathcal{S}]$
Output: The set of updated k-vertex connected subgraphs $\mathcal{G}[\mathcal{S}]$

```
foreach G[S] ∈ G[S] do
    if G[S] satisfies the conditions in Corollary 1 then
        Put G[S] into G[R];
        Delete G[S] from G[S];
while ∃G[S], G[S'] ∈ G[S] satisfies the condition in Theorem 7 do
    if G[S ∪ S'] satisfies the conditions in Corollary 1 then
        Put G[S ∪ S'] into G[S];
        Delete G[S] and G[S'] from G[S];
    else
        Put G[S ∪ S'] into G[S];
        Delete G[S] and G[S'] from G[S];
return G[S];
```

Theorem 7 *Assume that S and S' are sets of vertices in a graph G, i.e., $S \subseteq V$, $S' \subseteq V$, and the induced subgraph $G[S]$ and $G[S']$ are k-vertex connected. We say $G[S \cup S']$ is a k-vertex connected subgraph, if the following condition is satisfied, $|S \cap S'| + \min\{|nb_{S' \setminus S}(S \setminus S')|, |nb_{S \setminus S'}(S' \setminus S)|\} \geq k$.*

Proof Based on Definition 2, graph $G[S \cup S']$ is k-vertex connected if the vertex connectivity between any pair of vertices r, r' in $G[S \cup S']$ are not smaller than k. Since $G[S]$ and $G[S']$ have already been k-vertex connected, we only need to prove that $\forall s \in S \setminus S'$, $\forall s' \in S' \setminus S$, $\kappa(s, s') \geq k$. A S-S' path is a path from some $s \in S$ to some $s' \in S'$ that passes through no other vertex of S or S'. If a vertex x belongs to both S and S', then x itself can be viewed as a S-S' path. According to Menger's Theorem, if the number of disjoint S-S' pathes is not smaller than k, S and S' are k-vertex connected. Thus, based on the conditions, if the sum of the number of sharing vertices and length 1 disjoint paths is not smaller than k, $G[S \cup S']$ is k-vertex connected. □

Furthermore, based on Theorem 6 and Theorem 7, we obtain Corollary 1. In Merging(), Corollary 1 can be exploited as an *early termination condition*. That is once finding some subgraphs which satisfy the conditions in Corollary 1, we can put these subgraphs into the result set, because it is impossible to be combined with any other subgraphs. This can significantly reduce the number of subgraph combining operations.

Corollary 1 *Let $G[S]$ be a k-vertex connected subgraph. If the following two conditions are satisfied, we say $G[S]$ is a k-VCC:*

(1) $\nexists v \in \delta(\overline{S})$, $|nb(v) \cap \delta S| > k$;
(2) $\min\{|\delta S|, |\delta \overline{S}|\} < k$.

Example 4 A running example of Merging() is given using G in Figure 2. Suppose we already have three 3-vertex connected subgraphs G_1, G_2 and G_3. We observe that G_3 satisfies the conditions in Corollary 1. That is, v_8, v_9 and v_{10} are only adjacent to one boundary vertex v_{13} of G_3, and $\min\{|\delta V(G_3)|, |\delta \overline{V(G_3)}|\} = \min\{1, 3\} = 1 < 3$. Thus, G_3 is a 3-VCC. For G_1 and G_2, we have that $|V(G_1) \cap V(G_2)| + \min\{|\delta V(G_1) \cap \delta \overline{V(G_2)}|, |\delta \overline{V(G_1)} \cap$

$\delta V(G_2)|\} = 1 + \min\{2, 2\} = 3 \geq 3$. Thus, we can merge G_1 and G_2 together based on Theorem 7.

Time complexity: First, in the seed subgraph identification step, the Seeding() invokes n times of the LkVCS algorithm at the worst case, whose time complexity is $O(\alpha|E_{avg}|)$ where $|E_{avg}|$ represents the average edge number in $G[N_2(u)]$. Thus, the time complexity for Seeding() is $O(n\alpha|E_{avg}|)$. Then, the time complexity of Expanding() is $O(N_s d_{avg}|B_{avg}|)$, where N_s represents the number of seed subgraphs, d_{avg} is the average degree in G and $|B_{avg}|$ represents the average number of vertices in the boundary of seed subgraphs. Also, the time complexity of Merging() is $O(N_s^2 d_{avg}|B_{avg}|)$. Suppose that Algorithm 2 executes at most N_{iter} iterations of expanding and merging, the overall time complexity of Algorithm 2 is $O(n\alpha|E_{avg}| + N_{iter}(N_s d_{avg}|B_{avg}| + (N_s^2 d_{avg}|B_{avg}|))$.

Space complexity: Algorithm 2 needs to record all the seed subgraphs and it takes $O(N_s|E_{avg}|)$ space.

6 Hybrid framework for *k*-VCCs detection

When identifying the target k-VCCs, the bottom-up framework only exploits the pairwise relationship between a k-vertex connected subgraph and a vertex (or a k-vertex connected subgraph) corresponding to Expanding()(or Merging()). To fix this limitation, in this section, we propose a hybrid framework for k-VCCs detection.

The hybrid framework focuses on the group relationships among the subgraphs and vertices, and thus is able to detect larger k-vertex connected subgraphs building on the results obtained from the bottom-up framework. For example, in Figure 3a, A, B, C and D (enclosed by the dashed circles) represent four 3-vertex connected subgraphs that are discovered by the bottom-up framework and E denotes a single vertex. Intuitively, we can find that each pair of them cannot be merged, however, the whole graph is a 3-VCC, if we consider the group $\{A, B, C, D, E\}$ together.

Algorithm 7 summarizes the hybrid framework. We first construct a novel *mixed graph* with the discovered k-vertex connected subgraphs and the vertices that are not assigned to any k-vertex connected subgraphs after the bottom-up framework (line 1), and then find larger k-vertex connected subgraphs by further iteratively contracting the mixed graph (line 2). In the end, we identify the target k-VCCs by a top-down way (line 3).

Algorithm 7 Hybrid framework for k-VCC detection.

Input: The results of Bottom-up framework $\mathcal{G}[\mathcal{S}]$, unassigned vertices $V_{unasign}$, integer k
Output: The set of updated k-vertex connected subgraph, $\mathcal{G}[\mathcal{R}]$
1 Construct the mixed graph G_{mix} with $\mathcal{G}[\mathcal{S}]$ and $V_{unasign}$; //detailed in Section 6.1;
2 Updated G_{mix} ←Contraction(G_{mix}); //detailed in Section 6.2;
3 Updated $\mathcal{G}[\mathcal{R}]$ ←Topdown_revise(G_{mix}); //detailed in Section 6.3;
4 **return** $\mathcal{G}[\mathcal{R}]$;

6.1 Constructing the mixed graph

We propose the *mixed graph* to facilitate the identification of the target k-VCCs. After processed by the bottom-up framework, the original graph is covered by the discovered k-vertex connected subgraphs and the vertices unassigned to any k-vertex connected subgraphs.

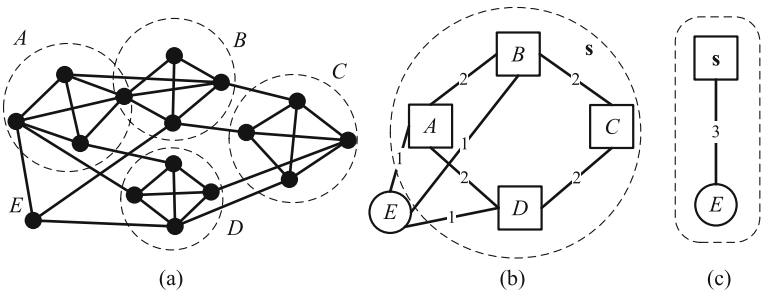

Figure 3 An example of the construction of the mixed graph for $k = 3$

Here, we also view the single vertex as a subgraph. In mixed graph, we model these subgraphs as vertices and the connectivity between them as weighted edges.

Specifically, we denote the discovered k-vertex connected subgraphs as type T_1 vertices and the subgraphs of single vertex as type T_2 vertices, respectively. To preserve the relationship lossless, according to Theorem 6 and 7, we use the number of adjacent vertices to measure the edge weight between vertices of type T_1 and T_2, while utilize the sum of overlapping vertices and length 1 independent paths to qualify the weights of edges connecting both two type T_1 vertices. Definition 8 shows the formal definition of mixed graph.

Definition 8 (Mixed graph) Let $G_{mix}(V_{mix}, E_{mix})$ represent the mixed graph, where V_{mix} is the set of vertices and E_{mix} is the set of edges. Each vertex $v_i^{t_i} \in V_{mix}$ corresponds to a subgraph in G, where $t_i \in \{T_1, T_2\}$ denotes the type of v_i. T_1 and T_2 represent v_i to be a k-vertex connected subgraph and a single vertex in G, respectively. For each edge $e = (v_i^{t_i}, v_j^{t_j}) \in E_{mix}$, its edge weight $w(e)$ measures the vertex connectivity between v_i and v_j and can be computed as follows,

$$w(v_i^{t_i}, v_j^{t_j}) = \begin{cases} |\mathcal{Z}| + \min\{|nb_{\mathcal{Y}}(\mathcal{X})|, |nb_{\mathcal{X}}(\mathcal{Y})|\} \\ \quad t_i = T_1, t_j = T_1, \\ |nb_{map(v_i)}(map(v_j))| \; t_i = T_1, t_j = T_2, \\ 1 \; t_i = T_2, t_j = T_2 \end{cases}$$

where $\mathcal{X} = map(v_i) \setminus map(v_j)$, $\mathcal{Y} = map(v_j) \setminus map(v_i)$, $\mathcal{Z} = map(v_i) \cap map(v_j)$ and the function $map(\cdot)$ maps the vertices to their original vertex sets in the original graph G.

Example 5 Figure 3 illustrates an example of the mixed graph when $k = 3$. Figure 3a is the original graph. A, B, C and D are four discovered 3-vertex connected subgraphs. E represents the unassigned vertex. Figure 3b shows the corresponding mixed graph, in which all these subgraphs/vertices in the original graph are treated as vertices and the values of connectivity between them are represented as weighted edges. In particular, we use '□' to represent the discovered k-vertex connected subgraphs (type T_1) and '◯' to mark the unassigned vertices (type T_2).

Upper bound of vertex connectivity Let G_{mix} represent a mixed graph and v belongs to G_{mix}, we derive the upper bound of vertex connectivity between the corresponding subgraph $G[map(v)]$ of v and the rest of graph G. We denote this upper bound as $\overline{\mathsf{vc}}(v)$, which equals to the sum of the weights of edges that are adjacent to v, i.e.,$\overline{\mathsf{vc}}(v) = \sum_{u \in nb_{G_{mix}}(v)} w(u, v)$. Next, we prove the correctness of this upper bound.

Theorem 8 $\overline{\mathsf{vc}}(v) \geq \kappa(G[map(v)], G[V \backslash map(v)])$, *where* $\kappa(G[map(v)], G[V \backslash map(v)])$ *represents the real vertex connectivity between subgraphs* $G[map(v)]$ *and* $G[V \backslash map(v)]$ *in* G.

Proof Because for different vertices such as $u, h \in nb_{G_{mix}}(v)$, the independent paths between $map(v)$ and $map(u)$ and between $map(v)$ and $map(h)$ may incident on the same vertices, the real number of independent paths between $map(v)$ and $map(u) \cup map(h)$ may be less than $w(v,u) + w(v,h)$. Similarly, $\kappa(G[map(v)], G[V \backslash map(v)])$ may be also less than $\overline{\mathsf{vc}}(v)$. And the equal sign is established when the independent paths between $map(v)$ to other subgraphs are all incident on different vertices. □

Based on Theorem 8, for a vertex v, if $\overline{\mathsf{vc}}(v) < k$, we can explicitly prune v from G_{mix} without exactly computing the vertex connectivity in G. This is because if v is a type T_1 vertex, $G[map(v)]$ is already a k-VCC, otherwise v is a type T_2 vertex, and it cannot be in any of the k-VCCs. $\overline{\mathsf{vc}}(v)$ is exploited as an *early deletion condition* in the following steps. In addition, $\overline{\mathsf{vc}}(v)$ becomes tighter along with the contraction of the mixed graph (detailed in Section 6.2). This is because when the size of G_{mix} becoming smaller, the number of neighbors for a vertex v may also get smaller, which decrease the repeated counting of independent paths. For example, in Figure 4, $\overline{\mathsf{vc}}(v)$ is originally equal to 3, and after S contracted to one vertex $\mathbf{u}$, $\overline{\mathsf{vc}}(v)$ becomes to 1, which is the same as the exact vertex connectivity between $G[map(v)]$ and $G[map(\mathbf{u})]$.

Implementation When the bottom-up framework is conducted, we can easily map the discovered k-vertex connected subgraph to its corresponding vertex in G_{mix}. Meanwhile, in Expanding() and Merging(), the number of direct independent paths between subgraphs or vertices needed to be computed when merging two subgraphs or adding one vertex into a subgraph, which is just corresponding to the weight between two vertices in G_{mix}. Therefore, the construction of G_{mix} can naturally carry out along with the conducting of bottom-up framework and do not bring in extra computation.

6.2 Contracting the mixed graph

Due to the probably large size of the mixed graph, in this section, we study how to contract the mixed graph. This will largely reduce the computational cost for finding out target k-VCCs in a top-down manner, which has to exploit the global structure of the mixed graph.

To shrink the scale of the mixed graph, we design an iterative mixed graph contraction method, called Contraction(). In each iteration, (1) we try to contract groups of T_1 vertices in the mixed graph by utilizing the efficient state-of-the-art k-ECC detection method KECC() [6]; (2) Due to the change of edge weight, we have to renew the graph after

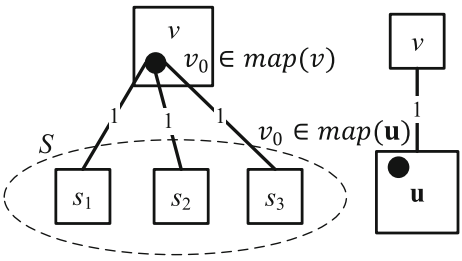

Figure 4 An example of renewing the mixed graph

contracting to be a mixed graph again. In addition, we can safely remove the vertex if its connectivity to other vertices are less than k. The whole process ceases when the mixed graph cannot be further updated. Next, we detail the above two procedures.

Contracting type T_1 vertices To further reduce the size of G_{mix}, we design to contracting some of the type T_1 vertices. Here, we observe that (1) each type T_1 vertex can only belong to one k-ECC in $G_{mix}[V_{mix}^{T_1}]$, which is detailed in Lemma 2; (2) only the vertices corresponding to groups of type T_1 vertices in the same k-ECC could make up larger k-vertex connected subgraph in G. This is verified in Lemma 3.

Lemma 2 *$\forall v \in V_{mix}^{T_1}$, v can only belong to one k-ECC in $G_{mix}[V_{mix}^{T_1}]$.*

Proof We prove this lemma by contradiction. Assume v belongs to different k-ECCs. Based on the definition of k-ECC, let $G_{mix}[S]$ be a k-ECC in $G_{mix}[V_{mix}^{T_1}]$ and after removing any k-1 edges from it, $G_{mix}[S]$ is still connected. Thus, all these k-ECCs containing v is k-edge connected, which violates the k-ECC is a maximal subgraph. Thus, v can only belong to one k-ECC in $G_{mix}[V_{mix}^{T_1}]$. □

Lemma 3 *Let $G_{mix}[S]$ represent a k-ECC in $G_{mix}[V_{mix}^{T_1}]$, only for $S' \subseteq S$, $G[\bigcup_{v \in S'} map(v)]$ could make up larger k-vertex connected subgraph in G.*

Proof We prove this lemma by contradiction. Assume v and v' belong to different k-ECCs in $G_{mix}[V_{mix}^{T_1}]$ and $map(v)$ and $map(v')$ are in the same k-vertex connected subgraph in G. Because of belonging to different k-ECCs in $G_{mix}[V_{mix}^{T_1}]$, v and v' are less than k-edge connected. Correspondingly, $map(v)$ and $map(v')$ are less than k-edge connected in G, and thus are less than k-vertex connected, which contradicts to the assumption. □

According to Lemma 2, we can first partition $G_{mix}[V_{mix}^{T_1}]$ into different k-ECCs. Here we adopt the KECC() method, which only takes $O(h \times l|E|)$ time [6], where h is the height of the decomposition tree and $l \ll |V|$. This is much faster than the top-down framework for k-VCC detection.

Then, based on Lemma 3, we want to find out the set of type T_1 vertices in the same k-ECC that can be further contracted to shrink G_{mix}. As we know, for every vertex v in the current k-ECC, i.e., $G_{mix}[S]$, even if its upper bound $\overline{\mathsf{vc}}(v) \geq k$, we still cannot determine whether its corresponding subgraph $G[map(v)]$ is k-vertex connected to the rest of the induced subgraph corresponding to S in G, denoted by $G[map(S) \setminus map(v)]$. Here, we calculate the exact k-vertex connectivity between $G[map(v)]$ and $G[map(S) \setminus map(v)]$. Moreover, we observe that if every $G[map(v)]$ is k-vertex connected to the rest of the subgraph, then $G[map(S)]$ is a larger k-vertex connected subgraph, which is proved in Theorem 9. Therefore, we can safely combine the type T_1 vertices in S into a larger one.

Theorem 9 *Assume $G_{mix}[S]$ denote a k-ECC in G_{mix}, $S \subseteq V_{mix}^{T_1}$ and $S' \subset S$, $G[map(S')]$ is a k-vertex connected subgraph iff $\forall v \in S'$, $G[map(v)]$ is k-vertex connected to $G[map(S') \setminus map(v)]$.*

Proof First, we prove the sufficiency. If $G[map(S')]$ is a k-vertex connected subgraph, then $\forall v, v' \in S'$, $\kappa(v, v') \geq k$. Thus, $\forall v \in S'$, $G[map(v)]$ is k-vertex connected to

$G[map(S') \setminus map(v)]$. Then, we prove the necessity. Because v is a type T_1 vertex, $G[map(v)]$ itself is a k-vertex connected subgraph, for every v, deleting any $k-1$ vertices within $G[map(v)]$ will not disconnect $G[map(S')]$. Also, $G[map(v)]$ is k-vertex connected to $G[map(S') \setminus map(v)]$ for every v in S'. Therefore, even if removing any $k-1$ vertices from different $map(v)$s, $G[map(S')]$ is still connected. Overall, $G[map(S')]$ is a k-vertex connected subgraph. □

In practice, we can iteratively remove v such that $\kappa(G[map(v)], G[map(S) \setminus map(v)]) < k$ from S until all the vertices in the current S satisfy Theorem 9. Finally, we contract all the vertices in the current S into a larger one without destroying the connectivity of the corresponding subgraph in G.

Renewing the mixed graph After contracting the type T_1 vertices, we have to update the mixed graph. Let $\mathcal{S}$ represent the set of type T_1 vertices that can be grouped together. After contracting $\mathcal{S}$ into a new type T_1 vertex (e.g. $\mathbf{u}$) in the mixed graph, we first need to duplicate all the vertex mapping information of S into $\mathbf{u}$. Then, we reconnect and reweigh the edge weight between the neighbors of $\mathcal{S}$ in G_{mix} (e.g., $nb_{G_{mix}}(\mathcal{S})$) and $\mathbf{u}$. For a vertex $v \in nb_{G_{mix}}(\mathcal{S})$, when computing $w(v, \mathbf{u})$, we cannot easily use $\sum_{s \in \mathcal{S}} w(v, s)$ as its edge weight, because not all the paths between v and $\mathcal{S}$ are independent. For example, in Figure 4, all the vertices in $\mathcal{S}$ are adjacent to v at $v_0 \in map(v)$, thus $w(v, \mathbf{u}) = 1$.

After renewing $\mathbf{u}$ and the weights of its incident edges, there may exist edges with weight no less than k. Due to the weights of all edges in the mixed graph are less than k, it is necessary to merge the two parts of the edge with weight larger or equal to k until there is no such cases by Expanding() and Merging(). Notably, once a new type T_1 vertex $\mathbf{u}$ is generated, it is necessary to utilize Expanding() to enlarge $map(\mathbf{u})$ in G. This will ensure the maximality of the final results. For example, in Figure 4, if $v_0 \notin map(\mathbf{u})$, we should add v_0 into $map(\mathbf{u})$. The following example illustrates the complete process of contracting the mixed graph.

Example 6 In Figure 3b, the subgraph induced by the type T_1 vertices including $\{A, B, C, D\}$ is a 3-ECC for $G_{mix}[V_{mix}^{T_1}]$ and each one of them is k-vertex connected to the rest. Hence, we can merge all these vertices together into a new type T_1 vertex, denoted as $\mathbf{s}$ and reconnect all the edges originally connected to $\{A, B, C, D\}$ to $\mathbf{s}$, which is shown in Figure 3c. Because all the paths are independent, $w(E, \mathbf{s}) = 3$. Finally, we continue to merge E and $\mathbf{s}$ as a whole 3-VCC, due to $w(E, \mathbf{s}) \geq 3$.

6.3 Identifying *k*-VCCs based on the mixed graph

In this section, we propose the Topdown_revise() method to find the target k-VCCs from the contracted G_{mix}. The main idea is to iteratively divide the mixed graph in a top-down manner until the separated parts themselves are corresponding to the k-VCCs in the original graph. Because the size of G_{mix} is much smaller than G, the Topdown_revise() is much more efficient than the top-down framework, which directly divides the original graph G.

First, we study how to determine the minimum vertex cut of G_{mix}. Recall that the top-down framework (detailed in Section 4) detects k-VCCs by iteratively computing $\kappa(G[C])$ for the current graph $G[C]$ and reduce it to a maximum flow problem in directed graph. Correspondingly, Topdown_revise() identifies k-VCCs by computing $\kappa(G_{mix}[C])$ for the current $G_{mix}[C]$. Here, we only emphasize the differences when constructing the directed

graph G'_{mix} for G_{mix}. For each input G_{mix} and vertices $s, t \in V_{mix}$, the directed flow network G'_{mix} is constructed as below.

1. For each $v \in V^{T_1}_{mix}$ ($v \neq s$ and $v \neq t$), add two vertices v', v'' into V'_{mix}, and the directed edge(v', v'') and (v'', v') into E'_{mix} with edge weights equal to k. And for each $u \in V^{T_2}_{mix}$, add two vertices u', u'' into V'_{mix}, and the directed edge(u', u'') and (u'', u') into E'_{mix} with edge weights equal to 1.
2. For each edge $(s, v) \in E_{mix}$, add edge (s, v') to E'_{mix} with weight $w(s, v)$; for each edge $(v, t) \in E_{mix}$, add edge (v'', t) to E'_{mix} with weight $w(v, t)$; for each edge connecting two type T_1 vertices, such as $(u, v) \in E_{mix}$, add two edges (u'', v') and (v'', u') to E'_{mix} and each edge has capacity $w(u, v)$; for each edge connecting type T_1 and T_2 vertices, such as $(x, y) \in E_{mix}$, add two edges (x'', y') and (y'', x') to E'_{mix} and each edge has capacity ∞.
3. Assign s as the source vertex and t as the sink vertex.

Next, we analyze the rationale to construct G'_{mix} like above. When the maximum flow (minimum cut) between s and t in G'_{mix} is less than k, G'_{mix} needs to be separated. For type T_1 vertices, the edges connecting two different type T_1 vertices can belong to the edge cut set. And based on the connectivity information, we can locate its corresponding vertex cut set. However, for the same T_1 vertex, e.g. v, by setting $w(v', v'')$ and $w(v'', v')$ to k, we find that v' and v'' are inseparable due to these two edges cannot be included in the edge cut set. This observation is in accordance with Lemma 2. On the other hand, for type T_2 vertices such as u, because $w(u', u'')$ and $w(u'', u')$ are equal to 1 in G'_{mix}, only these two edges are possible to be contained in the edge cut set. Further, the corresponding vertex u could be in the vertex cut set.

When the current induced subgraph $G_{mix}[S]$ cannot be further separated, we have to check if $G[map(S)]$ is a k-VCC in G. The process is similar to that in Section 6.2. Based on Theorem 9, we find out $v \in S$ such that $G[map(v)]$ is less k-vertex connected to $G[map(S) \setminus map(v)]$ and iteratively remove it from S, until all the vertices left are k-vertex connected in G; that is the current $G[map(S)]$ is a k-VCC. In addition, if the discarded v is a type T_1 vertex, $G[map(v)]$ is still a k-VCC.

By exploiting the top-down detection in Topdown_revise(), we can indicate that any super graph of the obtained subgraphs are not k-vertex connected, and the post process guarantees that they are k-vertex connected. Therefore, all these subgraphs are maximal k-vertex connected, i.e., k-VCCs.

6.4 Complexity analysis

We now analyze the computational complexity of the hybrid framework, which mainly consists of contracting the mixed graph (Section 6.2) and identifying k-VCCs from the contracted one (Section 6.3). For the former part, let N_{iter1} represent the number of iterations. In each iteration, $|\overline{E}_{mix}|$, $|\overline{E}^{upt}_{mix}|$ and $|\overline{V}^{map}_{mix}|$ denote the average number of edges in the current G_{mix}, the average number of edges that need to be updated and the average number of vertices, to which each vertex in V_{mix} corresponds in G. Thus, it takes $Time_1 = O(N_{iter1} \cdot (h \cdot l|\overline{E}_{mix}| + |\overline{E}^{upt}_{mix}| \cdot |\overline{V}^{map}_{mix}|))$ time to contract G_{mix}, where l and h are mentioned after Lemma 3. For the later part, let N_{iter2} denote the total number of subgraphs detected when dividing G^{cur}_{mix}, which represents the mixed graph after contracting. For a certain subgraph, $|\overline{V}^{cur}_{mix}|$ and $|\overline{E}^{cur}_{mix}|$ refer to its average number of vertices and edges. So, the computational cost is bounded by $Time_2 = O((|\overline{V}^{cur}_{mix}| - \delta - 1 + \delta(\delta - 1)/2) \cdot |\overline{E}^{cur}_{mix}| \cdot |\overline{V}^{cur}_{mix}|^{2/3} \cdot N_{iter2})$

Table 1 Statistics of real networks ($K = 10^3$ and $M = 10^6$)

Network	abbr.	$\|V(G)\|$	$\|E(G)\|$	d_{max}	$\mathcal{D}$	$\#C$
Yeast	YA	$6.5K$	$229K$	2587	86	–
Amazon	AZ	$335K$	$926K$	549	6	$151K$
DBLP	DP	$317K$	$1M$	343	113	$13K$
Youtube	YT	$1.1M$	$3M$	28754	51	$8K$
LiveJournal	LJ	$4M$	$35M$	14815	360	$287K$

where δ represent the largest vertex degree in G_{mix}^{cur}. Totally, the hybrid framework costs $Time_1 + Time_2$ time.

7 Experiments

We conduct extensive experiments to evaluate the effectiveness and efficiency of the proposed methods by using a variety of real and synthetic datasets. All algorithms are implemented in C++. All the experiments are conducted on a Linux Server with Intel Xeon 3.2 GHz CPU and 64 GB main memory.

7.1 Datasets and compared methods

The statistics of real networks used in the experiments are shown in Table 1. d_{max} denotes the maximum vertex degree of G. $\mathcal{D}$ is the degeneracy of G in Definition 5. $\#C$ is the number of ground-truth communities. The first Yeast dataset is a protein-protein interaction network downloaded from BioGRID.[1] The other four datasets are networks with ground-truth communities.[2] We abbreviate these datasets as YA, AZ, DP, YT and LJ.

First, we compare our k-VCC with k-CC [3] and k-ECC [6] for effectiveness evaluation. Especially, we use k-VCC-B and k-VCC-H to represent the heuristic and exact k-VCCs obtained from the bottom-up and hybrid frameworks, respectively. Secondly, we evaluate the following algorithms for efficiency comparison:

- TkVCC: the top-down framework for k-VCC detection shown in Algorithm 1, discussed in Section 4. This is also used as the baseline method.
- BkVCC-Ran: the bottom-up framework for k-VCC detection shown in Algorithm 2 with random vertex order priority strategy in Section 5.1.
- BkVCC-NI: the bottom-up framework for k-VCC detection shown in Algorithm 2 with non-increasing vertex order priority strategy in Section 5.1.
- BkVCC-ND: the bottom-up framework for k-VCC detection shown in Algorithm 2 with non-decreasing vertex order priority strategy in Section 5.1.
- HkVCC-ND: the hyrid framework for k-VCC detection shown in Algorithm 7 with non-decreasing vertex order priority strategy in Section 5.1.

In the end, we show the case studies of k-VCC for different applications.

[1] thebiogrid.org
[2] http://snap.standford.edu

7.2 Evaluation on real networks

Effectiveness evaluation To evaluate the effectiveness of different component models, we compare the proposed k-VCC with k-CC [3] and k-ECC [6] on 4 real datasets including AZ, DP, YT and LJ with ground-truth communities [48] under different types of criteria. In the experiments, for different input k, we detect the heuristic k-VCCs (k-VCC-B) by BkVCC-ND, the exact k-VCCs (k-VCC-H) by HkVCC-ND, the k-CC by the method in [3] and the k-ECCs by the method in [6] as communities, respectively.

First, we use F-score to measure the accuracy of the detected communities with regard to the ground-truth communities. Given the discovered community $G[S]$ and the ground-truth community $G[T]$, F-score is defined as $F(S,T) = 2 * \frac{prec(S,T)*rec(S,T)}{prec(S,T)+rec(S,T)}$ where $prec(S,T) = \frac{|S \cap T|}{|S|}$ represents the precision and $rec(S,T) = \frac{|S \cap T|}{|T|}$ represents the recall. We can see that higher F-score value means the detected community is more similar with the ground-truth.

For each discovered community S_i, we compute the F-score with every ground-truth community T_j of the dataset and choose the largest $F(S_i, T_j)$ as the final F-score, F_i of S_i. Further, we use the average value of all F_i, denote as $\overline{F}$ to represent the F-score corresponding to a given dataset. Figure 5 shows the F-scores of the compared methods for different value of k. We find that the k-VCC-B and k-VCC-H have relatively higher F-score on AZ, YT and LJ datasets than k-ECC and k-CC. This is because in these datasets, the ground-truth communities are defined based on common interest or function, which is very cohesive. Also, the k-VCC-B has higher F-score than k-VCC-H, when k is smaller. This is because k-VCC-H obtains the exact k-VCCs, whose scales are usually larger than that

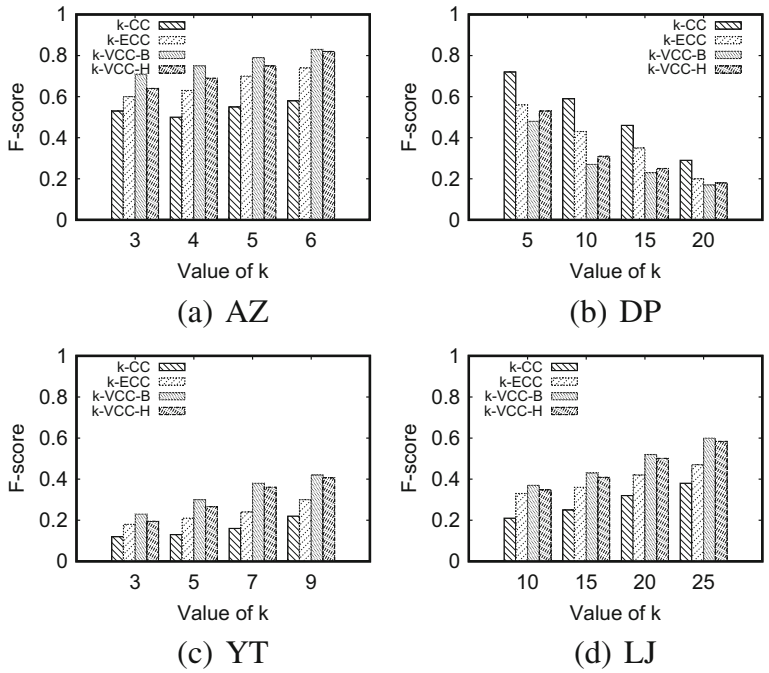

Figure 5 F-score on different real networks

of heuristic ones. Along with k becoming larger, the F-scores of k-VCC-B and k-VCC-H are almost the same, since the communities discovered with larger k-vertex connectivity are very similar.

However, the ground-truth community in DP is defined based on publication venues. The authors publishing papers in the same conference or journal may be not densely connected [48]. Thus, the trend we observe is opposite to the above three datasets, that is k-VCC-B has the lowest F-score on DP. Furthermore, k-VCC-B and k-VCC-H is almost the same along with the increasing of k.

Then, we use *density* and *diameter* to measure the goodness (cohesiveness) of the detected communities. Here, the *density* of a graph is defined as the fraction of the edges that appear between the vertices to that of all possible edges and the *diameter* is defined as the longest shortest paths between any pair of vertices in a network. Given a community $G[S]$, the density and diameter of a subgraph $G[S]$ is represented as $dens(G[S]) = \frac{2|E(S)|}{|S||S-1|}$ and $diam(G[S]) = max_{u,v \in V(G[S])} dist(u, v)$, respectively. We say a community is more cohesive than others when it possesses larger *density* and smaller *diameter*.

Figure 6 presents the density of the obtained k-CC, k-ECC, k-VCC-B and k-VCC-H for different k values on different datasets. we can observe that along with the increasing of k, the density of all these communities is becoming larger. The reason is that when k becoming larger, the vertices with smaller degree are continually removed. Thus, the communities with larger k values are more cohesive than that with smaller k values. In addition, for the same k-value, k-VCC has larger density than k-CC and k-ECC. This is because k-VCC is nested in k-CC and k-ECC, and the structure of k-VCC has more

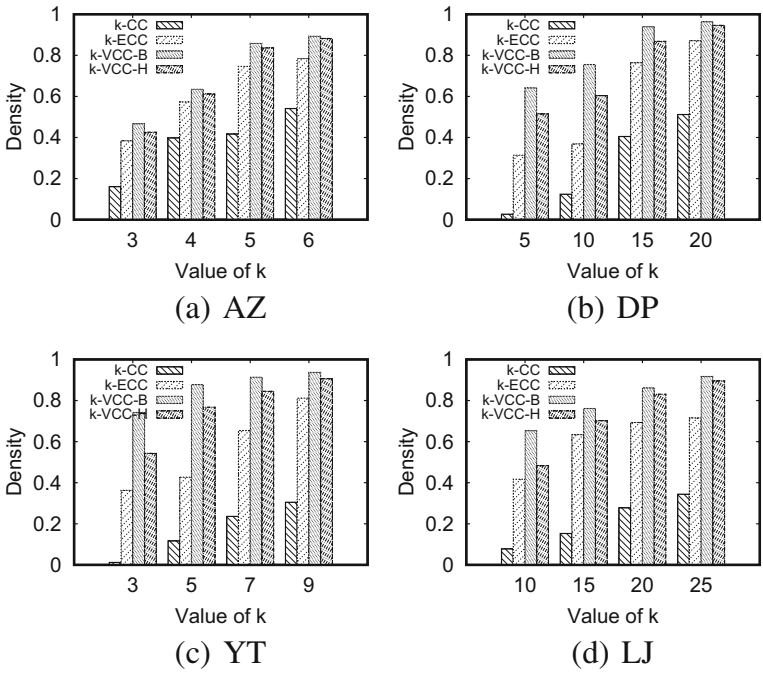

Figure 6 Density on different real networks

connectivity constraint and thus is more cohesive. We can also find that the detected k-VCC-B are more denser than k-VCC-H. This is because the exact k-VCC may be composed of different heuristic k-VCCs. Thus, when k is small, it is less cohesive than heuristic ones.

Figure 7 shows the diameter of the detected k-CC, k-ECC, k-VCC-B and k-VCC-H for different k on different networks. We can see that, opposite to the density, with the increasing of k, the diameter becomes smaller for all kinds of communities with similar reasons as density. Moreover, for the same k value, the diameter of k-VCC-B and k-VCC-H is both smaller than that of k-CC and k-ECC. That means k-VCC is more cohesive and thus can effectively reduce the occurrence of free rider effect.

Next, we present the number of k-vertex connected subgraphs (or k-VCCs) produced in different stages of various algorithms for different k. In particular, *seed*, *heuristic* and *exact* represents the number of seed subgraphs, heuristic k-VCCs and exact k-VCCs generated by the (BkVCC-ND) and (HkVCC-ND) methods, respectively. From Figure 8, we can see that *seed*, *heuristic* and *exact* are becoming smaller for the same k. This is because the seed subgraphs are expended and merged into heuristic k-VCCs and the heuristic are further contracted into the exact k-VCCs. However, for different k values, *seed* is getting smaller with the increasing of k, due to the number of local k-vertex connected subgraphs decreases when k raises. And the *heuristic* is first increases and then decreases. The reason is that when k is small, many seed subgraphs can be merged into the same larger subgraphs. At last, when k is large, the *heuristic* is very close to *exact* for at this moment, the target k-VCCs is very cohesive, and it is less possible that the heuristic k-VCCs far away from each other can be contracted into a larger one.

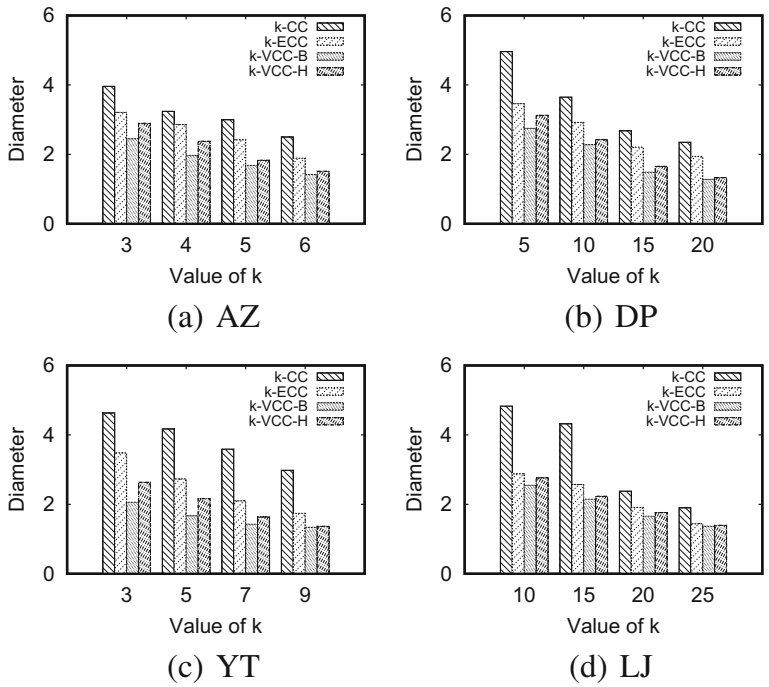

Figure 7 Diameter on different real networks

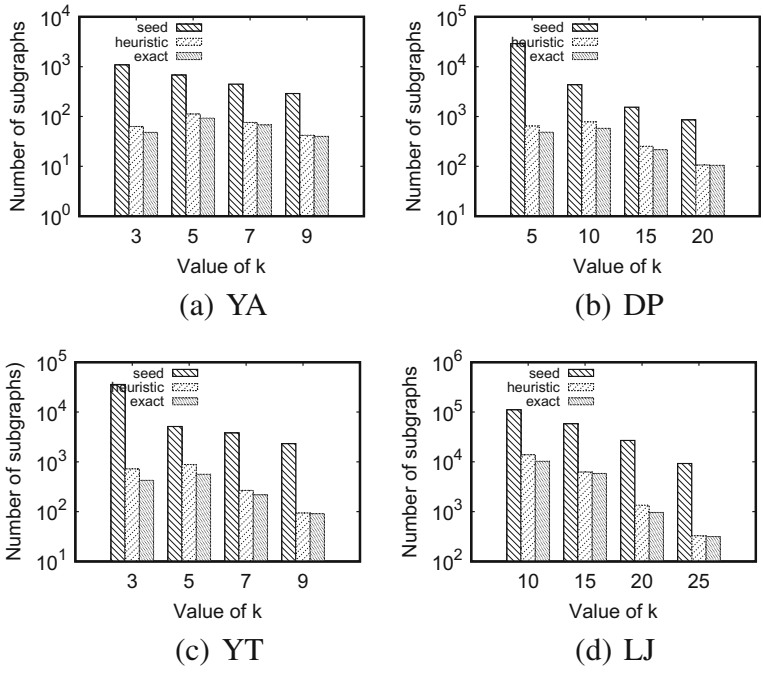

Figure 8 The number of subgraphs generated in different stages on different real networks

Efficiency evaluation In this section, we conduct experiments to evaluate the efficiency of top-down, bottom-up and hybrid frameworks for detecting k-VCCs on different real networks.

Figure 9 presents the overall running time of comparing methods, including TkVCC, BkVCC-Ran, BkVCC-NI, BkVCC-ND and HkVCC-ND for varying parameter k. We first observe that the TkVCC method always runs slowest on all the datasets. This is because that it exploits the structure of the entire graph to find the minimum vertex cut set. When the scale of the graph becoming larger, it will be more time-consuming. Thus, for large real network such as LJ shown in Figure 9d, the program even cannot end within the required time. For the bottom-up framework, BkVCC-ND method runs much faster than BkVCC-Ran and BkVCC-NI over all the datasets. Recall that in BkVCC-ND, we assign vertices with smaller vertex degree have higher priority, which reduce the combination number of the neighbors for a given vertex. When we visit vertices with large vertex degree, they are very probably having been included in the vertices with small degree, which reduces the running time a lot. We further evaluate the running time of hybrid framework (HkVCC-ND). Although the hybrid framework spends a little more time than the bottom-up framework (BkVCC-ND), it not only runs much faster than the top-down framework but can find out the exact k-VCCs. This is because the size of the constructed mixed graph is based on the results of bottom-up framework, which is much smaller than the original graph. Thus, the hybrid framework does not spend too much extra time in detecting the k-VCCs by the top-down manner.

On the other hand, along with the increasing of parameter k, the running time of these methods first increases. This is because it needs more time to compute minimum vertex cut for TkVCC and seed subgraphs for the bottom-up based methods. Then, when the k

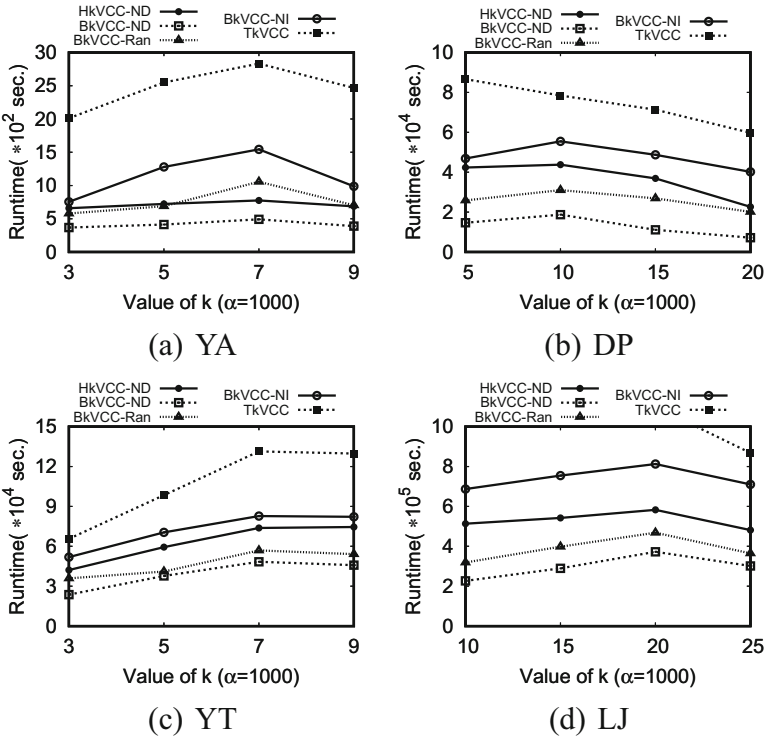

Figure 9 Runtime comparison between top-down and bottom-up frameworks on different real networks

value reaches a turning point, the running time begins to decrease. The reason is that with k becoming larger, more and more vertices are pruned by the k-core component.

Next, we test the memory usage of the HkVCC-ND algorithm, when varying k on different real datasets. Here, we compare the methods based on whether using the disjoint-set structure when merging the k-vertex connected subgraphs. We denote the methods with (or without) using the disjoint-set as HkVCC-wDS and HkVCC-woDS, respectively. Figure 10 presents the corresponding results. We can see that HkVCC-wDS utilizes smaller memory storage than that of HkVCC-woDS. This is because by disjoint-set, we can merge multiple subgraphs simultaneously, instead of combining only two subgraphs at each time. To do like this, it can large reduce the number of intermediate results. On the other hand, we observe that the memory usage decreases with the increasing of k. This is because although the number of results may get larger, the scale of the graphs and the number of seed subgraphs become smaller, and thus consume less memory.

7.3 Evaluation on synthetic networks

We generate a set of synthetic bipartite networks to evaluate the performance of the selected methods. The number of vertices are balanced in each part of these bipartite networks. The degree of both parts follow the power-law distribution with exponent γ and $d_{max} = n/2$. Here, we set $\gamma = 2$ as usual and the vertices in the networks are linked according to [34].

We evaluate the efficiency and effectiveness of the TkVCC and BkVCC-ND methods. We set $k = 4$ for all the situations. Figure 11a shows the running time when varying the number

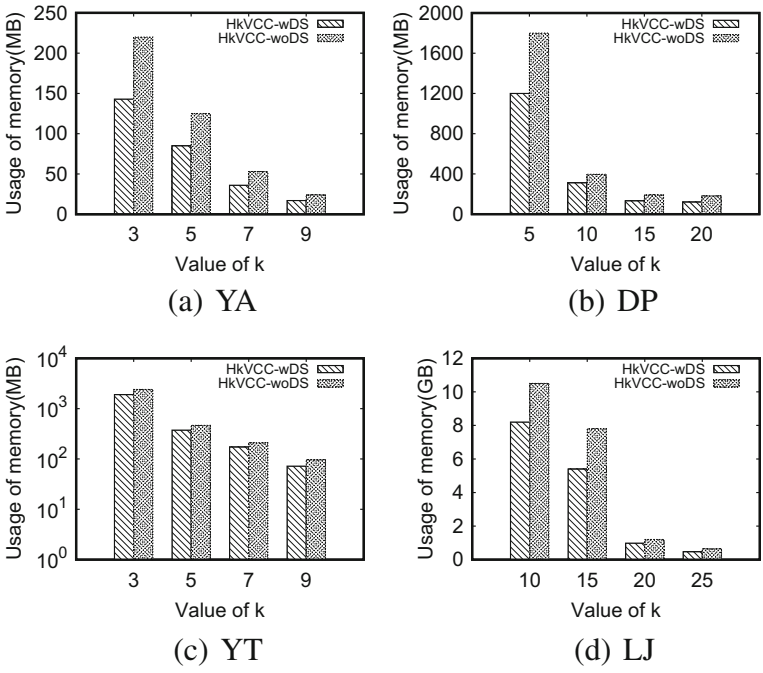

Figure 10 The memory usage of the HkVCC-ND algorithm on different real networks

of vertex in the network. We can see that BkVCC-ND method is much more efficient than TkVCC method, which is consistent with the results on real datasets. Figure 11b shows the F-score of the result of BkVCC-ND corresponding to that of TkVCC. Since the result of TkVCC are exact solution, the relative high values of F-score indicates that although the BkVCC-ND method is heuristic, it could generate results with high quality and hence proves its effectiveness.

7.4 Case studies

In this section, we explore two interesting networks, including the author collaboration network of DBLP and the genetic interaction network of hypertension (HT), that act as solid evidence of the utility of k-VCCs in the real world.

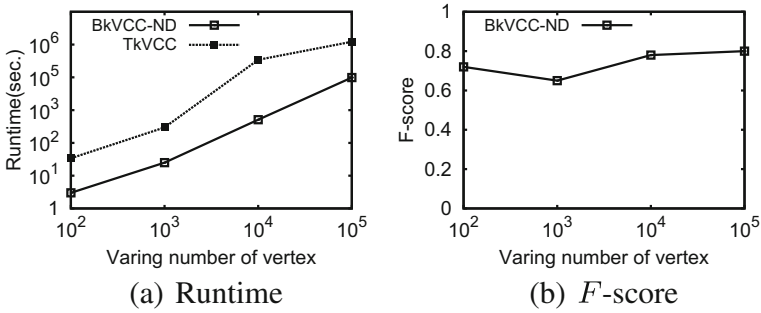

Figure 11 Results on synthetic network

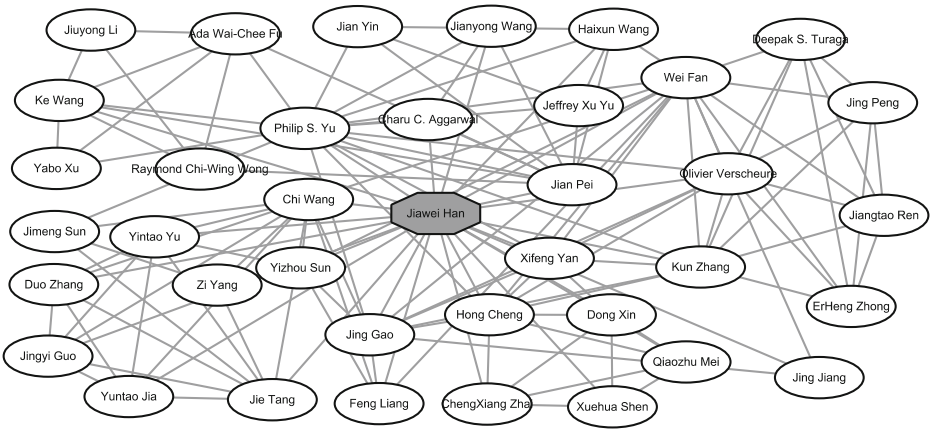

Figure 12 One discovered 3-VCC in DBLP containing Prof. Jiawei Han

Case study on DBLP We construct an author collaboration network on KDD conference extracted from the raw DBLP dataset (http://dblp.uni-trier.de/xml/) for case study. A vertex represents an author, and an edge between two authors indicates they have co-authored. Figure 12 presents one of the discovered 3-VCCs containing Prof. Jiawei Han. With the increasing of k, the current subgraph could be separated into different smaller subgraphs having higher vertex connectivity. For example, Figure 13a and b are two 4-VCCs derived from the graph in Figure 12. Based on the background knowledge, Figure 13a shows his cooperation with the group of his colleague, Prof. Chengxiang Zhai, when he began to work at UIUC. Figure 13b shows his research group at UIUC along with some often cooperative famous professors. In addition, Figure 13c presents another 4-VCC, which represents his co-authors when he worked at SFU. We also found that Prof. Jian Pei often cooperates with Prof. Jiawei Han in his research career. Oppositely, if we use 4-ECC or 4-Core, we can only acquire one community containing Prof. Jiawei Han. Thus, we say that the communities detected by k-VCCs are more reasonable and interpretable, which effectively reduces the free rider effect.

Case study on HT We also apply the k-VCC model in analyzing the genetic interaction network of HT. This network is produced based on the Welcome Trust Case Control Consortium (WTCCC) hypertension dataset [12]. Specifically, we first map all the genetic markers to their corresponding genes, and then calculate all the chi-square test values between genes.

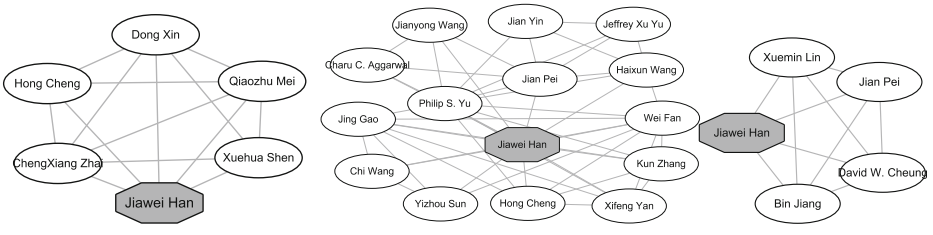

Figure 13 Some of 4-VCCs containing Prof. Jiawei Han

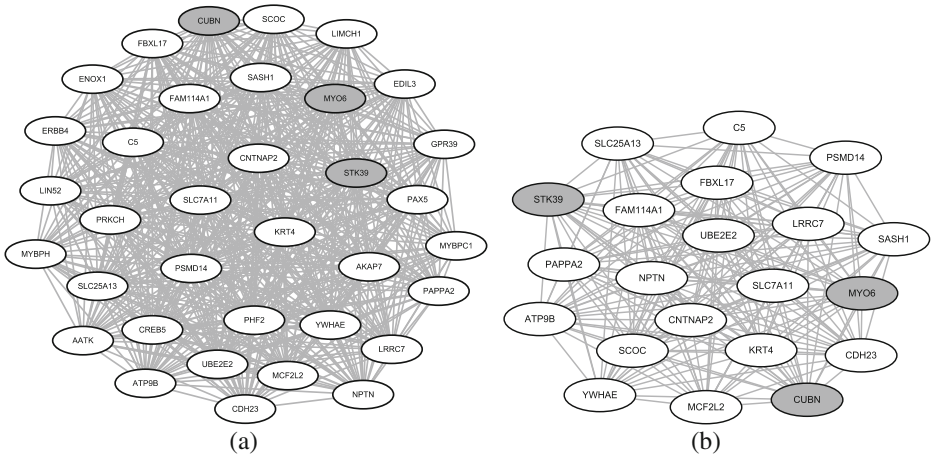

Figure 14 Identified k-VCCs with $k = 10$ and $k = 15$ on genetic interaction network of HT

Because different genetic markers may map to the same gene, for each pair of genes, we choose the most significant test value among all the corresponding genetic marker pairs as its genetic interaction value. Here, we set the chi-square value threshold to 70. If the interaction values is larger than this threshold, we add an edge between the genes. Finally, the genetic interaction network of HT contains 8468 vertices (genes) and 17741 edges (interactions).

Figure 14a and b shows the identified k-VCCs with $k = 10$ and $k = 15$, respectively. From the figure, it is clear that with the increasing of k, the scale of the identified subgraph become smaller. We can obtain different size of candidate results based on our requirement by tuning the value of k. More importantly, we find out several genes (with gray background in Figure 14) in both of these subgraphs that haven been verified to be associated with HT. For example, MYO6 gene encodes a reverse-direction motor protein that moves toward the minus end of actin filaments and plays a role in intracellular vesicle and organelle transport. It has been reported to have association with HT in [38]. The CUBN is a gene locus for albuminuria, and hypertension is a major risk factor for albuminuria [5]. Also, STK39 gene encodes a serine/threonine kinase that is thought to function in the cellular stress response pathway. This gene is obviously associated with HT and has been reported many times [8, 44]. Besides these three genes, other genes such as C5, SCOC, SLC25A13 and KRT4 are potential HT candidate genes, because these genes appear in important signal transduction in related HT pathway, which have been reported in [47].

8 Conclusion

Component detection is a fundamental problem in network analysis and has attracted intensive interests. Most existing component detection methods suffer from the low connectivity issue, whose results will contain irrelevant subgraphs. In this paper, we propose the k-vertex connected component model, which focuses on the vertex connectivity of networks. We study the k-VCC detection problem. First, we develop exact top-down and heuristic bottom-up frameworks for k-VCC detection. Further, we propose the hybrid framework, which takes

advantages of the above two frameworks. Although the hybrid framework is a little slower than the bottom-up framework, it can find the exact results. Through extensive experiments on large real and synthetic networks, we verify that the detected k-VCCs by both exact and heuristic frameworks are very effective, which largely reduce the free-ride effect, and the bottom-up and hybrid frameworks are much more efficient.

Acknowledgments This research is partially supported by the National NSFC (61672041, 61772124, 61732003,61902004,61977001), National Key Research and Development Program of China (2018YFB1004402), the Start-up Funds of North China University of Technology, and the National Research Foundation, Prime Ministers Office, Singapore under its International Research Centres in Singapore Funding Initiative and the Pinnacle lab for Analytics at SMU.

References

1. Adamcsek, B., Palla, G., Farkas, I., Derényi, I., Vicsek, T.: Cfinder: Locating cliques and overlapping modules in biological networks. Bioinformatics **22**(8), 1021–1023 (2006)
2. Akiba, T., Iwata, Y., Yoshida, Y.: Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In: CIKM, pp. 909–918 (2013)
3. Batagelj, V., Zaversnik, M.: An o (m) algorithm for cores decomposition of networks. arXiv:cs/0310049 (2003)
4. Berlowitz, D., Cohen, S., Kimelfeld, B.: Efficient enumeration of maximal k-plexes. In: SIGMOD, pp. 431–444 (2015)
5. Böger, C.A., Chen, M.H., Tin, A., Olden, M., Köttgen, A., de Boer, I.H., Fuchsberger, C., O'Seaghdha, C.M., Pattaro, C., Teumer, A., et al: Cubn is a gene locus for albuminuria. J. Am. Soc. Nephrol. **22**(3), 555–570 (2011)
6. Chang, L., Yu, J.X., Qin, L., Lin, X., Liu, C., Liang, W.: Efficiently computing k-edge connected components via graph decomposition. In: SIGMOD, pp. 205–216 (2013)
7. Chang, L., Lin, X., Qin, L., Yu, J.X., Zhang, W.: Index-based optimal algorithms for computing steiner components with maximum connectivity. In: SIGMOD, pp. 459–474. ACM (2015)
8. Chen, L.Y., Zhao, W.H., Tian, W., Guo, J., Jiang, F., Jin, L.J., Sun, Y.X., Chen, K.M., An, L.L., Li, G., et al: Stk39 is an independent risk factor for male hypertension in Han Chinese. Int. J. Cardiol. **154**(2), 122–127 (2012)
9. Cheng, J., Ke, Y., Chu, S., Özsu, M.T.: Efficient core decomposition in massive networks. In: ICDE, pp. 51–62 (2011)
10. Christophides, V., Karvounarakis, G., Plexousakis, D., Scholl, M., Tourtounis, S.: Optimizing taxonomic semantic Web queries using labeling schemes. Web Semantics: Science Services and Agents on the World Wide Web **1**(2), 207–228 (2004)
11. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. National Security Agency Technical Report 16 (2008)
12. Consortium, W.T.C.C. et al.: Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. Nature **447**(7145), 661 (2007)
13. Conte, A., Firmani, D., Mordente, C., Patrignani, M., Torlone, R.: Fast enumeration of large k-plexes. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 115–124. ACM (2017)
14. Cui, W., Xiao, Y., Wang, H., Wang, W.: Local search of communities in large graphs. In: SIGMOD, pp. 991–1002 (2014)
15. Diestel, R.: Graph theory. Grad Texts in Math (2005)
16. Esfahanian, A.H., Louis Hakimi, S.: On computing the connectivities of graphs and digraphs. Networks **14**(2), 355–366 (1984)
17. Even, S., Tarjan, R.E.: Network flow and testing graph connectivity. SIAM J. Comput. **4**(4), 507–518 (1975)
18. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3), 75–174 (2010)
19. Gregory, S.: Finding overlapping communities in networks by label propagation. J. Phys. **12**(10), 103018 (2010)
20. Hariharan, R., Kavitha, T., Panigrahi, D., Bhalgat, A.: An o (mn) gomory-hu tree construction algorithm for unweighted graphs. In: ACM Symposium on Theory of Computing, pp. 605–614 (2007)

21. Hu, J., Wu, X., Cheng, R., Luo, S., Fang, Y.: Querying minimal steiner maximum-connected subgraphs in large graphs. In: CIKM, pp. 1241–1250. ACM (2016)
22. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: SIGMOD, pp. 1311–1322 (2014)
23. Huang, X., Lu, W., Lakshmanan, L.V.: Truss decomposition of probabilistic graphs: Semantics and algorithms. In: ACM Proc. of SIGMOD, pp. 77–90. ACM (2016)
24. Kane, V., Mohanty, S.: A lower bound on the number of vertices of a graph. Proc. Am. Math. Soc. **72**(1), 211–212 (1978)
25. Kargar, M., An, A.: Keyword search in graphs: Finding r-cliques. PVLDB **4**(10), 681–692 (2011)
26. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: KDD, pp. 467–476. ACM (2009)
27. Lee, C., Reid, F., McDaid, A., Hurley, N.: Detecting highly overlapping community structure by greedy clique expansion. arXiv:1002.1827 (2010)
28. Li, Y., Zhao, Y., Wang, G., Zhu, F., Wu, Y., Shi, S.: Effective k-vertex connected component detection in large-scale networks. In: International Conference on Database Systems for Advanced Applications, pp. 404–421. Springer (2017)
29. Li, L., Zheng, K., Wang, S., Hua, W., Zhou, X.: Go slow to go fast: Minimal on-road time route scheduling with parking facilities using historical trajectory. VLDB J. **27**(3), 321–345 (2018)
30. Lian, D., Zheng, K., Ge, Y., Cao, L., Chen, E., Xie, X.: Geomf++: Scalable location recommendation via joint geographical modeling and matrix factorization. ACM Trans. Inf. Syst. (TOIS) **36**(3), 33 (2018)
31. Lim, S., Ryu, S., Kwon, S., Jung, K., Lee, J.G.: Linkscan*: Overlapping community detection using the link-space transformation. In: ICDE, pp. 292–303. IEEE (2014)
32. Liu, G., Liu, Y., Zheng, K., Liu, A., Li, Z., Wang, Y., Zhou, X.: Mcs-gpm: Multi-constrained simulation based graph pattern matching in contextual social graphs. IEEE Trans. Knowl. Data Eng. **30**(6), 1050–1064 (2017)
33. Mokken, R.J.: Cliques, clubs and clans. Quality & Quantity **13**(2), 161–173 (1979)
34. Molloy, M., Reed, B.: The size of the giant component of a random graph with a given degree sequence. Comb. Probab. Comput. **7**(03), 295–305 (1998)
35. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**(7043), 814–818 (2005)
36. Pattillo, J., Youssef, N., Butenko, S.: On clique relaxation models in network analysis. Eur. J. Oper. Res. **226**(1), 9–18 (2013)
37. Shan, J., Shen, D., Nie, T., Kou, Y., Yu, G.: Searching overlapping communities for group query. World Wide Web **19**(6), 1179–1202 (2016)
38. Slavin, T.P., Feng, T., Schnell, A., Zhu, X., Elston, R.C.: Two-marker association tests yield new disease associations for coronary artery disease and hypertension. Human Gen. **130**(6), 725–733 (2011)
39. Sozio, M., Gionis, A.: The community-search problem and how to plan a successful cocktail party. In: SIGKDD, pp. 939–948 (2010)
40. Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM (JACM) **44**(4), 585–591 (1997)
41. Sun, H., Huang, J., Bai, Y., Zhao, Z., Jia, X., He, F., Li, Y.: Efficient k-edge connected component detection through an early merging and splitting strategy. Knowl.-Based Syst. **111**, 63–72 (2016)
42. Wang, J., Cheng, J.: Truss decomposition in massive networks. PVLDB **5**(9), 812–823 (2012)
43. Wang, N., Zhang, J., Tan, K.L., Tung, A.K.: On triangulation-based dense neighborhood graph discovery. PVLDB **4**(2), 58–68 (2010)
44. Wang, Y., O'Connell, J.R., McArdle, P.F., Wade, J.B., Dorff, S.E., Shah, S.J., Shi, X., Pan, L., Rampersaud, E., Shen, H., et al.: Whole-genome association study identifies stk39 as a hypertension susceptibility gene. Proc. Natl. Acad. Sci. **106**(1), 226–231 (2009)
45. Wu, Y., Jin, R., Li, J., Zhang, X.: Robust local community detection: On free rider effect and its elimination. PVLDB **8**(7), 798–809 (2015)
46. Wu, Y., Jin, R., Zhu, X., Zhang, X.: Finding dense and connected subgraphs in dual networks. In: ICDE, pp. 915–926 (2015)
47. Wu, Y., Zhu, X., Li, L., Fan, W., Jin, R., Zhang, X.: Mining dual networks: Models, algorithms and applications. TKDD **10**(4), 40 (2016)
48. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. In: ICDM, pp. 745–754 (2012)
49. Zeng, Z., Wang, J., Zhou, L., Karypis, G.: Coherent closed quasi-clique discovery from large dense graph databases. In: KDD, pp. 797–802 (2006)
50. Zhao, Y., Zheng, K., Li, Y., Su, H., Liu, J., Zhou, X.: Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach. IEEE Transactions on Knowledge and Data Engineering (2019)

51. Zheng, K., Zheng, Y., Yuan, N.J., Shang, S., Zhou, X.: Online discovery of gathering patterns over trajectories. IEEE Trans. Knowl. Data Eng. **26**(8), 1974–1988 (2013)
52. Zheng, B., Su, H., Hua, W., Zheng, K., Zhou, X., Li, G.: Efficient clue-based route search on road networks. IEEE Trans. Knowl. Data Eng. **29**(9), 1846–1859 (2017)
53. Zheng, K., Zhao, Y., Lian, D., Zheng, B., Liu, G., Zhou, X.: Reference-based framework for spatio-temporal trajectory compression and query processing. IEEE Transactions on Knowledge and Data Engineering (2019)
54. Zhou, R., Liu, C., Yu, J.X., Liang, W., Chen, B., Li, J.: Finding maximal k-edge-connected subgraphs from a large graph. In: EDBT, pp. 480–491 (2012)

Affiliations

Yuan Li[1] · Guoren Wang[2] · Yuhai Zhao[3] · Feida Zhu[4] · Yubao Wu[5]

1 North China University of Technology, Beijing, China
2 Beijing Institute of Technology, Beijing, China
3 Northeastern University, Shenyang, China
4 Singapore Management University, Singapore, Singapore
5 Georgia State University, Atlanta, GA, USA