Published in Database systems for advanced applications: 22nd International Conference, DASFAA 2017, Suzhou, China, March 27-30: Proceedings. pp. 404-421.
https://doi.org/10.1007/978-3-319-55699-4_25

Effective k-Vertex Connected Component Detection in Large-Scale Networks

Yuan Li[1], Yuhai Zhao[1(✉)], Guoren Wang[1], Feida Zhu[2], Yubao Wu[3], and Shengle Shi[1]

[1] Northeastern University, Shenyang, China
zhaoyuhai@ise.neu.edu.cn
[2] Singapore Management University, Singapore, Singapore
[3] Georgia State University, Atlanta, USA

Abstract. Finding components with high connectivity is an important problem in component detection with a wide range of applications, e.g., social network analysis, web-page research and bioinformatics. In particular, k-edge connected component (k-ECC) has recently been extensively studied to discover disjoint components. Yet many real applications present 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 therefore allows overlapping between components. To find k-VCCs, a top-down framework is first developed to find the exact k-VCCs. To further reduce the high computational cost for input networks of large sizes, a bottom-up framework is then proposed. Instead of using the structure of the entire network, it locally identifies the seed subgraphs, and obtains the heuristic k-VCCs by expanding and merging these seed subgraphs. Comprehensive experimental results on large real and synthetic networks demonstrate the efficiency and effectiveness of our approaches.

1 Introduction

Component detection is a fundamental problem [11,19] in the analysis of large-scale networks. Many real applications can benefit from finding highly connected components. For example, groups of intimate entities discovered in social networks can be exploited to analyze their social behaviors [11]; a set of servers with common contents in web server networks can be used to construct the network index [4]; clusters of interactive genetic markers discovered in genetic interaction networks can be utilized to infer the corresponding cause of diseases [25].

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 [3,6,27], distance-relaxed [14,16] and triangulation-relaxed [22,23] methods. Although succeeding to some extent, these methods still have respective drawbacks.

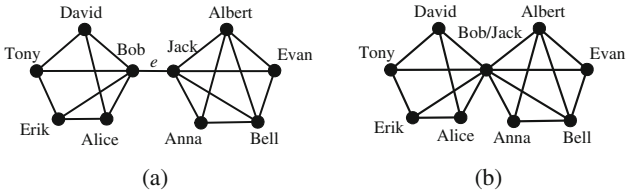


Fig. 1. A toy co-friendship network.

A toy co-friendship network is considered in Fig. 1(a) as an example. 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 Fig. 1(a) 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 steadiness and robustness of the component. A component with high connectivity could still be connected, even if losing a few relations or entities. Therefore, recently, connectivity-based methods such as k-edge connected components (k-ECC) have drawn great attention [1,5,28]. 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. The network in Fig. 1(a) 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*, Fig. 1(a) is equivalent to Fig. 1(b). In this case, Fig. 1(b) 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 Fig. 1(b) can be identified as two 3-VCCs, {*Bob/Jack, David, Tony, Erik, Alice*} and {*Bob/Jack, Anna, Bell, Evan, Albert*}. Unlike k-ECC, k-VCCs has three unique advantages: (1) k-VCC captures more connectivity of networks than k-ECC. According to [8], 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 in Figure 1(b), which is more natural and reasonable for real-world networks [18]; (3) k-VCC can prevent the detected communities from including irrelevant subgraphs, i.e. free rider effect [24].

Unfortunately, the methods for k-ECC [1,5] cannot be directly utilized to find k-VCCs. Because each vertex only belongs to at most one k-ECC [5], these methods could obtain k-ECCs by vertex contraction [21]. In our case, however, each vertex can be in more than one k-VCCs, which makes the former trick not work. In this paper, we devise two novel frameworks to tackle the k-VCC detection problem, namely top-down and bottom-up 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.

Our contributions are summarized as below: (1) a novel k-VCC detection problem is proposed from the perspective of vertex connectivity; (2) the top-down and bottom-up frameworks are developed to solve the problem. Specifically, in the bottom-up framework, a concept of local k-vertex connected subgraph is proposed to accelerate k-VCC detection, which enables identifying seed subgraphs locally instead of globally. In addition, several optimization techniques are proposed to further reduce the search space; (3) extensive experiments on both real and synthetic datasets demonstrate the efficiency and effectiveness of our frameworks.

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

2 Notions and Problem Statement

In this paper, we focus on an undirected and unweighted graph $G(V, E)$, where V is the set of vertices and E is the set of edges. We denote the number of vertices and the number of edges by $n = |V|$ and $m = |E|$, respectively. A graph $G[S]$ is called an *induced subgraph* of G when $S \subseteq V$, and $E(S) = \{(u, v) \in E | u, v \in S\}$. We use $nb_G(v)$ to denote the set of neighbors of a vertex v in G, that is, $nb_G(v) = \{u|(u, v) \in E\}$. We define the degree of v in G as $deg_G(v) = |nb(v)|$. If there is no ambiguity, we denote them as $nb(v)$ and $deg(v)$. In addition, d_{max} denotes the maximum vertex degree of G.

Notions. We first give some formal definitions used in this work.

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, $\kappa(u, v)$ 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 if $(u, v) \in E$, then set $\kappa(u, v) = n - 1$.*

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 $G[V \backslash S]$ 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: $\kappa(G) = \min\{\kappa(v, w) | unordered\ pair\ v, w\ in\ G\}$.

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, in other words, $\kappa(G) \geq k$.*

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 Fig. 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 efficiently computing 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, which is the maximal subgraph $G[C_k]$ 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.*

Proof. Based on Definition 3, in each k-VCC $G[S], \forall u \in S, deg_{G[S]}(u) \geq k$. And, k-core is the maximal subgraph $G[V_k]$ of G such that $\forall v \in V_k, deg_{G[V_k]}(v) \geq k$. Thus, for any vertex in the k-VCC, it must be contained in the corresponding k-core subgraph. Also, k-VCC and k-core are induced subgraphs, hence all the edges in k-VCC are contained in the k-core.

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., $\mathcal{D} = \max_{v \in V} \psi(v)$.*

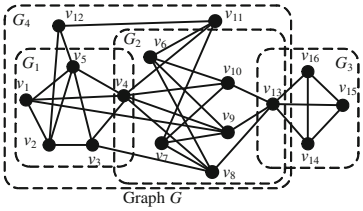


Fig. 2. An example of k-VCCs

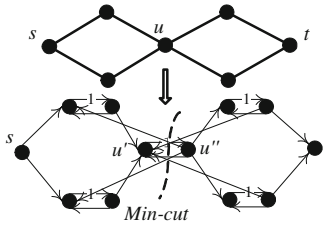


Fig. 3. An example of computing minimum vertex cut

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. The degeneracy $\mathcal{D}$ of graph G could be calculated efficiently by the algorithm proposed by Batagelj and Zaversnik [2], which repeatedly remove vertices from G whose degree is less than k until no more vertices can be removed and use bin-sort to order the vertices to achieve $O(m + n)$ time complexity. Therefore, once finding out the given k value is larger than $\mathcal{D}$, we will make sure that the result for k-VCCs in G is $\emptyset$.

3 Top-Down Framework for k-VCCs Detection

In this section, we detail the top-down framework for k-VCCs detection. The main idea of this framework is to iteratively compute the minimum vertex cut set V_{cut} of the current graph $G[C]$, 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.

Another important problem is how to find the minimum vertex cut set (line 9). Unlike the min-cut [21] and GomoryHu tree [12] methods, which can efficiently find the minimum edge cut set in an undirected graph, we have to reduce the problem of computing $\kappa(G)$ into a maximum flow problem in directed graph.

For each input $G(V, E)$ and vertices s, t, we construct the directed flow network $G'(V', E')$ as follows.

1. For each $v \in V$ ($v \neq s$ and $v \neq t$), add two vertices v', v'' into V', and the directed edge (v', v'') and (v'', v') into E'. The edge (v', v'') has weight 1 and (v'', v') has weight ∞.

Algorithm 1. Top-down Framework

Input: $G(V, E)$, k
Output: The set of k-VCCs $\mathcal{G}[\mathcal{R}]$

```
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];
```

2. For each edge $(s, v) \in E$, add edge (s, v') to E'; for each edge $(v, t) \in E$, add edge (v'', t) to E'; for each other edge $(u, v) \in E$, add two edges (u'', v') and (v'', u') to E'. Each edge has capacity ∞.
3. Assign s as the source vertex and t as the sink vertex.

Even and Tarjan [10] 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'. Figure 3 shows an example of the above process. Also, $\kappa(G)$ can be calculated in $O(n - \delta - 1 + \delta(\delta - 1)/2)$ calls to maximum flow algorithm where $\delta = d_{max}$ [9]. 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 $u, v \in G[C]K$, $G[C]$ 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 2 guarantees the top-down framework is correct.

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 $O((n' - \delta - 1 + \delta(\delta - 1)/2) \cdot m'n'^{2/3} \cdot L)$.

Theorem 2. *Given a graph G and a value k, the top-down framework for k-VCC detection can correctly find all the k-VCCs.*

Proof. Suppose a graph $G[V_0] \in \mathcal{G}[\mathcal{R}]$ is k-vertex connected subgraph, but not a k-VCC, which indicates it is not maximal, then there must be a k-VCC $G[V_{max}]$ such that $V_0 \subset V_{max}$, and there must also exist a vertex cut set in some loop which separates a vertex or some vertices in V_{max} away from V_0. However, this cannot happen because $G[V_{max}]$ is supposed to be k-vertex connected. Therefore, $G[V_0]$ is a k-VCC. In addition, Algorithm 1 operates until Q is empty, which means all the subgraphs have been processed. Thus, the theorem is correct.

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 Subsection 4.1;
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 Subsection 4.2;
7 $\quad \mathcal{G}[\mathcal{S}] \leftarrow$ Merging($G[V_k]$, k, $\mathcal{G}[\mathcal{S}]$); //detailed in Subsection 4.2;
8 $\mathcal{G}[\mathcal{R}] \leftarrow \mathcal{G}[\mathcal{R}] \cup \mathcal{G}[\mathcal{S}]$;
9 **return** $\mathcal{G}[\mathcal{R}]$;

4 Bottom-Up Framework for k-VCCs Detection

The top-down framework highly depends on global structure of graph, which may not be efficient and practical when graph scale becomes huge. In this section, we develop a bottom-up framework for k-VCCs detection.

The Bottom-up framework 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. First, we utilize Seeding() to find local k-vertex connected subgraphs around the neighborhood of vertices as seed subgraphs (line 3). Then, we exploit Expanding() and Merging() to expand and merge these seed subgraphs (lines 4–7). Although this framework is heuristic, the experiment in Sect. 5.3 shows that the result is comparable to the real.

4.1 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 [15], the local k-vertex connected subgraph is more generalized as seed than clique, whose structure is too strict [19].

In this section, seeding() is developed to identify such graphs. And there exists two important problems: (1) how to find the seed subgraph for a given 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 accelerate the process of identifying seed subgraphs, in which we do not have to identify seed subgraphs for all the vertices.

Identifying seed subgraph for a given vertex. Here, we study how to define and find local k-vertex connected subgraph for a given vertex u. We first give the definition of 2-ego neighborhood. We then exploit 2-ego neighborhood to

add additional constraint on the path length between vertices in the defined local k-vertex connected subgraph, which makes it possible to find the seed subgraph within the neighborhood of a 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 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) $u \in S$.*

Based on [7], in real networks, community is usually existed in the neighborhood of vertices, hence it is rational to define the local k-vertex connected subgraph 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.

Further, we propose the LkVCS method, which can find one of the local k-vertex connected subgraphs from the induced subgraph $G[N_2(u)]$ as the seed subgraph for u. The main idea of LkVCS is to start with every different subset of vertices of size k from the neighborhood of u, denoted as R and then continue bring vertices from $P' \backslash R$ 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, γ subsets are randomly tested. Example 1 illustrates how the LkVCS method works.

Example 1. We apply LkVCS on the graph G in Fig. 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 compute the local k-vertex connected subgraph for every vertex in the network by LkVCS method. However, it is not efficient enough. Here, we devise two optimization strategies to further reduce the computational cost.

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

Theorem 3 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.

Example 2. We apply the above two strategies in seeding() procedure on the graph shown in Fig. 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.

4.2 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 [8] indicates that a graph is k-vertex connected if and only if it contains k vertex independent paths between any two 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., $\overline{S} = V \backslash S$, and $\delta\, S$ 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(), we add vertices that are connected to at least k vertices in $G[S]$ into the current subgraph as shown in Fig. 4. 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. Expanding() ensures that for each generated subgraph $G[S]$ in the result set $\mathcal{G}[\mathcal{S}]$, $\forall u \in V \backslash S, G[S \cup u]$ is not a k-vertex connected subgraph. Theorem 4 guarantees the correctness of the Expanding() approach.

Theorem 4. *Suppose $G[S]$ is a k-vertex connected subgraph. If vertex u is adjacent to at least k vertices in $\delta S, G[S \cup u]$ is also a k-vertex connected subgraph.*

Proof. Theorem 4 can be induced from Menger's Theorem [8].

Example 3. We use the graph in Fig. 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 one of the boundary vertices v_{11} is adjacent to 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 shown in Fig. 5. Here, we develop Merging() to integrate different k-vertex connected subgraphs with at least k direct independent paths into a new one.

The process of Merging() is described as bellow. We first detect whether the input subgraphs in $\mathcal{G}[\mathcal{S}]$ are k-VCCs. If so, we put these subgraphs into the result set $\mathcal{G}[\mathcal{R}]$. Then, we iteratively merge the subgraphs satisfying the condition in Theorem 5 that will be detailed later until no subgraphs can be merged. If the subgraph $G[S \cup S']$ after combined meets the conditions in Corollary 1, we directly put it into result set $\mathcal{G}[\mathcal{R}]$, otherwise we put it back the candidate set $\mathcal{G}[\mathcal{S}]$ for further processing. In the implementation, we use the disjoint sets structure to accelerate the merging operation.

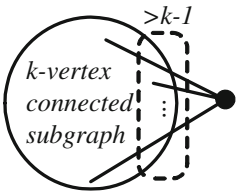


Fig. 4. The process of Expanding

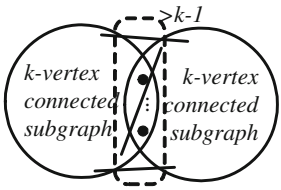


Fig. 5. The process of Merging

Formally, Theorem 5 provides the sufficient condition to guarantee Merging() is correct. If the sum of the number of overlapping vertices and length-1 independent paths between two k-vertex connected subgraphs is more than or equal to k, they can be combined together.

Theorem 5. *Let $G(V, E)$ be a graph, $S \subseteq V, S' \subseteq V$ and $G[S], G[S']$ be two k-vertex connected subgraphs. If the following condition is satisfied, we say $G[S \cup S']$ is a k-vertex connected subgraph: $|S \cap S'| + \min\{|\delta S \cap \delta\overline{S'}|, |\delta\overline{S} \cap \delta S'|\} \geq k$.*

Proof. The idea of this proof is similar to that of Theorem 4.

Furthermore, based on Theorems 4 and 5, 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 Fig. 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 5.

5 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.

5.1 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.

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	$287K$

We compare our k-VCC with k-CC [2] and k-ECC [5] for effectiveness evaluation. Further, we evaluate the following algorithms for efficiency comparison:

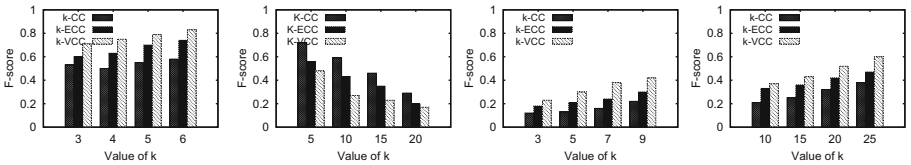
- TkVCC: the top-down framework for k-VCC detection shown in Algorithm 1, discussed in Sect. 3. This is also used as the baseline method.
- BkVCC-Ran: the bottom-up framework for k-VCC detection shown in Algorithm 2 with random vertex order priority strategy in Sect. 4.1.
- BkVCC-NI: the bottom-up framework for k-VCC detection shown in Algorithm 2 with non-increasing vertex order priority strategy in Sect. 4.1.
- BkVCC-ND: the bottom-up framework for k-VCC detection shown in Algorithm 2 with non-decreasing vertex order priority strategy in Sect. 4.1.

5.2 Evaluation on Real Networks

Effectiveness Evaluation. To evaluate the effectiveness of different community models, we compare the proposed k-VCC with k-CC [2] and k-ECC [5] on 4 real datasets including AZ, DP, YT and LJ with ground-truth communities [26] under different types of criteria.

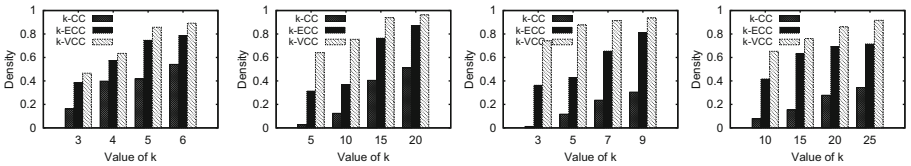
First, we use F-score to measure the accuracy of the detected communities with regard to the ground-truth communities. Given the discovered community $G[S]$ and the ground-truth community $G[T]$, F-score is defined as $F(S,T)=2*\frac{prec(S,T)*rec(S,T)}{prec(S,T)+rec(S,T)}$ where $prec(S,T) = \frac{|S\cap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

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



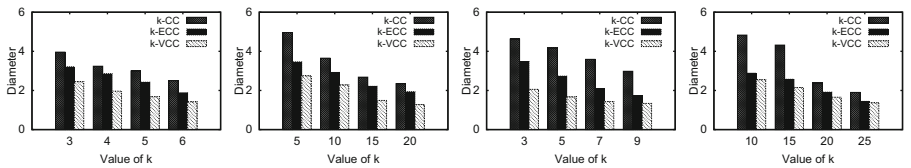
(a) AZ (b) DP (c) YT (d) LJ

Fig. 6. F-score on different real networks



(a) AZ (b) DP (c) YT (d) LJ

Fig. 7. Density on different real networks

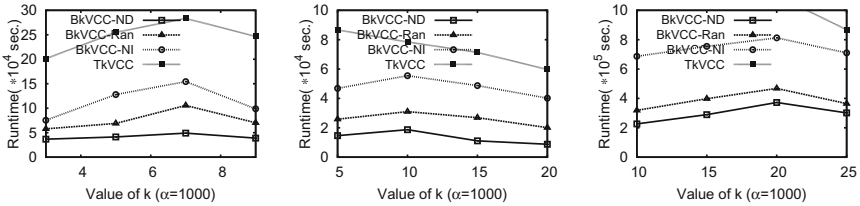


(a) AZ (b) DP (c) YT (d) LJ

Fig. 8. Diameter on different real networks

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.

In the experiments, for different input k, we detect the k-VCCs by BkVCC-ND, the k-CC by the method in [2] and the k-ECCs by the method in [5] as communities, respectively. 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 6 shows the F-scores of the compared methods for different value of k. We find that the k-VCCs have the highest F-score on AZ, YT and LJ datasets. This is because in these datasets, they defined the ground-truth communities based on common interest or function, which is very cohesive. 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 [26]. Thus, we see k-VCCs have the lowest F-score on DP.



(a) YA (b) DP (c) LJ

Fig. 9. Runtime on different real networks

Then, we use *density* and *diameter* to measure the goodness of the detected communities. *Density* is defined as the fraction of the edges that appear between the vertices to that of all possible edges and *diameter* is defined as the longest distance among all shortest paths between vertices in G. Given a community $G[S]$, the density and diameter of $G[S]$ is denoted $dens(G[S]) = \frac{2|E(S)|}{|S||S-1|}$ and $diam(G[S])$, respectively. The communities are more cohesive when they have larger *density* and smaller *diameter*. Figures 7 and 8 show the density and diameter of the detected communities on different networks. It can be seen that with the increasing of k, the density becomes larger and the diameter become smaller for all the methods. Moreover, for the same k value, k-VCC has the best performance. It has the highest density and lowest diameter, that is, the results of k-VCC are more cohesive than that of k-ECC and k-CC.

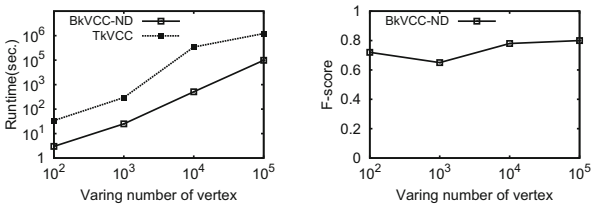
Efficiency Evaluation. In this section, we conduct experiments to study the efficiency of different methods to detect k-VCCs on different real networks. Figure 9 shows the comparison on overall running time of TkVCC, BkVCC-Ran, BkVCC-NI and BkVCC-ND for varying parameter k. We can see that the TkVCC 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 getting larger, it will be very time-consuming. Thus, for large real network such as LJ in Fig. 9(c), it even cannot finish within the required time.

On the contrary, 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. On the other hand, along with the increasing of parameter k, the running time of these methods first increase. 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 value reaches a turning point, the running time begin to decrease. The reason is that with k becoming larger, more and more vertices are pruned by the k-core component.

5.3 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 $d_{max} = n/2$. We set $\gamma = 2$. The vertices in the networks are linked according to [17].

We evaluate the efficiency and effectiveness of the TkVCC and BkVCC-ND methods. We set $k = 4$ for all the situations. Figure 10(a) shows the running time when varying the number 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 10(b) 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.



(a) Runtime (b) F-score

Fig. 10. Results on synthetic network

5.4 Case Study

We construct an author collaboration network on KDD conference extracted from the raw DBLP dataset[3] for case study. A vertex represents an author, and an edge between two authors indicates they have co-authored. Figure 11 presents three 4-VCCs containing professor Jiawei Han. Based on the background knowledge, Fig. 11(a) shows the research group when he worked at SFU. Figure 11(b) shows his cooperation with the group of his colleague Chengxiang Zhai, when he began to work at UIUC. And, Fig. 11(c) shows his research group at UIUC and some very famous professors. In particular, we can find that professor Jian Pei often cooperates with Jiawei Han. However, if we use 4-ECC, we can only acquire one community containing Jiawei Han. Thus, we say that the communities detected by k-VCCs are more reasonable and interpretable, which effectively reduces the free rider effect.

[3] http://dblp.uni-trier.de/xml/.

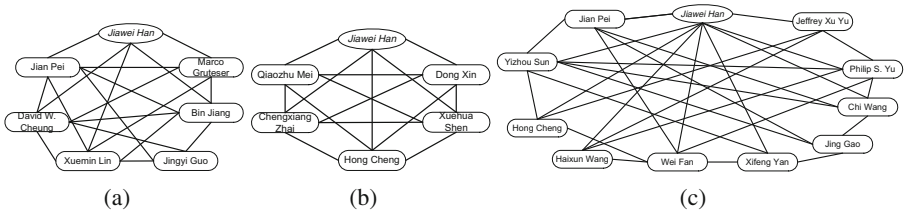


Fig. 11. Real examples of 4-VCCs containing Jiawei Han

6 Related Works

Our work relates to two main streams of research, concerning graph connectivity and component detection, 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 [9,10,12,21]. Recently, there exists several research on finding k-edge connected subgraphs, which concern about the local edge connectivity in the subgraphs [1,5,28]. Whereas, in this paper, we focus on the k-vertex connectivity of subgraphs.

Component detection. The existing component detection methods can be roughly devided into clique-based [19] 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 several categories: (1) **Distance-based relaxed methods**. *n-clique* is a maximal subgraph such that the distance of each pair of its vertices is not larger than n in the whole network [14]. *n-clan* is an n-clique whose diameter is no larger than n [16]. *n-club* is a maximal subgraph whose diameter is no larger than n [16]. (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 [3]. 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 [6,20] and local search method [7] 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 $\gamma * \binom{n}{2}$ edges [27]. (3) **Triangulation-based relaxed methods**. *DN*-graph [23] is a connected subgraph in which the lower bound of shared neighborhood between any connected vertices is locally maximized. k-truss [13,22] is the largest subgraph in which every edge is contained in at least $(k-2)$ triangles within the subgraph. Based on the limitation of the above methods detailed in Sect. 1, we propose the k-VCC model, which has high vertex connectivity.

7 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. 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 and develop the top-down and bottom-up frameworks for k-VCC detection. Extensive experimental results on large real and synthetic networks demonstrate the effectiveness and efficiency of our proposed approaches.

Acknowledgments. This research is partially supported by the National NSFC (No. 61272182, 61100028, 61332014, U1401256, 61672144), the Fundamental Research Funds for the Central Universities (N150402002, N150404008), 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. 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)
2. Batagelj, V., Zaversnik, M.: An o(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003)
3. Berlowitz, D., Cohen, S., Kimelfeld, B.: Efficient enumeration of maximal k-plexes. In: SIGMOD, pp. 431–444 (2015)
4. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the web. Comput. Netw. **33**(1), 309–320 (2000)
5. 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)
6. Cheng, J., Ke, Y., Chu, S., Özsu, M.T.: Efficient core decomposition in massive networks. In: ICDE, pp. 51–62 (2011)
7. Cui, W., Xiao, Y., Wang, H., Wang, W.: Local search of communities in large graphs. In: SIGMOD, pp. 991–1002 (2014)
8. Diestel, R.: Graph Theory. Graduate Texts in Mathematics. Springer, Heidelberg (2005)
9. Esfahanian, A.H., Louis Hakimi, S.: On computing the connectivities of graphs and digraphs. Networks **14**(2), 355–366 (1984)
10. Even, S., Tarjan, R.E.: Network flow and testing graph connectivity. SIAM J. Comput. **4**(4), 507–518 (1975)
11. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3), 75–174 (2010)
12. 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)
13. 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)

14. Kargar, M., An, A.: Keyword search in graphs: finding r-cliques. PVLDB **4**(10), 681–692 (2011)
15. Lee, C., Reid, F., McDaid, A., Hurley, N.: Detecting highly overlapping community structure by greedy clique expansion. arXiv preprint arXiv:1002.1827 (2010)
16. Mokken, R.J.: Cliques, clubs and clans. Qual. Quant. **13**(2), 161–173 (1979)
17. Molloy, M., Reed, B.: The size of the giant component of a random graph with a given degree sequence. Comb. Probab. Comput. **7**(3), 295–305 (1998)
18. 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)
19. Pattillo, J., Youssef, N., Butenko, S.: On clique relaxation models in network analysis. Eur. J. Oper. Res. **226**(1), 9–18 (2013)
20. Sozio, M., Gionis, A.: The community-search problem and how to plan a successful cocktail party. In: SIGKDD, pp. 939–948 (2010)
21. Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM (JACM) **44**(4), 585–591 (1997)
22. Wang, J., Cheng, J.: Truss decomposition in massive networks. PVLDB **5**(9), 812–823 (2012)
23. Wang, N., Zhang, J., Tan, K.L., Tung, A.K.: On triangulation-based dense neighborhood graph discovery. PVLDB **4**(2), 58–68 (2010)
24. 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)
25. Wu, Y., Jin, R., Zhu, X., Zhang, X.: Finding dense and connected subgraphs in dual networks. In: ICDE, pp. 915–926 (2015)
26. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. In: ICDM, pp. 745–754 (2012)
27. Zeng, Z., Wang, J., Zhou, L., Karypis, G.: Coherent closed quasi-clique discovery from large dense graph databases. In: KDD, pp. 797–802 (2006)
28. 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)