Enumerating k-Vertex Connected Components in Large Graphs

Dong Wen*, Lu Qin*, Ying Zhang*, Lijun Chang† and Ling Chen*

*Centre for Artificial Intelligence, University of Technology Sydney, Australia
{dong.wen, lu.qin, ying.zhang, ling.chen}@uts.edu.au;

†The University of Sydney, Australia
lijun.chang@sydney.edu.au;

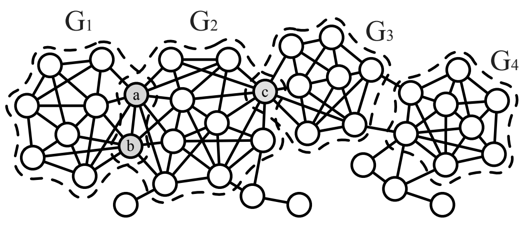
***Abstract*—In social network analysis, structural cohesion (or vertex connectivity) is a fundamental metric in measuring the cohesion of social groups. Given an undirected graph, a k-vertex connected component (k-VCC) is a maximal connected subgraph whose structural cohesion is at least k. A k-VCC has many outstanding structural properties, such as high cohesiveness, high robustness, and subgraph overlapping. In this paper, given a graph G and an integer k, we study the problem of computing all k-VCCs in G. The general idea for this problem is to recursively partition the graph into overlapped subgraphs. We prove the upper bound of the number of partitions, which implies the polynomial running time algorithm for the k-VCC enumeration. However, the basic solution is costly in computing the vertex cut. To improve the algorithmic efficiency, we observe that the key is reducing the number of local connectivity testings. We propose two effective optimization strategies, namely neighbor sweep and group sweep, to significantly reduce the number of local connectivity testings. We conduct extensive performance studies using ten large real datasets to demonstrate the efficiency of our proposed algorithms. The experimental results demonstrate that our approach can achieve a speedup of up to two orders of magnitude compared to the state-of-the-art algorithm.**

I. Introduction

With the proliferation of social information in networks, *social network analysis* [1] is an important subject, and significant research efforts have been done towards many fundamental problems in investigating social structures in networks [1], such as pagerank [2], centrality [3] and modularity [4]. In the social network group, *structural cohesion* [2] refers to the minimum number of members who, if removed from the group, would disconnect this group. It is identical to the conception of *vertex connectivity* in graph theory. The structural cohesion is a fundamental and important sociological metric in measuring the cohesion of social groups [5], [6].

Given an undirected graph G, a k-vertex connected component (k-VCC) [3], also named *k-component* [7] or *k-block* [8], is *a maximal connected subgraph whose vertex connectivity is at least k*. Fig. 1 shows an example; there are four 4-VCCs, namely G_1, G_2, G_3, and G_4 in G. The subgraph formed by the union of G_1 and G_2 is not a k-VCC, because it will be disconnected by removing two vertices a and b.

[1]https://en.wikipedia.org/wiki/Social_network_analysis
[2]https://en.wikipedia.org/wiki/Structural_cohesion
[3]https://en.wikipedia.org/wiki/K-vertex-connected_graph



4-VCCs: {G1, G2, G3, G4}

Fig. 1: An example for k-VCC in graph G.

In addition to the aforementioned powerful and robust sociological property of the vertex connectivity itself, the k-VCC has several outstanding structural properties. Firstly, given any graph G, Whitney Theorem [9] proves the vertex connectivity of G is no greater than the edge connectivity and the minimum degree. This implies a k-VCC is nested in a k-edge connected component (k-ECC) [10] and a k-core [11]. Therefore, a k-VCC is generally more cohesive and inherits all the structural properties of a k-ECC and a k-core. Secondly, small diameter is considered as an important feature for a good community in [12]. The diameter of a k-VCC $G'(V',E')$ is bounded by $\lceil \frac{|V'|-1}{\kappa(G')} \rceil$ where $\kappa(G')$ is the vertex connectivity of G' [13]. Thirdly, community overlap is regarded as an important feature of many real-world complex networks [14]. The vertex overlap is allowed between different k-VCCs, and the number of overlapped vertices between two k-VCCs is bounded by the parameter k. Even though overlap exists, the number of k-VCCs in the graph is bounded by $n/2$, where n is the number of vertices in the graph. The detailed proof can be found in Section IV. This indicates that redundancies in the k-VCCs are limited.

In this paper, given a graph G and an integer k, we aim to compute all k-VCCs in G. This problem has many applications. For example, in a social network, computing all k-VCCs can identify communities of highly related users, and provide valuable information for recommendation systems and advertisement platforms. In a co-authorship network, a k-VCC may be a research group. Some researchers may participate in several groups and perform as overlapped entities. This problem can also be used to visualize the graph structure [7].

Existing Solutions. Given a graph G and an integer k, a straightforward idea for computing k-VCCs is recursively finding a vertex cut with fewer than k vertices and partitioning the graph [15], [16], [17]. Here, a vertex cut of G is a set of vertices, the removal of which disconnects the graph. Based on the vertex cut, G is partitioned into overlapped subgraphs, each of which contains all vertices in the cut along with their incident edges. This approach recursively partitions each subgraph until no such cut exists. In this way, they compute all k-VCCs. For example, suppose the graph G is the union of G_1 and G_2 in Fig. 1. Given $k = 4$, we find a vertex cut containing two vertices a and b. We partition the graph into two subgraphs G_1 and G_2 that overlap two vertices a, b and an edge (a, b). Since neither G_1 nor G_2 has any vertex cut with fewer then k vertices, we compute G_1 and G_2 as 4-VCCs.

The key to this partition-based approach is computing the minimum vertex cut. Menger's Theorem [18] justifies that the minimum-size cut for a vertex pair is the number of vertex-independent paths between them. Based on this theorem, the minimum vertex cut can be computed in polynomial time using maximum flow techniques [15], [16].

Motivations. Even though the partition-based approach successfully computes the k-VCCs in a graph G, several challenges still remain. When performing a partition operation, overlapped vertices are duplicated, and the total number of partitions can be very large. Additionally, as discussed above, the crucial operation in the algorithm is computing the minimum vertex cut or, namely, local connectivity testing, which tests whether two vertices u and v can be disconnected in two components by removing at most $k - 1$ vertices from G. This is also the dominating cost in the algorithm. To find a vertex cut with fewer than k vertices, we need to conduct local connectivity testing between a source vertex s and each of other vertices v in G in the worst case. Given that straightforwardly using this framework is costly and not scalable to big graphs, a recent work [17] proposes an approximate solution. However, they do not have any approximation ratio. [19] also studies the same problem, while they design a special algorithm only for small parameter k such as 2 or 3.

Our Approaches. Given a graph G and an integer k, we first prove that both the number of overlapped partitions and the number of k-VCCs are bounded by $n/2$, which indicates the polynomial running time of the partition-based algorithm. We observe *the key to improving algorithmic efficiency is to reduce the number of local connectivity testings in a graph.* Given a source vertex s, if we can avoid testing the local connectivity between s and a certain vertex v, we call it as we can *sweep vertex v*. Given a graph G and a parameter k, we propose two strategies to sweep vertices.

- *Neighbor Sweep.* We locate a special kind of vertices with certain properties. Once any one of these vertices is tested or swept, we safely sweep all its neighbors. We call this strategy neighbor sweep. Additionally, we maintain a deposit value for each vertex, and once we finish testing or sweep a vertex, we increase the deposit values for its neighbors. If the deposit value of a vertex satisfies certain conditions, such vertex can also be swept.
- *Group Sweep.* We introduce a method to divide vertices in a graph into disjoint groups. If a vertex in a group has certain properties, vertices in the whole group can be swept. We call this strategy group sweep. Moreover, we maintain a group deposit value for each group. Once we test or sweep a vertex in the group, we increase the corresponding group deposit value. If the group deposit value satisfies certain conditions, vertices in the whole group can also be swept.

Even though these two strategies are studied independently, they can be used together and boost the effectiveness of each other. With these two sweep strategies, we significantly reduce the number of local connectivity testings in the algorithm.

Contributions. We summarize the main contributions below.

- *A polynomial time algorithm based on overlapped graph partition.* Given a graph G and an integer k, we prove that the sum of the number of overlapped partitions and the number of k-VCCs is less than number of vertices in G, which indicates a polynomial time algorithm to enumerate all k-VCCs in G.
- *Two effective pruning strategies.* We design two pruning strategies, namely neighbor sweep and group sweep, to largely reduce the number of local connectivity testings and thus significantly speed up the algorithm.
- *Extensive performance studies.* We conduct extensive performance studies on 10 large real-world graphs to demonstrate the efficiency of our proposed algorithms.

Outline. The rest of this paper is organized as follows. Section II formally defines the problem. Section III introduces an existing framework to compute all k-VCCs in a given graph. Section IV gives a basic implementation of the framework and analyzes the time complexity of the algorithm. Section V introduces several strategies to speed up the algorithm. Section VI evaluates the model and algorithms using extensive experiments. Section VII reviews related works and Section VIII concludes the paper. Note that due to the space limitation, we omit the proof for some lemmas and theorems.

II. Preliminary

A. Problem Statement

In this paper, we consider an undirected and unweighted graph $G(V, E)$, where V is the set of vertices and E is the set of edges. The number of vertices and the number of edges are denoted by $n = |V|$ and $m = |E|$ respectively. We denote the neighbor set of a vertex u by $N(u)$, i.e., $N(u) = \{v \in V | (u, v) \in E\}$, and the degree of u by $d(u) = |N(u)|$. Given two graphs g and g', we use $g \subseteq g'$ to denote that g is a subgraph of g'. Given a set of vertices V_s, the induced subgraph $G[V_s]$ is a subgraph of G such that $G[V_s] = (V_s, \{(u, v) \in E | u, v \in V_s\})$. For any two subgraphs g and g' of G, we use $g \cup g'$ to denote the union of g and g', i.e., $g \cup g' = (V(g) \cup V(g'), E(g) \cup E(g'))$. Before stating the problem, we give some basic definitions as follows.

Definition 1. (Vertex Connectivity) *The vertex connectivity of a graph G, denoted by $\kappa(G)$, is defined as the*

Algorithm 1 KVCC-ENUM(G, k)

```
Input: a graph $G$ and an integer $k$;
Output: all $k$-vertex connected components;
1:  $VCC_k(G) \leftarrow \emptyset$;
2:  while $\exists u : d(u) < k$ do remove $u$ and incident edges;
3:  identify connected components $\mathcal{G} = \{G_1, G_2, ..., G_t\}$ in $G$;
4:  for all connected component $G_i \in \mathcal{G}$ do
5:      $\mathcal{S} \leftarrow$ GLOBAL-CUT$(G_i, k)$;
6:      if $\mathcal{S} = \emptyset$ then
7:          $VCC_k(G) \leftarrow VCC_k(G) \cup \{G_i\}$;
8:      else
9:          $\mathcal{G}_i \leftarrow$ OVERLAP-PARTITION$(G_i, \mathcal{S})$;
10:         for all $G_i^j \in \mathcal{G}_i$ do
11:             $VCC_k(G) \leftarrow VCC_k(G) \cup$ KVCC-ENUM$(G_i^j, k)$;
12: return $VCC_k(G)$;
13: Procedure OVERLAP-PARTITION(Graph $G$, Vertex Cut $\mathcal{S}$)
14: $\mathcal{G} \leftarrow \emptyset$;
15: $G' \leftarrow G[V(G) \setminus S]$;
16: for all connected component $G_i'$ in $G'$ do
17:     $\mathcal{G} \leftarrow \mathcal{G} \cup \{G[V(G_i') \cup \mathcal{S}]\}$;
18: return $\mathcal{G}$;
```

minimum number of vertices whose removal results in either a disconnected graph or a trivial graph (a single-vertex graph).

DEFINITION 2. (K-VERTEX CONNECTED) *A graph G is k-vertex connected if: 1) $|V(G)| > k$; and 2) remaining graph is still connected after removing any $(k-1)$ vertices. That is, $\kappa(G) \geq k$.*

We use the term *k-connected* for short when the context is clear. It is easy to see that any nontrivial connected graph is at least 1-connected. Based on Definition 2, we define the *k-Vertex Connected Component* (k-VCC) as follows.

DEFINITION 3. (K-VERTEX CONNECTED COMPONENT) *Given a graph G, a subgraph g is a k-vertex connected component (k-VCC) of G if: 1) g is k-vertex connected; and 2) g is maximal. i.e., $\nexists g' \subseteq G$, s.t. $\kappa(g') \geq k$, $g \subsetneq g'$.*

Problem Definition. Given a graph G and an integer k, we study the problem of computing all k-VCCs in G.

III. PARTITION-BASED FRAMEWORK

In this section, we introduce the partition-based framework for computing all k-VCCs in a given graph G. For the ease of presentation, we named it KVCC-ENUM. Before introducing the details, we define the *vertex cut*.

DEFINITION 4. (VERTEX CUT) *Given a connected graph G, a vertex subset $\mathcal{S} \subset V$ is a vertex cut if the removal of $\mathcal{S}$ from G results in a disconnected graph.*

According to Definition 4, the vertex cut may not be unique for a given graph G, and the vertex connectivity is the cardinality of the minimum vertex cut. For a complete graph, there is no vertex cut since any two vertices are adjacent. The size of a vertex cut is the number of vertices in the cut. Similarly, the edge cut is a set of edges whose removal results in a disconnected graph. In the rest of this paper, we use the term cut for short to represent the vertex cut when the context is clear.

The Framework. Given a graph G, the general idea of the framework KVCC-ENUM is given as follows. If G is k-connected, G is a k-VCC. Otherwise, there exists a qualified cut $\mathcal{S}$ whose size is less than k. In this case, we compute such vertex cut and partition the graph G. The partition procedure is repeated until each remaining subgraph is a k-VCC. Given that a k-VCC must be contained in a k-core (a graph with the minimum degree no smaller than k [11]) [9], the k-core is computed in advance to reduce the graph size.

We arrange the pseudocode of KVCC-ENUM in Algorithm 1. In line 2, it computes k-core by iteratively removing the vertices whose degree is less than k. Then it identifies connected components of the input graph G. For each connected component G_i (line 4), KVCC-ENUM computes a cut of G_i by invoking GLOBAL-CUT (line 5). Here, we only need to find a cut with fewer than k vertices instead of a minimum cut. The detailed implementation of GLOBAL-CUT will be introduced later. If there is no such cut, that means G_i is k-connected and we add it to the result list $VCC_k(G)$ (line 6-7). Otherwise, the graph is partitioned into overlapped subgraphs using the cut $\mathcal{S}$ by invoking OVERLAP-PARTITION (line 9). KVCC-ENUM recursively cuts each of other subgraphs (line 11) until all remaining subgraphs are k-VCCs. Next, we introduce the subroutine OVERLAP_PARTITION.

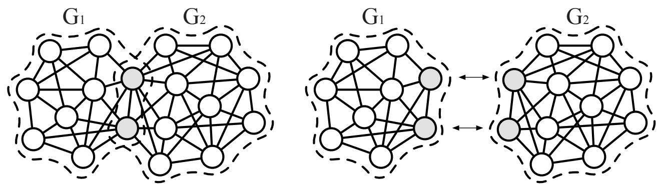


Fig. 2: An example of overlapped graph partition.

Overlapped Graph Partition. We partition a graph G into overlapped subgraphs using a cut $\mathcal{S}$. Subroutine OVERLAP-PARTITION is shown in line 13-18 of Algorithm 1. We add the cut $\mathcal{S}$ into each connected component G_i' of G' and return the induced subgraph $G[V(G_i') \cup S]$ as the partitioned subgraph (line 17-18). An example is given in Fig. 2. Given $k = 3$, we compute two 3-VCCs, G_1 and G_2, by duplicating the two cut vertices and their inner edges

IV. BASIC SOLUTION

In Algorithm 1, a crucial part is computing a vertex cut of G by invoking GLOBAL-CUT. In this section, we introduce an existing implementation for GLOBAL-CUT [15], [16], [17], [20], and then we explore optimization strategies to accelerate the computation of the vertex cut in Section V.

A. Vertex Cut Computation

We first give some necessary definitions before introducing the basic implementation of GLOBAL-CUT.

DEFINITION 5. (MINIMUM u-v CUT) *A vertex cut $\mathcal{S}$ is a u-v cut if u and v are in disjoint subsets after removing $\mathcal{S}$, and it is a minimum u-v cut if its size is no larger than that of other u-v cuts.*

DEFINITION 6. (LOCAL CONNECTIVITY) *Given a graph G, the local connectivity of two vertices u and v, denoted by $\kappa(u, v, G)$, is defined as the size of the minimum u-v cut. $\kappa(u, v, G) = +\infty$ if no such cut exists.*

Algorithm 2 GLOBAL-CUT(G, k)

```
Input: a graph $G$ and an integer $k$;
Output: a vertex cut with fewer than $k$ vertices;
1: compute a sparse certification $\mathcal{SC}$ of $G$;
2: select a source vertex $u$ with the minimum degree;
3: construct the directed flow graph $\overline{\mathcal{SC}}$ of $\mathcal{SC}$;
4: for all $v \in V$ do
5:     $\mathcal{S} \leftarrow$ LOC-CUT$(u, v, \overline{\mathcal{SC}}, \mathcal{SC})$;
6:     if $\mathcal{S} \neq \emptyset$ then return $\mathcal{S}$;
7: for all $v_a \in N(u)$ do
8:     for all $v_b \in N(u)$ do
9:         $\mathcal{S} \leftarrow$ LOC-CUT$(v_a, v_b, \overline{\mathcal{SC}}, \mathcal{SC})$;
10:        if $\mathcal{S} \neq \emptyset$ then return $\mathcal{S}$;
11: return $\emptyset$;
12: Procedure LOC-CUT$(u, v, \overline{G}, G)$
13: if $v \in N(u)$ or $v = u$ then return $\emptyset$;
14: $\lambda \leftarrow$ calculate the maximum flow from $u$ to $v$ in $\overline{G}$;
15: if $\lambda \geq k$ then return $\emptyset$;
16: compute the minimum edge cut in $\overline{G}$;
17: return the corresponding vertex cut in $G$;
```

Based on Definition 6, we define two local k connectivity relations as follows:

- $u \equiv_G^k v$: The local connectivity between u and v is no less than k in graph G, i.e., $\kappa(u, v, G) \geq k$.
- $u \not\equiv_G^k v$: The local connectivity between u and v is less than k in graph G, i.e., $\kappa(u, v, G) < k$.

We omit the index G when the context is clear. Once $u \equiv^k v$, we say u and v is k-local connected. Obviously, $u \equiv^k v$ is equivalent to $v \equiv^k u$ and the following lemma holds.

LEMMA 1. *$u \equiv^k v$ if $(u, v) \in E$.*

The GLOBAL-CUT Algorithm. The pseudocode of GLOBAL-CUT is given in Algorithm 2. Given a graph G, we assume that G contains a vertex cut $\mathcal{S}$ such that $|\mathcal{S}| < k$. Consider an arbitrary source vertex u. There are only two cases: (i) $u \notin \mathcal{S}$ and (ii) $u \in \mathcal{S}$. The general idea of algorithm GLOBAL-CUT is considering these two cases. In the first phase (line 4-6), we select a vertex u and test the local connectivity between u and all other vertices v in G by invoking LOC-CUT. We have either (a) $u \in \mathcal{S}$ or (b) G is k-connected if each local connectivity is no less than k. In the second phase (line 7-10), we consider the case $u \in \mathcal{S}$ and test the local connectivity between any two neighbors of u based on Lemma 2 [20].

LEMMA 2. *Given a non-k-vertex connected graph G and a vertex $u \in \mathcal{S}$ where $\mathcal{S}$ is a minimal vertex cut and $|\mathcal{S}| < k$, there exist $v, v' \in N(u)$ such that $v \not\equiv^k v'$.*

To test the connectivity of two vertices, we need to transform the original graph into a directed flow graph with $2n$ vertices and $n + 2m$ edges and the capacities of all edges are 1 (line 3). The local connectivity between u and v is equal to the max-flow between them on directed flow graph. LOC-CUT returns the corresponding vertex cut if the max-flow is less than k (line 15-17). More details about directed flow graph can be found in [21].

Sparse Certificate. Based on the basic GLOBAL-CUT, we adopt an optimization in Algorithm 2; that is computing a sparse certificate [22] of the original graph (line 1). We introduce the definition of sparse certificate as follows.

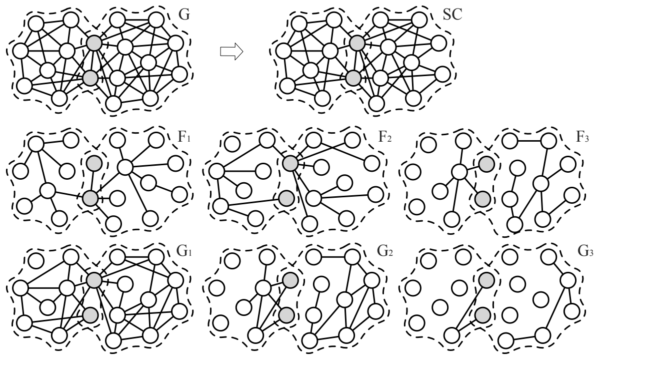


Fig. 3: The sparse certificate of G with $k = 3$

DEFINITION 7. (CERTIFICATE) *A certificate for the k-vertex connectivity of G is a subset E' of E such that the subgraph (V, E') is k-vertex connected if and only if G is k-vertex connected.*

DEFINITION 8. (SPARSE CERTIFICATE) *A certificate for k-vertex connectivity of G is called sparse if it has $O(k \cdot n)$ edges.*

From the definitions, we can see that a sparse certificate is equivalent to the original graph w.r.t k-vertex connectivity. Meanwhile, it can bound the edge size. We will show that the sparse certificate can not only be used to reduce the graph size, but also used to further reduce local connectivity testings in Section V. The sparse certificate is computed according to the following theorem [22].

THEOREM 1. *Let $G(V, E)$ be an undirected graph and let n denote the number of vertices. Let k be a positive integer. For $i = 1, 2, ..., k$, let E_i be the edge set of a scan first search forest F_i in the graph $G_{i-1} = (V, E - (E_1 \cup E_2 \cup ... \cup E_{i-1}))$, where $G_0 = G$. Then $E_1 \cup E_2 \cup ... \cup E_k$ is a certificate for the k-vertex connectivity of G, and this certificate has at most $k \times (n - 1)$ edges.*

The detail about the scan first search can be found in [23], [22]. For the simplicity, we can replace it by breadth first search here, since the breadth first search is a special case of scan first search. Based on Theorem 1, we can construct a sparse certificate of G using breadth first search k times, each of which creates a scan first search forest F_i. An example of constructing sparse certificate is given as follows.

EXAMPLE 1. *Fig. 3 presents construction of a sparse certificate for G. Let $k = 3$. For $i \in \{1, 2, 3\}$, F_i denotes the scan first search forest computed from G_{i-1}. G_i is computed by removing the edges in F_i from G_{i-1}. G_0 is the input graph G. The sparse certificate SC is shown on the right side of G with $SC = F_1 \cup F_2 \cup F_3$. All removed edges are shown in G_3.*

B. Algorithm Analysis

Given that there does not exist any theoretical analysis for the running time of the overall framework KVCC-ENUM [17], we prove that KVCC-ENUM terminates in polynomial time in this section. In the directed flow graph, all edge capacities are equal to 1 and every vertex either has a single edge emanating from it or has a single edge entering it. For this kind of graph, the time complexity for computing the maximum flow is $O(n^{0.5}m)$ [24]. Note that we do not need to calculate the exact flow value in the algorithm. Once the flow value reaches k, we know that local connectivity between any two given vertices is at least k and we can terminate the maximum flow algorithm. The time complexity for the flow computation is $O(\min(n^{0.5}, k) \cdot m)$. Given a flow value and corresponding residual network, we can perform a depth first search to find the cut. It costs $O(m+n)$ time. As a result, we have the following lemma:

LEMMA 3. *The time complexity of algorithm* LOC-CUT *is* $O(\min(n^{0.5}, k) \cdot m)$.

Next we discuss the time complexity of GLOBAL-CUT. The construction of both sparse certificate and directed flow graph costs $O(m+n)$ time. Let δ denote the minimum degree in the input graph. We can easily get following lemma.

LEMMA 4. GLOBAL-CUT *invokes* LOC-CUT $O(n+\delta^2)$ *times in the worst case.*

Below we discuss the time complexity of the entire algorithm KVCC-ENUM. KVCC-ENUM iteratively removes vertices with a degree of less than k (line 2). This costs $O(m+n)$ time. Identifying all connected components (line 3) can be performed by adopting a depth first search in $O(m+n)$ time. To study the number of times that GLOBAL-CUT is invoked, we first give the following lemmas.

LEMMA 5. *For each connected component C computed by overlapped partition in Algorithm 1,* $|V(C)| \geq k+1$.

PROOF. *Let $\mathcal{S}$ denote a vertex cut in an overlapped partition. C is one of the connected components computed in this partition. Let H denote the vertex set of all vertices in $V(C)$ but not in $\mathcal{S}$, i.e., $H = \{u | u \in V(C), u \notin \mathcal{S}\}$. We have $H \neq \emptyset$. Note that each vertex in the graph has a degree of at least k in G (line 5 in Algorithm 1). There exist at least k neighbors for each vertex u in H and therefore for each neighbor v of u we have $v \in C$ according to Lemma 1. Thus, we have* $|V(C)| \geq k+1$.

LEMMA 6. *The total number of overlapped partitions during the algorithm* KVCC-ENUM *is no greater than* $\frac{n-k-1}{2}$.

PROOF. *Suppose that λ is the total number of overlapped partitions during the whole algorithm* KVCC-ENUM. *This generates at least $\lambda+1$ connected components. We know from Lemma 5 that each connected component contains at least $k+1$ vertices. Thus, we have at least $(\lambda+1)(k+1)$ vertices in total. On the other hand, we increase at most $k-1$ vertices in each subgraph derived by an overlapped partition. Thus, at most $\lambda(k-1)$ vertices are added. We have $(\lambda+1)(k+1) \leq n+\lambda(k-1)$. Rearranging the formula, we have $\lambda \leq \frac{n-k-1}{2}$.*

We then prove the upper bound for the number of k-VCCs.

THEOREM 2. *Given a graph G and an integer k, the number of k-VCCs is bounded by $\frac{n}{2}$, i.e.,* $|VCC_k(G)| < \frac{|V(G)|}{2}$.

PROOF. *Similar to the proof of Lemma 6, let λ be the number of overlapped partitions in the whole algorithm* KVCC-ENUM. *At most $\lambda(k-1)$ vertices are added. Let σ be the number of connected components computed in all partitions. We have $\sigma > \lambda$. Each connected component contains at least $k+1$ vertices according to Lemma 5. Note that each connected component is either a k-VCC or a graph that does not contain any k-VCC. Otherwise, the connected component will be further partitioned. Let x be the number of k-VCCs and y be the number of connected components that do not contain any k-VCC, i.e., $x+y=\sigma$. We know that a k-VCC contains at least $k+1$ vertices. Thus there are at least $x(k+1)+y(k+1)$ vertices after finishing all partitions. We have $x(k+1)+y(k+1) \leq n+\lambda(k-1)$. Since $\lambda < \sigma$ and $\sigma = x+y$, we rearrange the formula as $x(k+1)+y(k+1) < n+x(k-1)+y(k-1)$. Therefore, we have $x < \frac{n}{2}$.*

THEOREM 3. *The total time complexity of* KVCC-ENUM *is* $O(\min(n^{0.5},\, k) \cdot m \cdot (n+\delta^2) \cdot n)$.

PROOF. *The total time complexity of* KVCC-ENUM *is dependent on the number of times* GLOBAL-CUT *is invoked. Suppose* GLOBAL-CUT *is invoked p times during the whole* KVCC-ENUM *algorithm, the number of overlapped partitions during the whole* KVCC-ENUM *algorithm is p_1 and the total number of k-VCCs is p_2. It is easy to see that $p = p_1 + p_2$. From Lemma 6, we know that $p_1 \leq \frac{n-k-1}{2} < \frac{n}{2}$. From Theorem 2, we know that $p_2 < \frac{n}{2}$. Therefore, we have $p = p_1 + p_2 < n$. According to Lemma 3 and Lemma 4, the theorem holds.*

Discussion. Theorem 3 shows that all k-VCCs can be enumerated in polynomial time. Although the time complexity is still high, it performs much better in practice. Note that the time complexity is the product of three parts:

- The first part $O(\min(n^{1/2},\, k) \cdot m)$ is the time complexity for LOC-CUT to test whether there exists a vertex cut of size smaller than k. In practice, the graph to be tested is much smaller than the original graph G since (1) The graph to be tested has been pruned using the k-core technique and sparse certification technique. (2) Due to the graph partition scheme, the input graph is partitioned into many smaller graphs.
- The second part $O(n+\delta^2)$ is the number of times such that LOC-CUT (local connectivity testing) is invoked by the algorithm GLOBAL-CUT. We will discuss how to significantly reduce the number of local connectivity testings in Section V.
- The third part $O(n)$ is the number of times GLOBAL-CUT is invoked. In practice, the number can be significantly reduced since the number of k-VCCs is usually much smaller than $\frac{n}{2}$.

V. SEARCH REDUCTION

In the previous section, we introduce a basic solution for k-VCC enumeration. Recall that in the worst case, we need to test local connectivity between the source vertex u and all other vertices in G in GLOBAL-CUT, and we also need to test local connectivity for every pair of neighbors of u. Therefore, the key to improving the algorithm is to reduce the number of local connectivity testings (LOC-CUT). In this section, we propose several techniques to avoid unnecessary testings. We can avoid testing local connectivity of a vertex pair (u, v) if we can guarantee that $u \equiv^k v$. We call such operation a sweep operation. Below, we introduce two ways to efficiently prune unnecessary testings, namely neighbor sweep and group sweep, in Section V-A and Section V-B respectively.

A. Neighbor Sweep

In this section, we propose a neighbor sweep strategy to prune unnecessary local connectivity testings (LOC-CUT) in the first phase of GLOBAL-CUT. Generally, given a source vertex u, we aim to skip testing the local connectivity of (u, v) according to the information of the neighbors of v. Below, we explore two neighbor sweep strategies, namely neighbor sweep using side-vertex and neighbor sweep using vertex deposit.

Neighbor Sweep using Side-Vertex. We first define *side-vertex* as follows.

DEFINITION 9. (SIDE-VERTEX) *Given a graph G and an integer k, a vertex u is called a side-vertex if there does not exist a vertex cut $\mathcal{S}$ such that $|\mathcal{S}| < k$ and $u \in \mathcal{S}$.*

Based on Definition 9, we give the following lemma to show the transitive property regarding the local k connectivity relation $\equiv^k$.

LEMMA 7. *Given a graph G and an integer k, suppose $a \equiv^k b$ and $b \equiv^k c$, we have $a \equiv^k c$ if b is a side-vertex.*

A wise way to use the transitive property of the local connectivity relation in Lemma 7 can largely reduce the number of unnecessary testings. Consider a selected source vertex u in algorithm GLOBAL-CUT. We assume that LOC-CUT (line 5) returns $\emptyset$ for a vertex v, i.e., $u \equiv^k v$. We know from Lemma 7 that the vertex pair (u, w) can be skipped for local connectivity testing if (i) $v \equiv^k w$ and (ii) v is a side-vertex. For condition (i), we can use a simple necessary condition according to Lemma 1, that is, for any vertices v and w, $v \equiv^k w$ if $(v, w) \in E$. In the following, we focus on condition (ii) and look for necessary conditions to efficiently check whether a vertex is a side-vertex.

Side-Vertex Detection. To check whether a vertex is a side-vertex, we can easily derive the following lemma based on Definition 9.

LEMMA 8. *Given a graph G, a vertex u is a side-vertex if and only if $\forall v, v' \in N(u)$, $v \equiv^k v'$.*

Recall that two vertices are k-local connected if they are neighbors of each other. For the k-local connectivity of non-connected vertices, we give another necessary condition below.

LEMMA 9. *Given two vertices u and v, $u \equiv^k v$ if $|N(u) \cap N(v)| \geq k$.*

Combining Lemma 8 and Lemma 9, we derive the following necessary condition to check whether a vertex is a side-vertex.

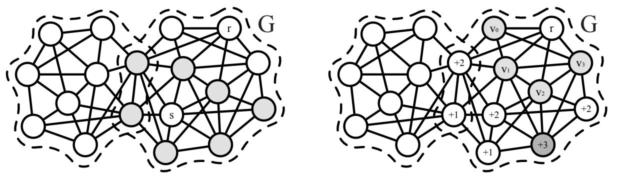
THEOREM 4. *A vertex u is a side-vertex if $\forall v, v' \in N(u)$, either $(v, v') \in E$ or $|N(v) \cap N(v')| \geq k$.*

DEFINITION 10. (STRONG SIDE-VERTEX) *A vertex u is called a strong side-vertex if it satisfies the conditions in Theorem 4.*

Based on Definition 10, we define the following sweep rule.

(Neighbor Sweep Rule 1) *Given a graph G and an integer k, let u be a selected source vertex in algorithm* GLOBAL-CUT *and v be a strong side-vertex in the graph. We can skip the local connectivity testings of all pairs of (u, w) if we have $u \equiv^k v$ and $w \in N(v)$.*

We give an example for *neighbor sweep rule 1* below.



(a) A strong side-vertex s (b) Vertex deposit

Fig. 4: Strong side-vertex and vertex deposit when $k = 3$

EXAMPLE 2. *Fig. 4 (a) presents a strong side-vertex s in G given $k = 3$. Assume that r is the source vertex. Any two neighbors of s are either connected by an edge or have at least 3 common neighbors. If $r \equiv^k s$, we safely sweep all neighbors of s, which are marked by the gray color.*

Next, we discuss how to efficiently detect and maintain all strong side-vertices.

Strong Side-Vertex Computation. Following Theorem 4, we can compute all strong side-vertices v in advance and sweep all neighbors of v once v is k connected with the source vertex (line 5 in GLOBAL-CUT). We derive the following lemma.

LEMMA 10. *The time complexity of computing all strong side-vertices in graph G is $O(\sum_{u \in V(G)} d(u)^2)$.*

After computing all strong side-vertices for the original graph G, we do not need to recompute strong side-vertices in the partitioned graph from scratch. Instead, we can reduce the number of strong side-vertex checks by making use of the already computed strong side-vertices in G. We efficiently detect non-strong side-vertices and strong side-vertices respectively based on Lemma 11 and Lemma 12.

LEMMA 11. *Let G be a graph and G_i be one of the graphs derived by partitioning G using* OVERLAP-PARTITION *in Algorithm 1, a vertex is a strong side-vertex in G if it is a strong side-vertex in G_i.*

From Lemma 11, we know that a vertex is not a strong side-vertex in G_i if it is not a strong side-vertex in G. This property allows us checking limited number of vertices in G_i, which is the set of strong side-vertices in G.

LEMMA 12. *Let G be a graph, G_i be one of the graphs derived by partitioning G using* OVERLAP-PARTITION *in Algorithm 1, and $\mathcal{S}$ is a vertex cut of G, for any vertex $v \in V(G_i)$, if v is a strong side-vertex in G and $N(v) \cap \mathcal{S} = \emptyset$, then v is also a strong side-vertex in G_i.*

Based on Lemma 11 and Lemma 12, in a graph G_i partitioned from graph G by vertex cut $\mathcal{S}$, we can reduce the scope of strong side-vertex checks from the vertices in the whole graph G_i to the vertices u satisfying following two conditions simultaneously:

- u is a strong side-vertex in G; and
- $N(u) \cap \mathcal{S} \neq \emptyset$.

Neighbor Sweep using Vertex Deposit. The strong side-vertex strategy heavily relies on the number of strong side-vertices. Next, we investigate a new strategy called vertex deposit, to further sweep vertices based on neighbor information. We first give the following lemma:

LEMMA 13. *Given a source vertex u, for any vertex $v \in V(G)$, we have $u \equiv^k v$ if there exist k vertices $w_1, w_2, \dots, w_k$ such that $u \equiv^k w_i$ and $w_i \in N(v)$ for any $1 \leq i \leq k$.*

PROOF. *We prove it by contradiction. Assume that $u \not\equiv^k v$. There exists a vertex cut $\mathcal{S}$ with $k-1$ or fewer vertices between u and v. For any $w_i (1 \leq i \leq k)$, we have $w_i \equiv^k v$ since $w_i \in N(v)$ (Lemma 1) and we also have $w_i \equiv^k u$. Since $u \not\equiv^k v$, w_i cannot satisfy both $w_i \equiv^k u$ and $w_i \equiv^k v$ unless $w_i \in \mathcal{S}$. Therefore, we compute a cut $\mathcal{S}$ with at least k vertices w_1, w_2, ..., w_k. This contradicts $|\mathcal{S}| < k$.*

Based on Lemma 13, given a source vertex u, once we find a vertex v with at least k neighbors w_i satisfying $u \equiv^k w_i$, we have $u \equiv^k v$ without testing their local connectivity. To efficiently detect such vertices, we define *deposit* of each vertex as follows.

DEFINITION 11. *(Vertex Deposit) Given a source vertex u, the deposit for each vertex v, denoted by $deposit_u(v)$, is the number of neighbors w of v such that $u \equiv^k w$.*

According to Definition 11, suppose u is the source vertex and for each vertex v, $deposit_u(v)$ is a dynamic value depending on the number of processed vertex pairs. The vertex deposit for each vertex is initialized by 0. Once we know $w \equiv^k u$ for a vertex w, we can increase $deposit_u(v)$ for each $v \in N(w)$ by 1. We deduce the following theorem according to Lemma 13 and Definition 11.

THEOREM 5. *Given a source vertex u, for any vertex v, we have $u \equiv^k v$ if $deposit_u(v) \geq k$.*

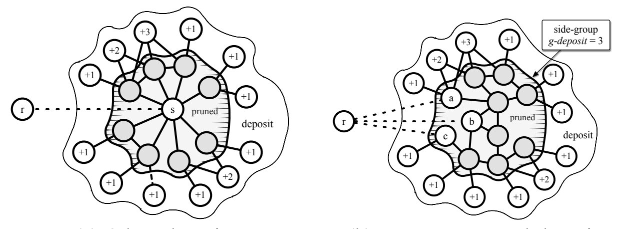
Based on Theorem 5, we derive our second neighbor sweep rule.

(Neighbor Sweep Rule 2) *Given a selected source vertex u, we can skip the local connectivity testing of pair (u, v) if $deposit_u(v) \geq k$.*

EXAMPLE 3. *Fig. 4 (b) gives an example of vertex deposit strategy. Given the graph G and parameter $k = 3$, let vertex r be the selected source vertex. We have v_0, v_1, v_2 and v_3 are local k-connected with vertex r, i.e., $r \equiv^k v_i, i \in 0, 1, 2, 3$, since v_0, v_1, v_2 and v_3 are neighbors of r. We deposit once for the neighbors of each tested vertex. The deposit value for each influenced vertices is given in the figure. We mark the vertices with deposit no less than 3 by dark gray. The local connectivity testing between r and such a vertex can be skipped.*

To increase the deposit of a vertex v, we only need any neighbor of v is local k-connected with the source vertex u. We can also use vertex deposit strategy when processing strong side-vertices. Given a source vertex u and a strong side-vertex v, we sweep all $w \in N(v)$ if $u \equiv^k v$. Then we increase the deposit for each non-swept vertex $w' \in N(w)$. In other words, for a strong side-vertex, we can possibly sweep its 2-hop neighbors by combining the two neighbor sweep strategies. An example is given below.

EXAMPLE 4. *Fig. 5 (a) shows the process for a strong side-vertex s. Given a source vertex r, assume s is a strong side-vertex and $r \equiv^k s$. All neighbors of s are swept and all 2-hop neighbors of s increase their deposits accordingly. The increased deposit for each vertex is shown in the figure.*



(a) 2-hop deposit (b) group sweep and deposit

Fig. 5: Increasing deposit with neighbor and group sweep

B. Group Sweep

The neighbor sweep strategy only prune unnecessary local connectivity testings in the first phase of GLOBAL-CUT by using the neighborhood information. In this subsection, we introduce a new pruning strategy namely group sweep, which can prune unnecessary local connectivity testings in a batch manner. In group sweep, we do not limit the swept vertices to the neighbors of certain vertices. Specifically, we aim to partition vertices into vertex groups and sweep a whole group when it satisfies certain conditions. In addition, our group sweep strategy can also be applied to reduce unnecessary local connectivity testings in both phases of GLOBAL-CUT.

First, we define a new relation regarding a vertex u and a set of vertices C as follows.

$$u \equiv^k C\text{: For all vertices } v \in C, u \equiv^k v.$$

Given a source vertex u and a side-vertex v, we assume $u \equiv^k v$. According to the transitive relation in Lemma 7, we skip testing the pairs of vertices u and w if $w \equiv^k v$. In our neighbor sweep strategy, we select all neighbors of v as such vertices w, i.e., $u \equiv^k N(v)$. To sweep more vertices each time, we define the side-group.

DEFINITION 12. (SIDE-GROUP) *Given a graph G and an integer k, a vertex set $\mathcal{CC}$ is a side-group if $\forall u, v \in \mathcal{CC}, u \equiv^k v$.*

Next, we introduce how to construct the side-groups, and then discuss our group sweep rules.

Side-Group Construction. Section IV-A introduces sparse certificate to bound the graph size. Let F_i and G_i be the notations defined in Theorem 1. Assume G is not k-connected and there exists a vertex cut $\mathcal{S}$ with $|\mathcal{S}| < k$. We introduce the following lemma [22].

LEMMA 14. *F_k does not contain a tree path whose two end points are in different connected components of $G - \mathcal{S}$.*

Here, a tree path is a sequence of tree nodes $u_0, u_1, ..., u_t$, in which there is a tree edge connecting u_i and u_{i+1} for each $0 \leq i < t$. Based on Lemma 14, we deduce the following theorem.

THEOREM 6. *Let $\mathcal{CC}$ denote the vertex set of any connected component in F_k. $\mathcal{CC}$ is a side-group.*

EXAMPLE 5. *Review the construction of a sparse certificate in Fig. 3. Given $k = 3$, two connected components with more than one vertex are computed in F_3. The number of vertices in the two connected components are 6 and 9 respectively.*

Each of them is a side-group and any two vertices in the same connected component is local 3*-connected. Note that the connected component with* 6 *vertices contains two vertices in the vertex cut as marked by gray.*

We denote all side-groups as $\mathcal{CS} = \{\mathcal{CC}_1, \mathcal{CC}_2, \ldots, \mathcal{CC}_t\}$. According to Theorem 6, $\mathcal{CS}$ can be easily computed as a by-product of the sparse certificate. With $\mathcal{CS}$, according to the transitive relation in Lemma 7, we can easily infer the following pruning rule.

(Group Sweep Rule 1) *Let u be the source vertex in the algorithm* GLOBAL-CUT*, given a side-group $\mathcal{CC}$, if there exists a strong side-vertex $v \in \mathcal{CC}$ such that $u \equiv^k v$, we can skip the local connectivity testings of vertex pairs (u, w) for all $w \in \mathcal{CC} - \{v\}$.*

The above rule relies on the successful detection of a strong side-vertex in each side-group. In the following, we further introduce a deposit based scheme to handle the scenario that no strong side-vertex exists in the side-group.

Group Deposit. Similar with the vertex deposit strategy, the group deposit strategy aims to deposit the values in a group level. To show our group deposit scheme, we first introduce the following lemma.

LEMMA 15. *Given a source vertex u, an integer k, and a side-group $\mathcal{CC}$, we have $u \equiv^k \mathcal{CC}$ if $|\{v|v \in \mathcal{CC}, u \equiv^k v\}| \geq k$.*

PROOF. *We prove it by contradiction. Assume that there exists a vertex w in $\mathcal{CC}$ such that $u \not\equiv^k w$. A vertex cut $\mathcal{S}$ exists with $|\mathcal{S}| < k$. Let $v_0, v_1, ..., v_{k-1}$ be the k vertices in $\mathcal{CC}$ such that $u \equiv^k v_i, 0 \leq i \leq k-1$. We have $w \equiv^k v_i$ based on Definition 12. Each v_i must belong to $\mathcal{S}$ since $u \not\equiv^k w$. As a result, the size of $\mathcal{S}$ is at least k. This contradicts $|\mathcal{S}| < k$.*

Based on Lemma 15, given a source vertex u, once we find a side-group $\mathcal{CC}$ with at least k vertices v with $u \equiv^k v$, we can get $u \equiv^k \mathcal{CC}$ without testing the local connectivity from u to other vertices in $\mathcal{CC}$. To efficiently detect such side-groups $\mathcal{CC}$, we define the group deposit of a side-group $\mathcal{CC}$ as follows.

DEFINITION 13. *(Group Deposit) Given a source vertex u, the group deposit for each side-group $\mathcal{CC}$, denoted by $g\text{-}deposit_u(\mathcal{CC})$, is the number of vertices $v \in \mathcal{CC}$ such that $u \equiv^k v$.*

According to Definition 13, for each side-group $\mathcal{CC} \in \mathcal{CS}$, $g\text{-}deposit_u(\mathcal{CC})$ is a dynamic value depending on the already processed vertex pairs. The group deposit for each side-group $\mathcal{CC}$ is initialized by 0. Once $v \equiv^k u$ for a certain vertex $v \in \mathcal{CC}$, we can increase $g\text{-}deposit_u(\mathcal{CC})$ by 1. We infer the following theorem according to Lemma 15 and Definition 13.

THEOREM 7. *Given a source vertex u, for any side-group $\mathcal{CC} \in \mathcal{CS}$, we have $u \equiv^k \mathcal{CC}$ if $g\text{-}deposit_u(\mathcal{CC}) \geq k$.*

Next, we derive our second group sweep rule as follows.

(Group Sweep Rule 2) *Given a selected source vertex u, we can skip the local connectivity testings between u and vertices in $\mathcal{CC}$ if $g\text{-}deposit_u(\mathcal{CC}) \geq k$.*

Note that a group sweep operation can further trigger a neighbor sweep operation and vice versa, since both operations result in new local k-connected vertex pairs. We show an example below.

Algorithm 3 GLOBAL-CUT$^*(G, k)$

```
Input: a graph $G$ and an integer $k$;
Output: a vertex cut with size smaller than $k$;
1: compute a sparse certification $\mathcal{SC}$ of $G$ and collect all side-groups
   as $\mathcal{CS} = \{\mathcal{CC}_1, ..., \mathcal{CC}_t\}$;
2: construct the directed flow graph $\overline{\mathcal{SC}}$ of $\mathcal{SC}$;
3: $\mathcal{SV} \leftarrow$ compute all strong side vertices in $\mathcal{SC}$;
4: if $\mathcal{SV} = \emptyset$ then
5:     select a vertex $u$ with the minimum degree;
6: else
7:     randomly select a vertex $u$ from $\mathcal{SV}$;
8: for all $\mathcal{CC}_i$ in $\mathcal{CS}$: $g\text{-}deposit_u(\mathcal{CC}_i) \leftarrow 0$;
9: for all $v$ in $V$: $deposit_u(v) \leftarrow 0, pru(v) \leftarrow$ false;
10: SWEEP$(u, pru, deposit_u, g\text{-}deposit_u, \mathcal{CS})$;
11: for all $v \in V$ in non-ascending order of $dist(u, v, G)$ do
12:     if $pru(v) =$ true then continue;
13:     $\mathcal{S} \leftarrow$ LOC-CUT$(u, v, \overline{\mathcal{SC}}, \mathcal{SC})$;
14:     if $\mathcal{S} \neq \emptyset$ then return $\mathcal{S}$;
15:     SWEEP$(v, pru, deposit_u, g\text{-}deposit_u, \mathcal{CS})$;
16: if $u$ is not a strong side-vertex then
17:     for all $v_a \in N(u)$ do
18:         for all $v_b \in N(u)$ do
19:             if $v_a$ and $v_b$ are in the same $\mathcal{CC}_i$ then continue;
20:             $\mathcal{S} \leftarrow$ LOC-CUT$(u, v, \overline{\mathcal{SC}}, \mathcal{SC})$;
21:             if $\mathcal{S} \neq \emptyset$ then return $\mathcal{S}$;
22: return $\emptyset$;
```

EXAMPLE 6. *Fig. 5 (b) presents an example of group sweep. Suppose $k = 3$ and the gray area is a detected side-group. Given a source vertex r, assume that a, b, c are the tested vertices with $r \equiv^k a, r \equiv^k b$ and $r \equiv^k c$ respectively. According to Theorem 7, we can safely sweep all vertices in the same side-group. Also, we apply the vertex deposit strategy for neighbors outside the side-group. The increased value of deposit is shown on each vertex.*

Next we show that the side-groups can also be used to prune the local connectivity testings in the second phase of GLOBAL-CUT. Recall that in the second phase of GLOBAL-CUT, given a source vertex u, we need to test the local connectivity of every pair (v_a, v_b) of the neighbors of u. With side-groups, we can easily infer the following group sweep rule.

(Group Sweep Rule 3) *Let u be the source vertex, and v_a and v_b be two neighbors of u. If v_a and v_b belong to the same side-group, we have $v_a \equiv^k v_b$ and thus we do not need to test the local connectivity of (v_a, v_b) in the second phase of* GLOBAL-CUT.

The detailed implementation of the neighbor sweep and group sweep techniques is given in the following section.

C. The Overall Algorithm

In this section, we combine our pruning strategies and give the implementation of optimized algorithm GLOBAL-CUT*. The pseudocode is presented in Algorithm 3. We can replace GLOBAL-CUT with GLOBAL-CUT* in KVCC-ENUM to generate our final algorithm to compute all k-VCCs.

The GLOBAL-CUT* algorithm still follows the idea of GLOBAL-CUT. Given a source vertex u, phase 1 (line 8-15) considers the case that $u \notin \mathcal{S}$. Phase 2 (line 16-21) considers

Algorithm 4 SWEEP$(v, pru, deposit_u, g\text{-}deposit_u, \mathcal{CS})$

```
1: $pru(v) \leftarrow$ true;
2: for all $w \in N(v)$ s.t. $pru(w) =$ false do
3:     $deposit_u(w)$++;
4:     if $v$ is a strong side-vertex or $deposit_u(w) \geq k$ then
5:         SWEEP$(w, pru, deposit_u, g\text{-}deposit_u, \mathcal{CS})$;
6: if $v$ is contained in a $\mathcal{CC}_i$ and $\mathcal{CC}_i$ has not been processed then
7:     $g\text{-}deposit_u(\mathcal{CC}_i)$++;
8:     if $v$ is a strong side-vertex or $g\text{-}deposit_u(\mathcal{CC}_i) \geq k$ then
9:         mark $\mathcal{CC}_i$ as processed;
10:        for all $w \in \mathcal{CC}_i$ s.t. $pru(w) =$ false do
11:            SWEEP$(w, pru, deposit_u, g\text{-}deposit_u, \mathcal{CS})$
```

the case that $u \in \mathcal{S}$. If in both phases, the vertex cut $\mathcal{S}$ is not found, we simply return $\emptyset$ in line 22.

We compute the side-groups $\mathcal{CS}$ while computing the sparse certificate (line 1). We only consider the side-group whose size is larger than k, since the group can be swept only if at least k vertices in the group are swept according to Theorem 7. Then we compute all strong side-vertices, $\mathcal{SV}$ based on Theorem 4 (line 3). Here, the strong side-vertices are computed based on the method discussed in Section V-A. If $\mathcal{SV}$ is not empty, we can select one inside vertex as source vertex u and do not need to consider the phase 2, because u cannot be in any cut $\mathcal{S}$ with $|\mathcal{S}| < k$ in this case. Otherwise, we still select the source vertex u with the minimum degree (line 4-7).

In phase 1 (line 8-15), we initialize the group deposit for each side-group as 0 (line 8) Also, we initialize the local deposit for each vertex as 0 and pru for each vertex as *false* (line 9). Here, pru is used to mark whether a vertex can be swept. We first apply the sweeping rules on the source vertex by invoking SWEEP procedure (line 10). Intuitively, a vertex that is close to the source vertex u tends to be in the same k-VCC with u. In other words, a vertex v that is far away from u tends to be separated from u by a vertex cut $\mathcal{S}$. Therefore, we process vertices v in G according to the non-ascending order of $dist(u, v, G)$ (line 11). For each vertex v to be processed in phase 1, we skip it if $pru(v)$ is true (line 12). Otherwise, we test the local connectivity of u and v using LOC-CUT (line 13). If there is a cut $\mathcal{S}$ with size smaller than k, we simply return $\mathcal{S}$ (line 15). Otherwise, we invoke SWEEP procedure to sweep vertices using the sweep rules. We will introduce the SWEEP procedure in detail later.

In phase 2 (line 16-21), we check whether u is a strong side-vertex. If so, we skip phase 2. Otherwise, we perform pair-wise local connectivity testings for all vertices in $N(u)$. Here, we apply the *group sweep rule 3* and skip testing those pairs of vertices that are in the same side-group.

Procedure SWEEP**.** The procedure SWEEP is shown in Algorithm 4. To sweep a vertex v, we set $pru(v)$ to be true. This operation may result in neighbor sweep and group sweep of other vertices as follows.

- (Neighbor Sweep) We adopt the neighbor sweep rules in line 1-5. For all the neighbors w of v that have not been swept, we increase $deposit_u(w)$ by 1. Then we consider two cases. The first case is that v is a strong side-vertex. According to *neighbor sweep rule 1*, w can be swept since w is a neighbor of v. The second case is $deposit_u(w) \geq k$. According to *neighbor sweep rule 2*, w can be swept. In both cases, we invoke SWEEP to sweep w recursively.
- (Group Sweep) In line 6-11, we adopt the group sweep rules if v is contained in a side-group. We first increase $g\text{-}deposit_u(\mathcal{CC}_i)$ by 1 based on Definition 13. Then we consider two cases. The first case is that v is a strong side-vertex. According to *group sweep rule 1* in Section V-B, we can sweep all vertices in $\mathcal{CC}_i$. The second case is that $g\text{-}deposit_u(\mathcal{CC}_i) \geq k$. According to *group sweep rule 2*, we sweep all vertices in $\mathcal{CC}_i$. In both cases we recursively invoke SWEEP to sweep vertices(line 8-11).

Datasets	$\lvert V\rvert$	$\lvert E\rvert$	$\overline{d}$
ca-CondMat	23,133	93,497	8.08
ca-AstroPh	18,772	198,110	21.11
Stanford	281,903	2,312,497	16.41
cnr	325,557	3,216,152	19.76
DBLP	986,324	6,707,236	13.60
Web-BerkStan	685,230	7,600,595	22.18
as-Skitter	1,696,415	11,095,298	13.08
cit-Patents	3,774,768	16,518,948	8.75
LiveJournal	4,847,571	68,993,773	28.47
Webbase	118,142,155	1,019,903,190	17.27

TABLE I: Network Statistics

VI. Experiments

In this section, we experimentally evaluate the performance of our proposed algorithms. We use VCCE to denote the state-of-the-art solution for exactly computing k-VCCs, which invokes GLOBAL-CUT (Algorithm 2) to compute a vertex cut [15], [16], [17]. We use VCCE* to denote our final optimized algorithm with both neighbor sweep and group sweep strategies. In contrast to VCCE, VCCE* invokes GLOBAL-CUT* which is given in Algorithm 3.

All algorithms are implemented in C++ using gcc complier at -O3 optimization level. All the experiments are conducted under a Linux operating system running on a machine with an Intel Xeon 3.4GHz CPU, 32GB 1866MHz DDR3-RAM.

Datasets. We evaluate algorithms on 10 publicly available real-world networks, collaboration network of Arxiv Condensed Matter (ca-CondMat), collaboration network of Arxiv Astro Physics (ca-AstroPh), web graph of Stanford.edu (Stanford), web graph of Italian CNR domain (cnr), DBLP collaboration network (DBLP), web graph of Berkeley and Stanford (Web-BerkStan), Internet topology graph (as-Skitter), citation network among US Patents (cit-Patents), LiveJournal online social network (LiveJournal), and web graph form WebBase crawler (Webbase). The detailed statistics are shown in Table I. The networks are displayed in non-decreasing order regarding the number of edges. All networks and corresponding detailed description can be found in SNAP[4] and Webgraph[5].

Parameter Setting. Given a graph G, let $k_{max}(G)$ be the maximum k such that a k-VCC exists. For the input parameter k, we choose 20%, 40%, 60%, and 80% of the $k_{max}(G)$ of each tested graph G, with $k = 40\% \cdot k_{max}(G)$ as default.

[4]http://snap.stanford.edu/index.html
[5]http://webgraph.di.unimi.it/

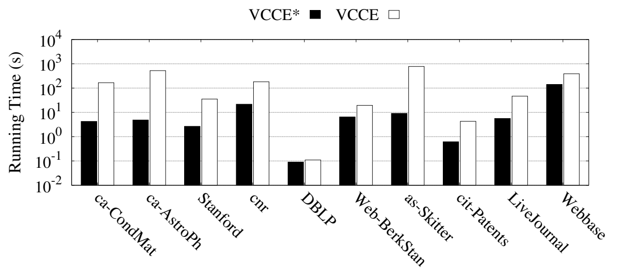


Fig. 6: Performance on different datasets

A. Performance Studies on Real-World Graphs

We report the running time of VCCE* on all datasets with VCCE as a comparison in Fig. 6. We can find that VCCE* is significantly faster than VCCE on most of datasets. It is the widest that the gap between the bars of two algorithms on ca-AstroPh; VCCE* costs about 4 seconds, while VCCE costs over 500 seconds. Note that the algorithmic speedup on DBLP is not obvious. In this case, a large number of vertices are removed due to the k-core constraint and the number of k-VCCs in DBLP is small under the default parameter setting. Even though about 45% of LOC-CUT invocations are avoided in VCCE* due to our pruning techniques, the total number of LOC-CUT invocations is quite small (about 100) in VCCE. When the parameter k drops to 20% of k_{max} on DBLP, VCCE costs about 80 seconds, while VCCE* costs only about 4 seconds. The detailed relationship between the algorithmic efficiency and the parameter k will be analyzed later. In the largest dataset Webbase, VCCE* costs about 140 seconds, while VCCE costs about 390 seconds.

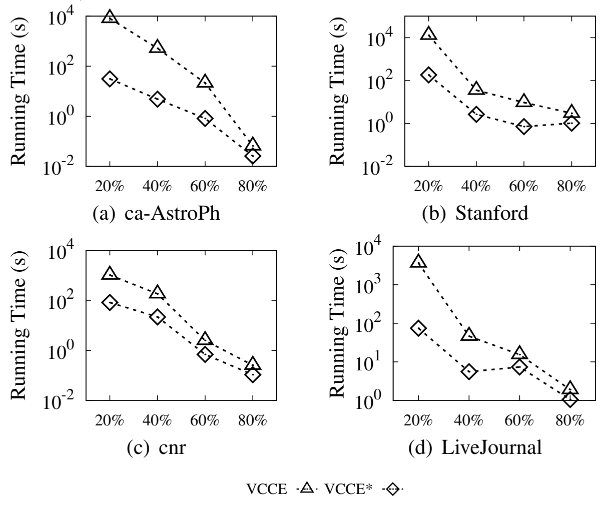


Fig. 7: Against basic algorithm (Vary k)

Vary k. We report the running time of our proposed algorithms on different datasets when varying k in Fig. 7. Due to the space limitation, we only show the results on ca-AstroPh, Stanford, cnr and LiveJournal, while the results on other datasets have similar trends.

The running time of VCCE presents a downward trend on all datasets when raising k. For example, in Fig. 7 (c), VCCE costs about 17 minutes to compute all k-VCCs in cnr when $k = 20\% \cdot k_{max}$; the running time drops to under 1 second given $k = 80\% \cdot k_{max}$. The decrease of running time of VCCE is mainly due to following two reasons. First, given a large value of parameter k, a large number of vertices are removed due to the k-core constraint (line 2 of Algorithm 1). Second, it is more likely to find a qualified vertex cut (line 15 of GLOBAL-CUT) given a high value of parameter k; that means the graph is more likely to be partitioned into small subgraphs.

Input Parameter	*Non-Pru*	*NS_1*	*NS_2*	*GS*
ca-AstroPh				
$20\% \cdot k_{max}$	3.5%	20.4%	30.2%	45.9%
$40\% \cdot k_{max}$	3.8%	40.0%	23.8%	32.3%
$60\% \cdot k_{max}$	13.8%	42.10%	38.40%	5.70%
$80\% \cdot k_{max}$	100%	0%	0%	0%
avg	4.1%	27.3%	28.7%	39.9%
Stanford				
$20\% \cdot k_{max}$	2.9%	32.3%	14.0%	50.9%
$40\% \cdot k_{max}$	10.1%	12.0%	18.7%	59.2%
$60\% \cdot k_{max}$	7.1%	3.6%	25.6%	63.6%
$80\% \cdot k_{max}$	12.0%	0%	14.7%	73.3%
avg	3.3%	30.7%	14.4%	51.6%
cnr				
$20\% \cdot k_{max}$	3.0%	12.4%	16.0%	68.6%
$40\% \cdot k_{max}$	30.7%	8.9%	47.3%	13.2%
$60\% \cdot k_{max}$	15.8%	27.1%	16.3%	40.7%
$80\% \cdot k_{max}$	61.1%	38.9%	0%	0%
avg	7.0%	12.1%	20.4%	60.5%
LiveJournal				
$20\% \cdot k_{max}$	1.5%	3.6%	39.2%	55.7%
$40\% \cdot k_{max}$	8.5%	6.9%	61.1%	23.4%
$60\% \cdot k_{max}$	19.3%	2.6%	69.7%	8.4%
$80\% \cdot k_{max}$	84.4%	15.6%	0%	0%
avg	3.4%	4.0%	43.2%	49.4%

TABLE II: Evaluating pruning rules

We can find in Fig. 7 that VCCE* is faster than VCCE on all parameter settings and the gap between the running time of two algorithms is wide when k is small. This phenomenon demonstrates that our optimized techniques is more effective when VCCE requires a great amount of computational cost. For example, in Fig. 7 (a), VCCE costs over 2 hours when k is 20% of k_{max}. In contrast, VCCE* only need about 30 seconds, which is over $200\times$ faster than VCCE. The running time of VCCE* and VCCE drops to about 0.3 seconds and 0.7 seconds respectively when $k = 80\% \cdot k_{max}$.

Note that there exists a slight increase for the running time of VCCE* from 60% to 80% in Fig. 7 (d). This is because both of the numbers of strong side-vertices and vertices in side-groups decrease when raising k. Even though the total number of max-flow computations in VCCE decreases, the ratio of pruned computations increases and this leads to a compromise result. Similar phenomenon also appears in Fig. 7 (b) from 60% to 80%.

B. Evaluating Optimization Techniques

To further investigate the effectiveness of our sweep rules, we also track each processed vertex during the performance of VCCE* and record the number of vertices pruned by each strategy. Specifically, when performing sweep procedure, we separately mark the vertices pruned by *neighbor sweep rule 1* (strong-side vertex), *neighbor sweep rule 2* (neighbor deposit) and group sweep. Here, we divide neighbor sweep into two detailed sub-rules since the both of them perform well and the effectiveness of these two strategies is not very consistent in different datasets. For each vertex v in line 11 of GLOBAL-CUT*, we increase the count for corresponding strategy if v is pruned (line 12). We also record the number of vertices which are non-pruned and really tested (line 13). For each dataset, we record these data on different k from 20% to 80% of k_{max}. Additionally, we sum the processed vertices on all parameter settings and calculate the average ratio of each pruning rule for each dataset. All results are summarized in Table II. *NS_1* and *NS_2* represent *neighbor sweep rule 1* and *neighbor sweep rule 2* respectively. *GS* is group sweep and *Non-Pru* means the proportion of non-pruned vertices. If more than one rules can be adopted to prune a vertex, we mark this vertex according to the following priority, NS_1, NS_2 and GS.

The result shows our pruning strategies are effective, especially when k is small. The average ratio of non-pruned vertices is less than 10% on all datasets, and when $k = 20\% \cdot k_{max}$, the ratio of non-pruned vertices is under 5% on all tested datasets and even drops to about 1.5% on LiveJournal. The ratio of non-pruned vertices roughly presents an upward trend on most of datasets. For example, on ca-AstroPh, that ratio reaches 3.8% and 13.8% on $40\% \cdot k_{max}$ and $60\% \cdot k_{max}$ respectively. We can see that there exists no pruned vertex when $k = 80\% \cdot k_{max}$ on ca-AstroPh, because the number of max-flow computations is already very small and the room for improvement is not enough. On ca-AstroPh, VCCE* computes max-flow only 1 time on $80\% \cdot k_{max}$.

The effectiveness of different rules depends on the detailed graph structure. We can find that the average ratio of group sweep rules is the largest on all datasets. It accounts for about 39.9% of total on ca-AstroPh and up to 60.5% on cnr. The *neighbor sweep rule 1* performs well on Stanford; its average ratio is about 30.7%. However, this rule only prunes about 4% of total on LiveJournal. By contrast, the average ratio of *neighbor sweep rule 2* is about 14.4% on Stanford, but up to 43.2% on LiveJournal.

C. Scalability Testing

In this section, we test the scalability of our proposed algorithms. We choose two real graph datasets cnr and LiveJournal as representatives. For each dataset, we vary the graph size and graph density by randomly sampling vertices and edges respectively from 20% to 100%. When sampling vertices, we get the induced subgraph of the sampled vertices, and when sampling edges, we get the incident vertices of the edges as the vertex set. About the parameter setting, we fix k to 10% of k_{max} of the original graph for all sampling ratio on each dataset. The experimental results are shown in Fig. 8.

Fig. 8 (a) and (b) report the processing time of our proposed algorithms when varying $|V|$ in cnr and LiveJournal respectively. The curves in Fig. 8 (c) and (d) report the processing time of our algorithms on cnr and LiveJournal respectively when varying $|E|$. Note that in Fig. 8 (b) and (d), the running time of VCCE is not given on 60% and 80%, since the corresponding procedure cannot finish in 24 hours.

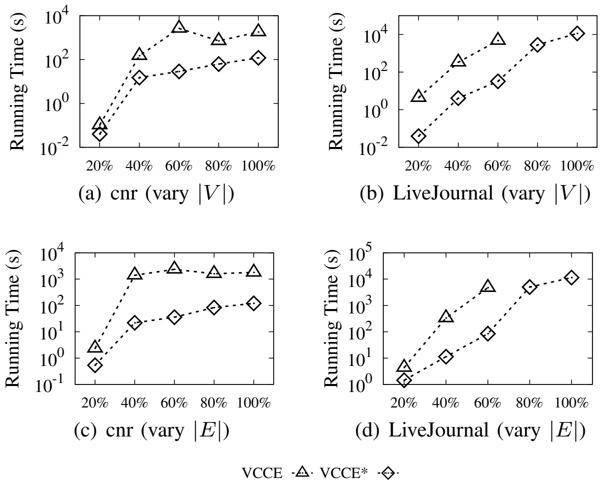


Fig. 8: Scalability testing

The efficiency of VCCE heavily relies on the detail graph structures, and the lines of VCCE are not stable. For example, in Fig. 8 (a), the running time of VCCE is about 3 seconds when sampling 20% nodes, and it reaches about 40 minutes on 60%. Then running time drops slightly to about 27 minutes on 80%, and increase back to about 30 minutes on 100%. By contrast, the running time of VCCE* presents a steadily upward trend on all datasets when the sampling ratio increases. For example, VCCE* costs less than 0.1 second on 20% in Fig. 8 (a); its running time reaches about 15 seconds, 28 seconds and 62 seconds on 40%, 60% and 80% respectively. VCCE* costs about 2 minutes on 100%, which is 15× faster than VCCE. The result shows that our pruning strategies are effective and our optimized algorithm is more efficient and scalable than the basic algorithm.

VII. Related Work

Cohesive Subgraph. In social network analysis, several cohesive subgraph metrics have been studied for graph segmentation. [25], [26] propose algorithms for maximal clique enumeration. However, the definition of clique is too strict, and some relaxed metrics are proposed, which can be roughly classified into the following three categories.
1. Global Cohesiveness. [27] defines an s-clique model by allowing the distance between two vertices to be at most s. However, it does not require all intermediate vertices are in the s-clique itself. [28] proposes an s-club model requiring that all intermediate vertices are in the same s-club. k-plex allows each vertex in such subgraph can miss at most k neighbors [29], [30]. Quasi-clique is a subgraph with n vertices and at least $\gamma * \binom{n}{2}$ edges [31].
2. Local Degree and Triangulation. k-core is maximal subgraph in which each vertex has a degree at least k [11]. k-truss has also been investigated in [32], [33], [34]. It requires each edge in a k-truss is contained in at least $k-2$ triangles. This model is also independently defined as k-mutual-friend subgraph in [35]. Based on triangles, *DN*-graph [36] with parameter k is a connected subgraph $G'(V', E')$ satisfying

following two conditions: 1) Every connected pair of vertices in G' shares at least λ common neighbors. 2) For any $v \in V \backslash V', \lambda(V' \cup \{v\}) < \lambda$; and for any $v \in V', \lambda(V' \backslash \{v\}) \leq \lambda$.
3. Connectivity Cohesiveness. In this category, most of existing works only consider edge connectivity of a graph. The edge connectivity of a graph is the minimum number of edges whose removal disconnect the graph. [37] first proposes algorithm to efficiently compute frequent closed k-edge connected subgraphs from a set of data graphs. However, a frequent closed subgraph may not be an induced subgraph. To conquer this problem, [10] gives a cut-based method to compute all k-edge connected components in a graph. [38] proposes a decomposition framework for the same problem to further improve efficiency. [17] follows the basic partition-based framework for computing k-VCCs. They also propose an approximate algorithm to achieve a speedup. [19] computes k-VCCs for small k values.

Vertex Connectivity. [24] proves the time complexity of computing maximum flow reaches $O(n^{0.5}m)$ in an unweighted directed graph while each vertex inside has either a single edge emanating from it or a single edge entering it. This result is used to test the vertex connectivity of a graph with given k in $O(n^{0.5}m^2)$ time. [39] further reduces the time complexity of such problem to $O(k^3m+knm)$. There are also other solutions for finding the vertex connectivity of a graph [40], [20]. To speed up the computation of vertex connectivity, [22] finds a sparse certificate of k-vertex connectivity.

VIII. Conclusions

Computing all k-vertex connected components is a foundational problem and has been studied recently. The state-of-the-art solution does not provide the polynomial running time guarantee, and requires high computational cost on computing the vertex cut. In this paper, we study the problem of k-VCC enumeration and prove that the algorithm terminates in polynomial time. We propose several optimization strategies to significantly improve the efficiency of the algorithm. We conduct extensive experiments using ten real datasets to demonstrate the efficiency of our approach.

Acknowledgements. Lu Qin is supported by ARC DP160101513. Ying Zhang is supported by ARC FT170100128 and DP180103096. Lijun Chang is supported by ARC DE150100563 and DP160101513. Ling Chen is supported by ARC DP180100966.

References

[1] E. Otte and R. Rousseau, "Social network analysis: a powerful strategy, also for the information sciences," *Journal of information Science*, vol. 28, no. 6, pp. 441–453, 2002.

[2] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[3] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*, 1994, vol. 8.

[4] M. E. Newman, "Modularity and community structure in networks," *PNAS*, vol. 103, no. 23, pp. 8577–8582, 2006.

[5] J. Moody and D. R. White, "Structural cohesion and embeddedness: A hierarchical conception of social groups." *American Sociological Review*, vol. 68, 2000.

[6] D. R. White and F. Harary, "The cohesiveness of blocks in social networks: Node connectivity and conditional density," *Sociological Methodology*, vol. 31, no. 1, pp. 305–359, 2001.

[7] J. Torrents and F. Ferraro, "Structural cohesion: Visualization and heuristics for fast computation." *JoSS*, vol. 16, pp. 1–36, 2015.

[8] D. W. Matula, "k-blocks and ultrablocks in graphs," *JCTB*, vol. 24, no. 1, pp. 1–13, 1978.

[9] H. Whitney, "Congruent graphs and the connectivity of graphs," *American Journal of Mathematics*, vol. 54, no. 1, pp. 150–168, 1932.

[10] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li, "Finding maximal k-edge-connected subgraphs from a large graph," in *EDBT*, 2012.

[11] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," *CoRR*, vol. cs.DS/0310049, 2003.

[12] J. Edachery, A. Sen, and F. Brandenburg, "Graph clustering using distance-k cliques," in *GD*, 1999.

[13] L. Caccetta and W. Smyth, "Graphs of maximum diameter," *Discrete mathematics*, vol. 102, no. 2, pp. 121–141, 1992.

[14] J. Xie, S. Kelley, and B. K. Szymanski, "Overlapping community detection in networks: The state-of-the-art and comparative study," *ACM Comput. Surv.*, vol. 45, no. 4, pp. 43:1–43:35, 2013.

[15] S. Pemmaraju and S. Skiena, "Computational discrete mathematics: Combinatorics and graph theory with mathematica®," pp. 290–291, 2003.

[16] S. S. Skiena, "The algorithm design manual," 1998.

[17] Y. Li, Y. Zhao, G. Wang, F. Zhu, Y. Wu, and S. Shi, "Effective k-vertex connected component detection in large-scale networks," in *DASFAA*, 2017, pp. 404–421.

[18] K. Menger, "Zur allgemeinen kurventheorie," *Fundamenta Mathematicae*, vol. 10, no. 1, pp. 96–115, 1927.

[19] R. S. Sinkovits, J. Moody, B. T. Oztan, and D. R. White, "Fast determination of structurally cohesive subgroups in large networks," *JoCS*, vol. 17, pp. 62–72, 2016.

[20] A. H. Esfahanian and S. Louis Hakimi, "On computing the connectivities of graphs and digraphs," *Networks*, vol. 14, no. 2, pp. 355–366, 1984.

[21] S. Even, "Graph algorithms," pp. 116–132, 1979.

[22] J. Cheriyan, M. Kao, and R. Thurimella, "Scan-first search and sparse certificates: An improved parallel algorithms for k-vertex connectivity," *SIAM J. Comput.*, vol. 22, no. 1, pp. 157–174, 1993.

[23] D. Wen, L. Qin, X. Lin, Y. Zhang, and L. Chang, "Enumerating k-vertex connected components in large graphs," *arXiv:1703.08668*, 2017.

[24] S. Even and R. E. Tarjan, "Network flow and testing graph connectivity," *SIAM J. Comput.*, vol. 4, 1975.

[25] L. Chang, J. X. Yu, and L. Qin, "Fast maximal cliques enumeration in sparse graphs," *Algorithmica*, vol. 66, 2013.

[26] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, 2011.

[27] R. D. Luce, "Connectivity and generalized cliques in sociometric group structure," *Psychometrika*, vol. 15, 1950.

[28] R. J. Mokken, "Cliques, clubs and clans," *Quality and Quantity*, vol. 13, 1979.

[29] D. Berlowitz, S. Cohen, and B. Kimelfeld, "Efficient enumeration of maximal k-plexes," in *SIGMOD*, 2015.

[30] S. B. Seidman and B. L. Foster, "A graph-theoretic generalization of the clique concept," *Journal of Mathematical Sociology*, vol. 6, 1978.

[31] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, "Coherent closed quasi-clique discovery from large dense graph databases," in *KDD*, 2006.

[32] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," *National Security Agency Technical Report*, p. 16, 2008.

[33] J. Wang and J. Cheng, "Truss decomposition in massive networks," *PVLDB*, vol. 5, 2012.

[34] Y. Shao, L. Chen, and B. Cui, "Efficient cohesive subgraphs detection in parallel," in *SIGMOD*, 2014.

[35] F. Zhao and A. K. Tung, "Large scale cohesive subgraphs discovery for social network visual analysis," in *PVLDB*, vol. 6, 2012.

[36] N. Wang, J. Zhang, K.-L. Tan, and A. K. H. Tung, "On triangulation-based dense neighborhood graph discovery," *PVLDB*, vol. 4, 2010.

[37] X. Yan, X. J. Zhou, and J. Han, "Mining closed relational graphs with connectivity constraints," in *ICDE*, 2005.

[38] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *SIGMOD*, 2013.

[39] S. Even, "An algorithm for determining whether the connectivity of a graph is at least k," *SIAM J. Comput.*, vol. 4, 1975.

[40] Z. Galil, "Finding the vertex connectivity of graphs," *SIAM J. Comput.*, vol. 9, 1980.