Bottom-up k-Vertex Connected Component Enumeration by Multiple Expansion

Haoyu Liu, Yongcai Wang* , Xiaojia Xu, Deying Li
School of Information, Renmin University of China, Beijing, China
{hyliu2187, ycw, xuxiaojia, deyingli}@ruc.edu.cn

***Abstract*—Bottom-up k-vertex connected component (k-VCC) enumeration methods, referred to as VCCE-BU, have exhibited better efficiency compared to the exact top-down k-VCC enumeration method (VCCE-TD). However, VCCE-BU has been found to have surprisingly low detection accuracy, that it may detect fewer vertices in k-VCC than VCCE-TD. This raises the question of what causes VCCE-BU to have a low k-VCC enumeration quality. This paper investigates the reason and proposes that the local expansion should be reformulated as a Multiple vertex collaborative Expansion problem instead of the traditional Unitary Expansion (UE). A Multiple Expansion (ME) approach, which allows to expand multiple neighboring vertices jointly and collaboratively is proposed, which is proven exact in local expansion. However, the exact ME-based local expansion needs to explore large neighborhoods in each step, which is time-consuming. To address the efficiency issue, a Ring-based Multiple Expansion (RME) is proposed to conduct ME within one-hop neighbors. A maximum flow-based merging algorithm FBM is proposed for effective merging. A maximal clique and breath-first-search-based quick seeding algorithm QkVCS is proposed to generate k-VCC seeds efficiently. As a result, RIPPLE which integrates QkVCS+FBM+RME is presented as a new accurate and efficient bottom-up approach. Extensive verifications in real large-scale graph datasets demonstrate that even the single-thread RIPPLE is much more accurate and a magnitude faster than the state-of-the-art VCCE-BU method. We also demonstrate the effective speeding up to run RIPPLE in parallel.**

***Index Terms*—k-Vertex Connected Component Enumeration, Bottom-up Approach, Multiple Vertex Expansion**

I. Introduction

The discovery of communities is crucial to understand the patterns, relationships, and organizational structures of complex networks. It has broad applications in numerous fields, including social network analysis [5], bioinformatics [31], recommender systems [13], etc. Various community detection algorithms have been presented. Clique [21] and variations of clique, including quasi-clique [29], k-plex [1] detect complete or nearly complete subgraphs; k-core [7] divides the graph into multiple layers according to the node degree, while k-truss [17] detects the subgraphs where each edge is supported by enough triangles. These methods define cohesive features mainly based on local features of edges or vertices. Whereas in multi-hop graphs, high connectivity among vertices is undoubtedly an important feature for evaluating a community [2][19]. In view of this, various connectivity-based cohesive subgraphs have been formulated, such as s-bundle [30], γ-relative vertex connected subgraph [34], γ-relative edge connected subgraph [21], k-vertex connected component (k-VCC) [25][36], k-edge connected component (k-ECC) [40][6].

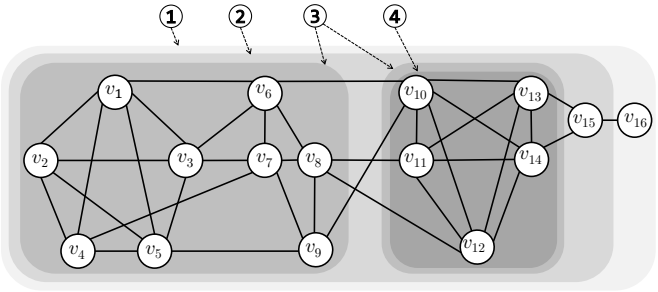


Fig. 1: k-VCCs in graph G.

Among them, the k-vertex connected component, i.e., k-VCC, is a subgraph that remains connected when any $k-1$ vertices are removed from the subgraph. It indicates there are at least k vertex disjoint paths between each pair of vertices in the component. The k-vertex-disjoint path is also generally required in networks [24][18] and traffic control [3] for transportation robustness against node failure. In graph theory, a k-VCC implies maximality, that each k-VCC is not a proper subgraph of any other larger k-VCC. k-VCC offers insight into a network's structural integrity and fault tolerance based on vertex connectivity. For its importance, it is valuable to ask to list all the k-VCCs, i.e., the k-VCC enumeration problem in large graphs. Figure 1 shows an example of k-VCC enumeration in graph G consisting of 16 vertices and 36 edges. The distribution of k-VCCs is illustrated for all possible k. When $k = 1$, the entire graph G represents a 1-VCC since it is a connected component. For $k = 2$, $G_1 = \{v_i \mid 1 \leq i \leq 15\}$ is a 2-VCC because v_{16} has only one vertex-disjoint path to other vertices, so v_{16} is excluded. When $k = 3$, there are two 3-VCCs, $G_2 = \{v_{10}, v_{11}, v_{12}, v_{13}, v_{14}\}$ and $G_3 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$. For $k = 4$, only one 4-VCC exists, namely G_2.

Existing methods [39][26][36][25] for k-VCC enumeration can be roughly divided into two categories. The first is "top-down", termed VCCE-TD, which iteratively finds vertex cuts of size less than k in the graph by computing the maximum

This work was partially supported by the National Natural Science Foundation of China Grant No. 61972404, No. 12071478, and Public Computing Cloud, Renmin University of China.

*Yongcai Wang is the corresponding author.

flow and removes them until each connected subgraph is k-vertex connected. VCCE-TD is proven to be exact [36]. However, it needs to cache large subnetworks, and the iterative maximum flow is computationally expensive. Another drawback is that it is hard to be parallelized. The second is the "bottom-up" method, termed VCCE-BU, which follows a *seeding, expansion, and merging* pipeline [25]. VCCE-BU shows better efficiency than VCCE-TD. Besides, Li et al. [26] adopt a combination of top-down and bottom-up approaches.

VCCE-BU firstly finds some seeding k-vertex connected subgraphs (k-VCSs) and then expands the k-VCSs from the seeds through local expansion. When two k-VCSs meet, conditions are designed to merge the two k-VCSs [25]. VCCE-BU shows one order of magnitude faster than VCCE-TD and is easily paralleled [25]. However, current VCCE-BU shows surprisingly low detection accuracy than VCCE-TD, i.e., the Cross Common Fraction (1), which evaluates the common fraction between the detected k-VCC vertices and the ground-truth k-VCC vertices is much lower than 100% in most cases. This problem is strange and raises the problem of how far the VCCE-BU is from the exact VCCE-TD in the enumeration quality. Meanwhile, the enumeration-based seeding in current VCCE-BU is inefficient in large graphs.

This paper revisits VCCE-BU and shows that the main reason for the inaccuracy is the heuristics used in the local expansion and merging. In current VCCE-BU, a Unitary Expansion (UE) heuristic, which expands one neighboring vertex at each time, is used, and a neighbor-counting-based merging method is used for merging k-VCSs. However, we show that the UE is only a special case of valid expansion options. Instead, we propose a *Multiple vertex collaborative Expansion* (ME) problem that allows multiple vertices to be expanded jointly and collaboratively since multiple expanded vertices can provide vertex disjoint paths for each other, while UE fails to expand in such cases. Therefore, a ME-based local expansion algorithm is proposed and we prove the ME-based local expansion is exact. We also show existing merging condition is problematic in some cases, thus we propose a new Flow-Based Merging (FBM) condition.

However, the exact multiple expansion and merging method needs high computation costs for the requirement to explore large neighborhoods to keep the "exact" property. To speed up, a Ring-based Multiple Expansion (RME) is proposed to restrict the expansion step size within one-hop neighbors to conduct ME efficiently. A maximum clique and breath-first-search-based seeding algorithm, i.e., QkVCS is also proposed to generate k-VCC seeds efficiently. At last, RIPPLE, which integrates QkVCS+RME+FBM is presented as a new practical bottom-up approach. Extensive verifications in real large-scale graph datasets demonstrate that even the single thread RIPPLE is much more accurate and one magnitude time faster than the state-of-the-art VCCE-BU. RIPPLE is easy to run in parallel and is much speeded up in parallel running. ME actually provides a flexible control to the local search step size based on the user's preference for efficiency or accuracy.

The main contributions are summarized as follows.

- The inaccuracy problem of VCCE-BU is shown due to UE and neighbor-counting-based merging. We reformulate the local expansion problem as a multiple vertex collaborative expansion problem and multiple expansion algorithm (ME), which is proven exact. We further propose a ring-based multiple expansion (RME), which is much more time-efficient.
- We propose a Flow-Based-Merging (FBM) method for k-VCS merging and QkVCS, a maximal clique and BFS-based seeding algorithm to speed up the seed generation.
- We propose RIPPLE, which conducts RME and FBM from the distributed k-VCC seeds generated by QkVCS. RIPPLE is not only accurate but also efficient.

II. RELATED WORK

A. Vertex Connectivity

As for the vertex connectivity, denoted as κ, the smallest cardinality of a vertex separator of a given graph G; To compute κ, Henzinger et al. [15] proposed an $O(min\{\kappa^3 + n, \kappa n\}m)$ time algorithm to calculate the connectivity of a digraph. Gabow [12] improved the best time bound of computing κ to $O((n + min\{\kappa^{\frac{5}{2}}, \kappa n^{\frac{3}{4}}\})\kappa n)$. There are some randomized algorithms to compute κ. Nanongkai et al. [28] presented a randomized Monte Carlo algorithm with time $O(m + k^{\frac{7}{3}} n^{\frac{4}{3}})$. Forster et al. [11] improved the efficiency of the algorithm to $O(m + k^3 n)$ time.

B. k-VCC Enumerating

Existing algorithms for k-VCC enumerating can be classified into two categories: exact algorithms (Top-down Framework) and heuristic algorithms (Bottom-up Framework). For the exact algorithms, Even and Tarjan [10] developed the first exact k-VCC enumerating algorithms based on network flow which is very efficient on sparse graphs with $O(|V|^{2.5} \cdot E)$ time complexity. Such an algorithm was improved by Cheriyan et al. [8] based on the sparse certificates technique. Subsequently, Wen et al. [36] studied the polynomial running time algorithm for the k-VCC enumeration. They proposed two optimization strategies to improve the algorithmic efficiency significantly. For the heuristic algorithm, Li et al. [26] proposed an approach in a bottom-up manner, which is more efficient in time and space. This algorithm can be further combined with the top-down manner approach to get exact results [25]. For k = 2, Tarjan [33] gave a linear time algorithm, and for k = 3, Hopcroft et al. [16] also proposed a linear time algorithm. For any constant k, Henzinger et al. [14] presented an $O(n^3)$ time algorithm for computing k-vertex strongly connected components in directed graphs.

C. Seed Expansion

The seed expansion method is widely used in different domains to identify overlapping communities. In the seed expansion method, a seed is a small set of vertices chosen in specific ways. Expansion is accomplished by traversing the vertices around the seed iteratively and adding them optionally. Lancichinetti et al. [22] first studied an algorithm that is based

on the local expansion of a community. A method called Rank Removal (RaRe) is proposed by Baumes et al. [4] for seed selection. Lee et al. [23] proposed Greedy Clique Expansion (GCE) algorithm. This algorithm takes different cliques as seeds and expands these seeds by greedy optimization of local fitness functions. An efficient overlapping community detection algorithm using a seed set expansion approach is proposed by Whang et al. [37]. They then studied a neighborhood-inflated seed expansion algorithm [38]. The neighborhood inflation step, where seeds are modified to represent their whole vertex neighborhood, is an essential step in this method. Kloumann et al. [20] developed a principled framework for evaluating ranking methods by studying seed set expansion applied to the stochastic block model. Sein [32] proposed an overlapping seed expansion algorithm to find the best seeds.

III. PRELIMINARIES AND BACKGROUND

In this section, we present notations and definitions related to the k-vertex connected component enumeration.

A. Problem Definition

We consider an undirected and unweighted graph $G(V, E)$, where V is the set of vertices and E is the set of edges. We consider only simple graphs, i.e., without self-loops and parallel edges. The number of vertices and the number of edges are denoted by $n = |V|$ and $m = |E|$. We denote the set of h-hop neighbors of vertex u in G by $N_G^h(u) = \{u \in V \mid dist(u, v) \leq h\}$, and the set of h-hop neighbors of vertex set S in G by $N_G^h(S) = \bigcup_{u \in S} N_G^h(u)$. For simplicity, we omit superscripts when $h = 1$. We denote the degree of u by $d_G(u) = |N_G(u)|$ since G is a simple graph. We use $\mathcal{B}(S)$ to denote the *boundary* [25] of the induced subgraph $G[S]$: $\forall v \in S$, if $\exists u \in N(v)$ and $u \notin S$, then $v \in \mathcal{B}(S)$; and $\bar{S}$ denotes the set of vertices in $G \setminus S$. Let $Max\_Flow_G^{s \to t}$ be the maximum flow from s to t in G. $\mathcal{MC}$ is the set of the maximal cliques. $u \equiv_G^k v$ means there are at least k vertex-disjoint paths between u and v in G. $u \equiv_G^k S$ means $\forall v \in S, u \equiv_G^k v$. Frequently used notations are summarized in Table I.

TABLE I: Notations

Notation	Meaning
$G = (V, E)$	an unweighted and undirected graph with vertex set V and edge set E
$N_G^h(u)$, $N_G^h(S)$	the set of h-hop neighbors of u or S in G, we omit superscript when $h = 1$
$G[S]$	the subgraph of G induced by the vertices in S
$d_G(u)$	the cardinality of $N_G(u)$
$\mathcal{B}(S)$	the set of vertices in boundary of S
$\bar{S}$	the set of vertices in $G \setminus S$
$Max\_Flow_G^{s \to t}$	maximum flow from s to t in G
$\mathcal{MC}$	the set of maximal cliques
$u \equiv_G^k v$	u has at least k vertex disjoint paths to u in G, or S is no less than k in G
$u \equiv_G^k S$	$\forall v \in S, u$ has at least k-vertex disjoint paths to v

A graph is k-vertex connected if it remains connected after the removal of any $k - 1$ vertices from it. The k-vertex connected subgraph is defined as follows.

Definition 1 (k-Vertex Connected Subgraph)**.** *Given a graph G, a subgraph g is a k-vertex connected subgraph (k-VCS) of G if $\forall u, v \in g$, there are at least k vertex-disjoint paths between u and v in g.*

Considering the potential inclusion relationships among the k-VCSs, we define the k-vertex connected component.

Definition 2 (k-Vertex Connected Component)**.** *Given a graph G, a subgraph g is a k-vertex connected component (k-VCC) of G if (i) g is a k-vertex connected subgraph and (ii) g is maximal (i.e., $\nexists$ another k-VCS $g' \subseteq G$, such that $g \subsetneq g'$).*

Definition 3 (k-VCC Enumeration Problem)**.** *Given a graph G and an integer k, the problem of k-VCC enumeration (k-VCCE) aims to find all k-VCCs of G. If the value of k is clear from the context, we use "VCCE" to mean k-VCCE.*

B. State-Of-The-Arts

The state-of-the-art of k-VCCE approaches can be roughly divided into top-down and bottom-up two approaches.

1) Top-Down Approach: VCCE-TD. The top-down approach recursively partitions the graph into smaller subgraphs until the remained graph is a single-vertex graph or a k-VCC. The partitioning process is achieved by removing the minimum cuts in the graph when the cardinality of the minimum cut is less than k [36]. VCCE-TD runs in $O\left(\min\left(n^{0.5}, k\right) \cdot m \cdot \left(n + \delta^2\right) \cdot n\right)$ time to exactly compute all k-VCCs, where δ is the minimum degree in the input graph.

Limitations of VCCE-TD. Although the top-down approach is exact, it has several limitations: (1) since VCCE-TD adopts the recursive graph partition scheme, it requires storing the partitioned subgraphs in memory, which is extremely space-inefficient for large graphs. (2) it relies on building a flow network for each subgraph to compute the minimum cut, which compromises efficiency even though some optimizations are implemented. (3) it is difficult to run in parallel.

2) Bottom-Up Approach: VCCE-BU. Seeing the limitations of the top-down approach, the bottom-up approach enumerates k-VCCs using a seeding, local expansion, and merging pipeline [25]. It first finds a set of k-VCSs as seeds and then expands these k-VCSs via local expansion to pursue the k-VCCs rooted in these seeds. When two k-VCSs meet, merging conditions are checked to merge the k-VCSs to pursue larger k-VCCs. Benefiting from the local expansion scheme, the bottom-up approach is efficient in storage and computation. Additionally, it can be implemented in parallel. However, VCCE-BU has a noticeable limitation in that the k-VCC enumeration accuracy is unsatisfactory compared to the exact VCCE-BU. In Table III, we evaluate the accuracy of VCCE-BU by F_{same} and J_{Index}. It can be seen that the accuracy results are unsatisfactory in most cases. The efficiency of enumeration-based seeding is also not satisfied in large graphs.

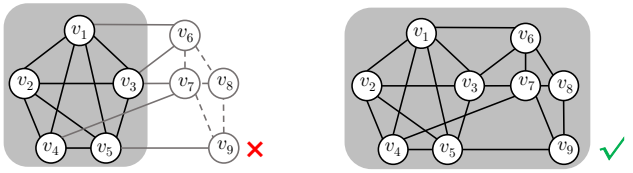
C. A Revisit to VCCE-BU

*1) **Why VCCE-BU has unsatisfactory accuracy?**:* We first investigate why the VCCE-BU has unsatisfactory accuracy.

Reason 1: The Limitation of Unitary Expansion

Firstly, expanding is the primary method for extending a k-VCS, but in the current VCCE-BU method, the local expansion considers expanding a single vertex at each time. We call it Unitary Expansion (UE). We will show that UE is one of the main reasons for losing accuracy.

For each seeding subgraph S, UE takes the vertices in $\mathcal{B}(\bar{S}) = \{u \mid u \in V(G) \setminus S, d_{G[S\cup u]}(u) \geq 1\}$ as the candidate set. Then it calculates $d_{G[S\cup u]}(u)$ for each vertex u in the candidate set $\mathcal{B}(\bar{S})$. u is added into S if $d_{G[S\cup u]}(u) \geq k$. UE iteratively executes this unitary vertex expansion procedure until there is no vertex in $\mathcal{B}(\bar{S})$ that satisfies the above expansion condition.



(a) Failure of Unitary Expansion (b) Successful Multiple Expansion

Fig. 2: Unitary Expansion vs. Multiple Expansion

An example in Fig. 2 shows the limitation of UE, where we set $k = 3$. Suppose $S = \{v_1, v_2, v_3, v_4, v_5\}$ is a 3-VCS found in the seeding phase. Using UE, $\mathcal{B}(\bar{S}) = \{v_6, v_7, v_9\}$ are considered as the candidate set, but none of the vertices in $\mathcal{B}(\bar{S})$ can be added into S because each vertex has fewer than three neighbors in S. However, if the connectivity between v_6 and v_7 is considered, we find that v_6 and v_7 are adjacent; v_6 provides a new vertex-disjoint path for v_7 (i.e., $v_7 \to v_6 \to v_1$), and vice versa. Thus, $\{v_6, v_7\}$ should actually be expanded by the seed subgraph at the same time. Furthermore, $\{v_8, v_9\}$ can also be added in a similar manner after S is updated, so UE misses the opportunity to expand four vertices, as shown in Figure 2(b).

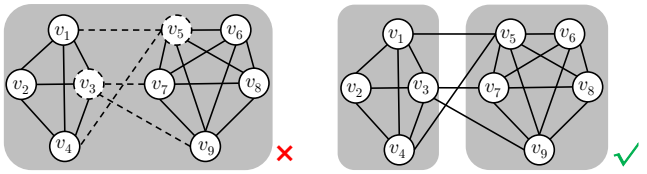
Reason 2: The Limitation of Neighbor-based Merging

In the VCCE-BU, in addition to local expansion, when some k-VCSs meet during expansion, VCCE-BU attempts to combine the k-VCSs to obtain a larger k-VCS, which is called *merging*. Assume that S and S' are sets of vertices in a graph G, i.e., $S \subseteq V, S' \subseteq V$, and the induced subgraph $G[S]$ and $G[S']$ are two detected k-VCSs, VCCE-BU gives a neighbor-based merging condition for merging $G[S]$ and $G[S']$.

Proposition 1 (Neighbor-based Merging (NBM) [25])**.** *$G[S \cup S']$ is k-vertex connected, if $|S \cap S'| + \min\{|N_{G[S'\setminus S]}(S \setminus S')|, |N_{G[S\setminus S']}(S' \setminus S)|\} \geq k$.*

Note that $N_{G[S'\setminus S]}(S \setminus S')$ is the *pure neighbors* of S in S', i.e., excluding the overlapped vertices of S and S'. As shown in Fig. 3, suppose $S = \{v_1, v_2, v_3, v_4\}$ and $S' = \{v_5, v_6, v_7, v_8, v_9\}$, then $N_{G[S'\setminus S]}(S \setminus S') = \{v_5, v_7, v_9\}$, $N_{G[S\setminus S']}(S' \setminus S) = \{v_1, v_3, v_4\}$, since $S \cap S' = \emptyset$.

According to Proposition 1, in Fig.3, $|S \cap S'| + \min\{|N_{G[S'\setminus S]}(S \setminus S')|, |N_{G[S\setminus S']}(S' \setminus S)|\} = 0 + \min\{3, 3\} = 3 \geq 3$. So NBM merges S and S' as shown in Fig. 3(a). However, if we delete two vertices v_3 and v_5 from the input graph, as



(a) Neighbor-Based Merging (b) Exact Merging

Fig. 3: An example when Neighbor-Based Merging is incorrect

shown in Fig. 3(a), there will be no edges preserved between S and S', i.e., $G[S \cup S']$ is not 3-vertex connected. The exact result is in Fig. 3(b). The mistake is that some vertices in $\mathcal{B}(S)$ have multiple neighbors in $\mathcal{B}(S')$, and vice versa. As a result, they are included multiple times in counting neighbors. Proposition 1 simply utilizes the minimum value of the vertices on two sides, which does not consider this duplication.

*2) **The enumeration-based seeding is not efficient**:* VCCE-BU [25] proposes a seeding method named LkVCS to find k-VCSs around the neighborhood of vertices as seed subgraphs. It restricts the search scope within the $N_G^2(u)$ neighborhood of the start vertex u as a path length constraint. Then LkVCS enumerates vertex subsets of size k from $N_G(u)$ and incrementally adds vertices until it becomes a k-VCS or is sufficient to reject the possibility of being a k-VCS. Because the combinational enumeration has time complexity $\binom{d_G(u)}{k} = \frac{d_G(u)!}{(d_G(u)-k)!k!}$, which can be very large, so the overall enumeration is slow.

Li et al. [25] further set a threshold α to restrict the number of enumerations. To visit all the vertices in the graph, LkVCS is invoked n times at the worst case, which runs in $O(\alpha |E_{avg}|)$, where $|E_{\text{avg}}|$ represents the average edge number in $G[N_G^2(u)]$. Thus, the time complexity for seeding is $O(n\alpha |E_{avg}|)$. However, if the distribution of k-VCCs is locally dense in the graph, setting a threshold may result in the loss of many potential k-VCCs, which decreases the accuracy.

D. A Novel Multiple Expansion Framework for VCCE-BU

Seeing the above limitations, we propose a novel Multiple Expansion framework for VCCE-BU. At first, to improve accuracy, we must consider that the candidate expandable multiple vertices can provide vertex-disjoint paths for each other. Considering S is the vertex set of a k-VCS and C is a candidate set in $V \setminus S$, we denote $S' = S \cup C$. We first give the proposition when the set C can be expanded as a whole.

Proposition 2 (Collaboratively Multiple Vertex Expansion (ME))**.** *Given a k-VCS $G[S]$, a candidate set C. Let $S' = S \cup C$. If $\forall u \in C$, $v \in S'$, $u \equiv^k_{G[S']} v$, i.e., u has at least k vertex-disjoint paths to all other vertices in S' through only edges in $G[S']$, then $G[S']$ is k-vertex connected, so all vertices in C can be added into S simultaneously, i.e. $S \leftarrow S'$.*

The ME raises two key problems: (1) how to select a candidate set. (2) how to efficiently count the vertex-disjoint paths. The first problem determines the local searching scope, and the second problem determines the verification complexity. In Section IV, we first present the exact ME. In Section V, we set C to be one-hop neighbors of S and present a Ring-based ME

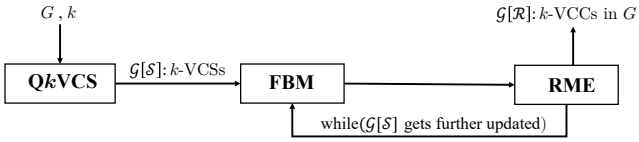


Fig. 4: Flow Diagram of RIPPLE

method (RME) for efficiency. So actually, the ME framework provides the flexibility for choosing local searching scope C based on the user's preference for accuracy or efficiency.

Further, to address the inaccuracy problem of merging, we present a Flow-Based Merging method (FBM) in Section IV. To address the inefficiency problem of enumeration-based seeding, we propose a Maximal Clique and BFS-based Seeding algorithm (QkVCS) in Section V. As showm in Figure 4, RME, FBM, and QkVCS are integrated as RIPPLE, a new practical VCCE-BU method in Section V. We show RIPPLE is not only more accuracy benefited by RME and FBM but also more efficient benefited by RME and QkVCS.

IV. Exact Local Expansion and Merging

In this section, we investigate the exact local expansion and merging method.

A. Multiple Expansion: ME

The limitation of UE is that it only considers a single vertex around the seed subgraph at each time of expansion. To solve this problem, we propose an exact expansion approach based on the idea of expanding multiple vertices simultaneously.

First, we introduce the definition of *local connectivity* between vertices as follows.

Definition 4 (local connectivity [36])**.** *Given a graph G, the local connectivity of two vertices u and v through G, denoted by $\kappa(u, v, G)$, is defined as the size of the minimum cut whose removal makes u and v disconnected. $\kappa(u, v, G)$ is assumed to be $+\infty$ if two vertices are directly connected.*

If the local connectivity of u and v is not less than k in graph G, we denote $u \equiv^k_G v$, which means u and v have at least k vertex-disjoint paths in G; and $u \not\equiv^k_G v$ means u and v have less than k vertex-disjoint paths in G. Moreover, we define $u \equiv^k_G S$, if $u \equiv^k_G v$ holds for every vertex $v \in S$. We omit the index G when the context is clear. Before giving multiple expansion conditions, we first introduce the following concept: Given a graph G and an integer k, a vertex v is called a *side-vertex* if there does not exist a vertex cut $\mathcal{VC}$ such that $|\mathcal{VC}| < k$ and $v \in \mathcal{VC}$ [36]. Based on the side-vertex, we have the following lemmas.

Lemma 1 ([36])**.** *Given a graph G and an integer k, suppose $u \equiv^k v$ and $v \equiv^k w$, we have $u \equiv^k w$ if v is a side-vertex.*

Lemma 2 ([36])**.** *Given a graph G, a vertex u is a side-vertex if and only if $\forall v, v \equiv^k v'$ where $v' \in N(u)$.*

Lemma 1 shows the transitive property about the local k connectivity relation $\equiv^k$, while Lemma 2 gives the conditions for determining the side-vertex.

Lemma 3. *Given a vertex $u \notin S$, a k-vertex connected subgraph $G[S]$, if there exists a vertex $v \in S \setminus \mathcal{B}(S)$ satisfying $u \equiv^k v$, then $u \equiv^k S$.*

Proof. $N(v) \subset S$ and S is k-vertex connected so that $\forall w, w' \in N(v), w \equiv^k w'$, thus v is a *side-vertex* (Lemma 2). We have (i) $u \equiv^k v$ (ii) $\forall v' \in S, v \equiv^k v'$ (iii) v is a *side-vertex*. According to Lemma 1, $\forall v' \in S, u \equiv^k v'$, i.e., $u \equiv^k S$.

□

Lemma 3 gives the condition to determine the local connectivity between a vertex and a k-vertex connected subgraph. However, if $S \setminus \mathcal{B}(S) = \emptyset$, Lemma 3 cannot be applied. To address this problem, we introduce a virtual vertex σ, which is adjacent to all vertices in S. Then we establish the following theorem to expand multiple vertices simultaneously.

Theorem 1. *Given a candidate set C and a k-vertex connected subgraph $G[S]$ with adding virtual vertex σ and edges $\{(\sigma, v) \mid v \in S\}$ to $G[S]$, if $u \equiv^k_{G[S \cup C]} \sigma$ for all vertices $u \in C$, $G[S \cup C]$ is k-vertex connected.*

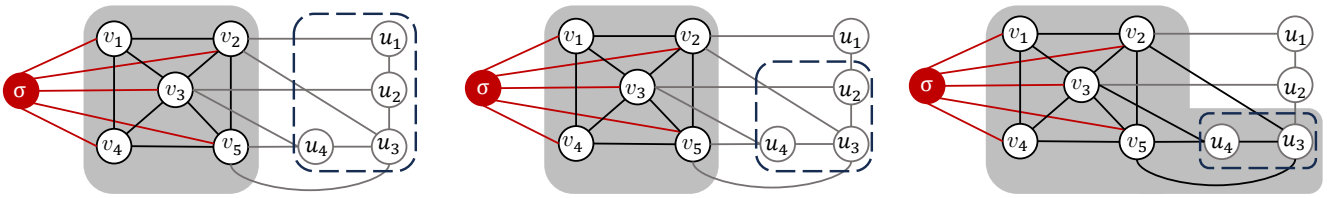
Proof. To prove that $G[S \cup C]$ is k-vertex connected, we need to demonstrate that for any two vertices a and b in $G[S \cup C]$, $a \equiv^k b$. There are three situations: (i) $\forall a \in S$, $\forall b \in S$. Obviously, $a \equiv^k b$ because $G[S]$ is k-vertex connected. (ii) $\forall a \in C$, $\forall b \in S$. $b \equiv^k \sigma$ because b and σ are adjacent; σ is a side-vertex as $\sigma \in S \setminus \mathcal{B}(S)$; and $a \equiv^k \sigma$ based on the conditions. Thus $a \equiv^k b$ (Lemma 3). (iii) $\forall a \in C$, $\forall b \in C$. Because we know $a \equiv^k \sigma$, $b \equiv^k \sigma$, and σ is a side-vertex, we can derive $a \equiv^k b$ directly. The theorem is proven.

□

Moreover, we demonstrate that adding σ will not affect the local connectivity from any vertex $u \in C$ to S because σ has no neighbors outside S, i.e., it does not change the number of vertex-disjoint paths from vertex u to any vertex $v \in S$.

Based on Theorem 1, an exact multiple local expansion approach is proposed which is called Multiple Expansion (ME). Considering S is a k-VCS and C is the candidate neighboring set that we want to explore. ME is based on calculating the maximum flow in $G[S \cup C]$. We use $Max\_Flow^{s \to t}_G$ to represent the maximum flow from s to t in graph G. ME adds a virtual vertex σ, and σ is connected to all vertices in S. In each iteration, ME only remains vertices u in C which satisfies $Max\_Flow^{u \to \sigma}_G \geq k$. ME repeats the above process until every vertex $u \in C$ satisfies $u \equiv^k_{G[S \cup C]} \sigma$.

The pseudocode of ME is shown in Algorithm 1. It uses $\mathcal{G}[\mathcal{S}]$ to denote the set of k-vertex connected subgraphs. For each k-VCS $G[S]$ to be expanded, we add virtual vertex σ to S and connect it to all vertices in S. We then initialize candidate set $C = V \setminus S$ and initialize the expanded vertex set $F = \emptyset$ (Line 4). Next, it computes the maximum flow from each vertex $u \in C$ to σ, i.e., $Max\_Flow^{u \to \sigma}_{G[S \cup C]}$ (Line 6). It adds the vertices whose max-flow value is no less than k to the updated candidate set C^* (Line 7). Then, if C^* is equal to C, i.e., all vertices in C^* have k vertex-disjoint paths to v



(a) (b) (c)

Fig. 5: An example of the Multiple Expansion (ME)

Algorithm 1: ME

```
1  6
   Input  : G(V, E), k, 𝒢[𝒮]
   Output: updated 𝒢[𝒮]
2  foreach G[S] ∈ 𝒢[𝒮] do
3      add virtual vertex σ and edges {(σ, v) | v ∈ S} to G[S] ; do
4          C ← V \ S, F ← ∅;
5          calculate Max_Flow^{u→σ}_{G[S∪C]} foreach u ∈ C;
6          C* ← {u | Max_Flow^{u→σ}_{G[S∪C]} ≥ k};
7          if C* = C then
8              F ← F ∪ C*;
9          else
10             C ← C*;
11             go back to Line 5;
12         G[S] ← G[S ∪ F];
13     while F ≠ ∅;
14     remove σ and adjacent edges from G[S];
15 return 𝒮;
```

in $G[S \cup C^*]$, ME adds C^* to F. Otherwise, C is updated by C^* for later use in the next iteration (Lines 8-12). It adds all vertices in F to S (Line 13) and iterates the above steps until F becomes empty, i.e., S can no longer be expanded. Finally, ME removes the virtual vertex σ and adjacent edge (Line 14).

An example to illustrate the process of the ME algorithm is shown in Figure 5. Without loss of generality, we assume that $k = 3$ and denote the virtual vertex as σ. The k-vertex connected subgraph $G[S] = \{v_1, v_2, v_3, v_4, v_5\}$, and the candidate set C is indicated by the dashed line. First, $C = \{u_1, u_2, u_3, u_4\}$ is considered as the candidate set shown in Figure 5(a). For all vertices in C, we calculate their maximum flow to σ in the graph $G[S \cup C]$, and we obtain $\text{Max\_Flow}^{u_1 \to \sigma}_{G[S \cup C]} = 2$, $\text{Max\_Flow}^{u_2 \to \sigma}_{G[S \cup C]} = 3$, $\text{Max\_Flow}^{u_3 \to \sigma}_{G[S \cup C]} = 3$, and $\text{Max\_Flow}^{u_4 \to \sigma}_{G[S \cup C]} = 3$. Therefore, u_1 is removed from C. After the update, $C = \{u_2, u_3, u_4\}$ as shown in Figure 5(b). We calculate the maximum flow again for each vertex in updated C and obtain $\text{Max\_Flow}^{u_2 \to \sigma}_{G[S \cup C]} = 2$, $\text{Max\_Flow}^{u_3 \to \sigma}_{G[S \cup C]} = 3$, and $\text{Max\_Flow}^{u_4 \to \sigma}_{G[S \cup C]} = 3$. So, u_2 is also removed from C. Similarly, for $C = \{u_3, u_4\}$, we calculate the maximum flow from u_3 and u_4 to σ, the results are both 3. Now the expansion condition of ME is satisfied; we add u_3 and u_4 to S as shown in Figure 5(c).

Theorem 2 (ME is exact)**.** *If the seed set is S and the candidate set is initialized as $C = V \setminus S$, let's denote C^* the vertex set expanded by ME, then ME can insure: (i) $G[S \cup C^*]$ is k-vertex connected and (ii) $\nexists C$, $C^* \subsetneq C$ and $G[S \cup C]$ is k-vertex connected.*

Proof. We prove the two conclusions separately. (i) According to Algorithm 1, $\forall u \in S \cup C^*$, $Max\_Flow^{u \to \sigma}_{G[S \cup C^*]} \geq k$. This is identical to $\forall u \in S \cup C^*$, $u \equiv^k_{G[S \cup C]} \sigma$. From Theorem 1, $G[S \cup C^*]$ is k-connected. (ii) We prove it by contradiction. Assume $C^* \subsetneq C$ and $G[S \cup C]$ is k-vertex connected, then $\forall u \in C$, $\text{Max\_Flow}^{u \to \sigma}_{G[S \cup C']} \geq k$ for all C', where C' is a candidate set updated by ME in a certain iteration and $C \subset C' \subset V \setminus S$. This contradicts that $\exists u \in C$ and $u \notin C^*$, u is removed from C', i.e., $\exists C'$, $\text{Max\_Flow}^{u \to \sigma}_{G[S \cup C']} \leq k$. □

However, setting $C = V \setminus S$ will make the multiple expansion very slow as the calculating max-flow overhead in the entire input graph G is huge. In practice, we can restrict the candidate set C as the h-hop neighbors of S (i.e., $N^h(S)$) to balance efficiency and accuracy.

B. Flow-Based Merging

We then revisit the condition of merging two k-VCSs in the context of ME. We employ a max-flow calculation to determine the vertex connectivity between two k-VCSs and propose a Flow-Based Merging (FBM) method. In order to compute the local connectivity between two k-VCSs, we present the following theorem.

Theorem 3. *Given two k-vertex connected subgraphs $G[S]$ and $G[S']$, adding virtual vertex σ and edges $\{(\sigma, v) \mid v \in S\}$ to $G[S]$, adding τ and edges $\{(\tau, u) \mid u \in S'\}$ to $G[S']$, if $\text{Max\_Flow}^{\sigma \to \tau}_{G[S \cup S']} \geq k$, then $G[S \cup S']$ is k-vertex connected.*

Proof. The proof is similar to Theorem 1. We still consider three situations for any two vertices a, b in $G[S \cup S']$: (i) $\forall a \in S, b \in S$ (ii) $\forall a \in S', b \in S'$. In Cases (i) and (ii), $a \equiv^k b$ obviously because both $G[S]$ and $G[S']$ are k-vertex connected. (iii) $\forall a \in S, b \in S'$. We know $\text{Max\_Flow}^{\sigma \to \tau}_{G[S \cup S']} \geq k$ (i.e., $\sigma \equiv^k \tau$) and $a \equiv^k \sigma$, then $b \equiv^k \tau$. Besides, σ and τ are *side-vertices*, thus $a \equiv^k b$. The theorem is proven.

□

Based on Theorem 3, we propose a Flow-Based Merging (FBM) method. Given two k-VCSs $G[S]$ and $G[S']$, FBM first adds two virtual vertices σ and τ to $G[S]$ and $G[S']$ respectively. Let σ be adjacent to all vertices in S and τ be adjacent to all vertices in S'. If $Max\_Flow^{\sigma \to \tau}_{G[S \cup S']}$, FBM merges $G[S]$ and $G[S']$ and removes the virtual vertices.

The pseudocode of the FBM is given in Algorithm 2. For each k-VCS $G[S]$, we traverse the other k-VCSs for merging. In detail, firstly it checks the size of the intersection between S

Algorithm 2: FBM

Input : $G(V,E), k, \mathcal{G}[\mathcal{S}]$
Output: updated $\mathcal{G}[\mathcal{S}]$

1 **foreach** $G[S] \in \mathcal{G}[\mathcal{S}]$ **do**
2 **foreach** $G[S'] \in \mathcal{G}[\mathcal{S}]$ **do**
3 **if** $| S \cap S' | \geq k$ **then**
4 put $G[S \cup S']$ into $\mathcal{G}[\mathcal{S}]$;
5 delete $G[S]$ and $G[S']$ from $\mathcal{G}[\mathcal{S}]$;
6 break and traverse the next $G[S]$;
7 **else**
8 add σ and edges $\{(\sigma, v) \mid v \in S\}$ to $G[S]$, add τ and edges $\{(\tau, u) \mid u \in S'\}$ to $G[S']$;
  // $\sigma \in S \setminus \mathcal{B}(S)$, $\tau \in S' \setminus \mathcal{B}(S')$
9 calculate $Max\_Flow^{\sigma \to \tau}_{G[S \cup S']}$;
10 **if** Max_Flow$^{\sigma \to \tau}_{G[S \cup S']} \geq k$ **then**
11 delete σ, τ and the adjacent edges from $G[S \cup S']$;
12 put $G[S \cup S']$ into $\mathcal{G}[\mathcal{S}]$;
13 delete $G[S]$ and $G[S']$ from $\mathcal{G}[\mathcal{S}]$;
14 break and traverse the next $G[S]$;
15 **return** $\mathcal{S}$;

and S'. If it is greater than k, $G[S \cup S']$ is k-vertex connected (Line 3). In this case, FBM adds $G[S \cup S']$ to $\mathcal{G}[\mathcal{S}]$ (Line 4) and remove $G[S]$ and $G[S']$ from $\mathcal{G}[\mathcal{S}]$ (Line 5). Then, it moves on to the next k-VCS (Line 6).

Secondly, if the intersection size is not greater than k, FBM adds a virtual vertex σ to S and makes it adjacent to all vertices in S, the same with S' by adding τ (Line 8). FBM then computes the $Max\_Flow^{\sigma \to \tau}_{\mathcal{G}[S \cup S']}$ between σ and τ (Line 9). If the $Max\_Flow^{\sigma \to \tau}_{\mathcal{G}[S \cup S']} \geq k$, it means that $G[S \cup S']$ is k-vertex connected. It then proceeds to the next k-vertex connected subgraph and continues the loop (Lines 10-14). FBM repeats the above process until any k-VCS pairs cannot be merged.

V. More Efficient Multiple Expansion

In the previous section, we introduced the exact expansion method ME. However, ME is inefficient in practice due to computing maximum flows frequently. In this section, we propose a more efficient expansion algorithm named Ring-based Multiple Expansion (RME) and a more efficient seeding algorithm.

A. Ring-based Multiple Expansion

In ME, the computation time of the maximum flow is influenced by selection C. The larger the C, the more accurate the expansion and the higher the calculation time cost. We design a ring-based multiple expansion algorithm (RME) that provides both good accuracy and efficiency.

Given a seed subgraph S, we classify the vertices in $\mathcal{B}(\bar{S})$ into K sets according to the vertex's number of neighbors in S. That is $\mathcal{B}(\bar{S})$ is classified into k sets, where $C_r = \{u \mid u \in \mathcal{B}(\bar{S})$ and $|N_S(u)| = r\}$ for $1 \leq r \leq k-1$; and $C_k = \{u \mid |u \in \mathcal{B}(\bar{S})$ when $|N_S(u)| \geq k\}$. Then a theorem for ring-based multiple expansion (RME) can be given as follows.

Theorem 4. *Given a k-vertex connected subgraph $G[S]$, if a maximal clique $\mathcal{K} \subset C_r$ satisfies (i) $|\mathcal{K}| \geq k+1-r$ and (ii) $|N_S(\mathcal{K})| \geq k$, then $G[S \cup \mathcal{K}]$ is k-vertex connected..*

Proof. Graph $G[S \cup \mathcal{K}]$ is k-vertex connected if $u \equiv^k_{G[S \cup \mathcal{K}]} v$ for any pair of vertices u, v in $G[S \cup \mathcal{K}]$. Since $G[S]$ has already been k-vertex connected and all vertices in $G[\mathcal{K}]$ are adjacent (Note that the local connectivity between adjacent vertices is defined as $+\infty$), so we only need to prove that $\forall u \in \mathcal{K}$, $\forall v \in S, u \equiv^k_{G[S \cup \mathcal{K}]} v$. (i) As $u \in C_r$, $|N_S(u)| = r$, i.e. u has r neighbors in S. We denote them $\{w_i | 1 \leq i \leq r\}, w_i \in N_S(u)$. According to Menger's Theorem, r vertex-disjoint paths from u to v can be provided through the paths from u to w_i, i.e., $u \to w_i \to$ *some vertices in* $S \to v$, $1 \leq i \leq r$. (ii) Known $|N_{\mathcal{K}}(u)| = k - r$ and $|N_S(\mathcal{K})| = k$, there are another $k-r$ vertex-disjoint paths from u to v, going through $u \to u_i \to w_{r+i} \to$ *some vertices in* $S \to v$, where $u_i \in \mathcal{K}, w_{r+i} \in N_S(u_i)$ and $w_{r+i} \notin N_S(u)$, $1 \leq i \leq k-r$. Further, these r paths in (i) and $k-r$ paths in (ii) can be vertex-disjoint because there exists 1 path from u to each w_i, and $w_i \equiv^k v, 1 \leq i \leq k$. Thus, there are k vertex-disjoint paths from u to v, i.e. $u \equiv^k_{G[S \cup \mathcal{K}]} v$.

□

Based on Theorem 4, we propose the ring-based multiple expansion algorithm (RME). For each k-vertex connected subgraph S, RME considers $\mathcal{B}(\bar{S})$ as the candidate set in one iteration and puts vertices $v \in \mathcal{B}(\bar{S})$ to one of the k groups $C_k, C_{k-1}, ..., C_1$ according to v's number of neighbors in S and expand multiple vertices simultaneously in each group based on the structure between them.

Algorithm 3: RME

Input : $G(V,E), k, \mathcal{G}[\mathcal{S}]$
Output: updated $\mathcal{G}[\mathcal{S}]$

1 **foreach** $G[S] \in \mathcal{G}[\mathcal{S}]$ **do**
2 **do**
3 $C_r = \emptyset, r \in [1,k]$, $F = \emptyset$;
4 **foreach** $u \in \mathcal{B}(\bar{S})$ **do**
5 $r = min(|N(u) \cap \mathcal{B}(S)|, k)$;
6 $C_r \leftarrow C_r \cup u$;
7 **foreach** $u \in C_k$ **do**
8 $F \leftarrow F \cup u$;
9 `UpdateNeighbours`(u);
10 **for** $r = k-1, k-2, ..., 1$ **do**
11 find maximal cliques $\mathcal{MC}$ of size $\geq k-r+1$ in $G[C_r]$;
12 **foreach** $\mathcal{K} \in \mathcal{MC}$ **do**
13 **if** $N_S(\mathcal{K}) \geq k$ **then**
14 $F \leftarrow F \cup \mathcal{K}$;
15 `UpdateNeighbours`(v) **for each** $v \in \mathcal{K}$;
16 $G[S] \leftarrow G[S \cup F]$;
17 **while** $F \neq \emptyset$;
18 **return** $\mathcal{G}[\mathcal{S}]$;
19 **Procedure** `UpdateNeighbours(u)`
20 **foreach** $v \in N_{\mathcal{B}(\bar{S})}(u)$ **do**
21 $r = |N_S(v)|$;
22 remove v from C_r to C_{r+1};
23 **if** $r+1 = k$ **then**
24 $F \leftarrow F \cup v$;
25 `UpdateNeighbours(v)`;

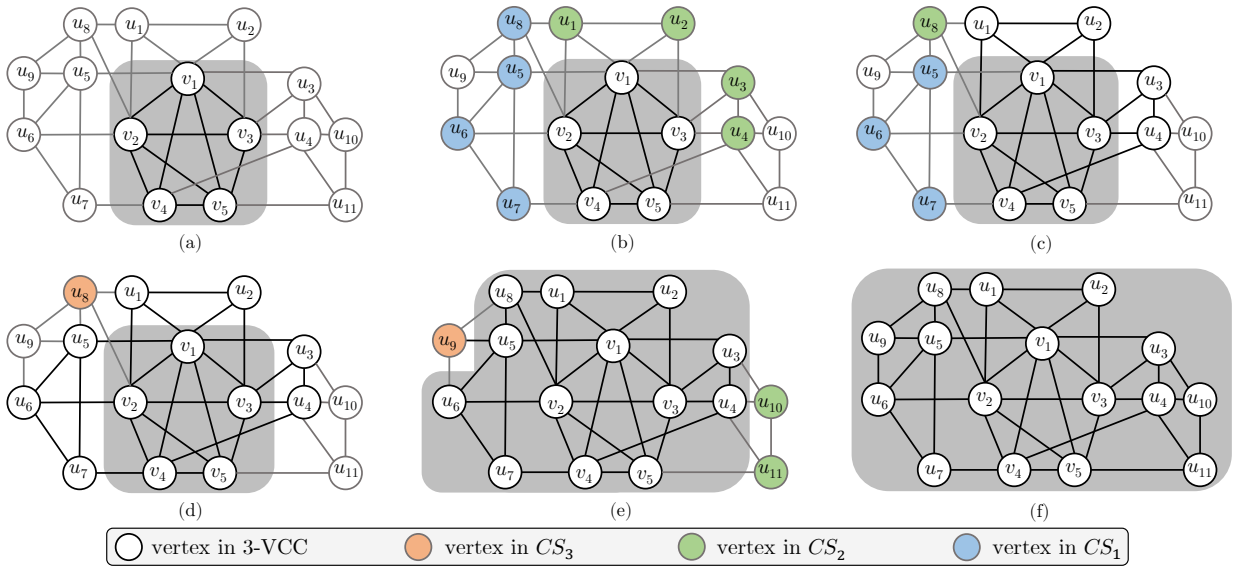


Fig. 6: An example for Ripple Expanding (k=3). C_r means vertices have r neighbors in seed subgraph (Shadow Area).

In Algorithm 3, we give the pseudocode of the RME algorithm. For each k-vertex connected subgraph $G[S]$, RME firstly iterates over all the vertices $u \in \mathcal{B}(\bar{S})$ and classifies them according to the number of neighbors they have in the seed subgraph, i.e., $|N_S(u)|$ (Lines 4-6). Then, RME adds the vertices in C_k to the expanded set F and updates the neighbors of them by calling UpdateNeighbors (Lines 7-9), which we provide explanations later. After that, RME finds the maximal cliques in C_r with the size $\geq k - r + 1$. If the maximal clique $\mathcal{K}$ has not less than k neighbors in S, we can add all the vertices in $\mathcal{K}$ to F and update the information about their neighbors for each vertex $v \in \mathcal{K}$ (Lines 10-15). Finally, we update $G[S]$ to $G[S \cup F]$ (Line 16) and repeat the above process until $F = \emptyset$. In the specific operations implemented by UpdateNeighbours, it iterates over all the neighbors in $\mathcal{B}(\bar{S})$ of input vertex u. It removes them from C_r to C_{r+1}. If $r+1 \geq k$, we add that vertex to the F as well and recursively call UpdateNeighbours(Lines 19-25).

An example is given in Figure 6(a), $\{v_1, v_2, v_3, v_4, v_5\}$ is a seed subgraph to be expanded. At first, we classify vertices in $N(S)$ as shown in Figure 6(b), $C_3 = \emptyset$, $C_2 = \{u_1, u_2, u_3, u_4\}$, and $C_1 = \{u_5, u_6, u_7, u_8\}$. Then, we find maximal cliques in C_2 with size ≥ 2. That is $\mathcal{K}_1 = \{u_1, u_2\}$ and $\mathcal{K}_2 = \{u_3, u_4\}$. Both of them satisfy Theorem 4 that $|N_S(\mathcal{K}_1)| = 3 \geq 3$, $|N_S(\mathcal{K}_2)| = 3 \geq 3$. So $\mathcal{K}_1$ and $\mathcal{K}_2$ are expanded to the seed subgraph, while we update u_8 from C_1 to C_2 as shown in Figure 6(c). However, C_2 only has one vertex u_8, so it can not be expanded. Next we consider C_1 and find $\mathcal{K}_3 = \{u_5, u_6, u_7\}$ as the maximal cliques with size ≥ 3, so we add $\mathcal{K}_3$ to the seed subgraph and update u_8 to C_3 as shown in Figure 6(d), and u_8 can be expanded directly. Therefore all vertices in $nb_1(S)$ are traversed. By updating the seed subgraph, we get 6(e). Repeating the above process, we finally get 6(f). Now, we successfully expand 11 vertices into the seed subgraph by RME, while UE can not expand anyone of them.

B. Quickly Seeding k-VCS

We use the k-vertex connected subgraph as seeds, as we introduced its definition before. Notice that k-VCS does not require the subgraph to be maximal. Here, we propose two approaches to extract k-VCS efficiently. The first one is to find the maximal cliques with size $\geq k+1$. The second is to run BFS k times to find k-VCS.

Our First seeding approach is based on the fact that a clique with $\geq k+1$ vertices is k-vertex connected. We can iteratively find r-cliques as a seed by increasing r, where r represents the size of a clique. However, there is a more efficient approach: we find maximal cliques with size $\geq k+1$, which can be achieved by Bron–Kerbosch algorithm in $O(3^{\frac{n}{3}})$. Although the algorithm's complexity is high, it finds the maximal cliques efficiently in practice. On one hand, maximal cliques can reduce the seed redundancy phenomenon that most seeds are finally combined into one k-VCS; On the other hand, the maximal cliques can effectively reduce the time required for subsequent operations because plenty of vertices have been added in the seeding stage.

The second seeding approach is running BFS k times to find k-VCS. We introduce the following lemma:

Lemma 4. *[27][36] Let $G(V, E)$ be an undirected graph, and let n denote the number of vertices. Let k be a positive integer. For $i = 1, 2, \ldots, k$, let E_i be the edge set of a breadth first search forest F_i in the graph $G_{i-1} = (V, E - (E_1 \cup E_2 \cup \ldots \cup E_{i-1}))$, where $G_0 = G$. Then any connected component in F_k is k-vertex connected subgraph.*

The proof can be found in [36]. Now, we can find more k-vertex connected subgraphs in $O(k \cdot (n+m))$ times. Finally, for all vertices that do not belong to any k-VCS, we use LkVCS to perform the search. To sum up, we propose a quickly finding k-vertex connected subgraphs algorithm QkVCS.

Algorithm 4: QkVCS

```
Input : G(V, E), k
Output: The set of seed subgraphs G[S]
1  G[S] ← kBFS(G, k);
   // Find k-VCS by running BFS k time [36]
2  G[S] ← G[S] ∪ BK-MCQ(V, k + 1);
   // collect maximal cliques with size ≥ k + 1
      as k-VCS
3  foreach v ∈ V do
4      if ∄G[S], G[S] ∈ G[S] s.t. v ∈ S then
5          CandMaintain[v] ← 0;
6  foreach v ∈ CandMaintain in non-decreasing order w.r.t. deg
   (v) do
7      if CandMaintain[v] = 0 then
8          G[C] ← LkVCS(G, k, v);
           // Find k-VCS including v [25]
9          if C ≠ ∅ then
10             G[S] ← G[S] ∪ G[C];
11             foreach u ∈ C do CandMaintain[v] ← 1;
12 return G[S];
```

The pseudocode of QkVCS is shown in Algorithm 4. First, QkVCS finds k-vertex connected subgraphs by running BFS k times and adds them to the seed subgraphs set $\mathcal{G}[\mathcal{S}]$ (Line 1). After that, it finds all maximal cliques with size $\geq k+1$ in the graph (Line 2). Then, it adds all the vertices not added into any k-VCS before to $CandMaintain$ and marks them unvisited (Lines 3-7). Finally, it visits all the vertices in the $CandMaintain$ according to the non-decreasing order of their vertex degree. If a vertex v has not been visited, QkVCS find local k-vertex connected subgraph containing v by calling the LkVCS algorithm (Lines 8-16).

C. The RIPPLE Algorithm

Now, we give the overall algorithm RIPPLE, which consists of three algorithms: QkVCS, RME, and FBM. The pseudocode of RIPPLE is shown in Algorithm 5.

Algorithm 5: RIPPLE(G, k)

```
Input : G(V, E), k
Output: G[R]: the set of k-VCCs
1  G[R] ← ∅; G[S] ← ∅; G [S'] ← ∅ ;
2  Find the k-core G^k of G;
3  G[S] ← QkVCS(G^k, k);
4  while G[S'] ≠ G[S] do
5      G[S'] ← G[S];
6      G[S] ←FBM(G^k, k, G[S]);
7      G[S] ←RME(G^k, k, G[S]);
8  G[R] ← G[R] ∪ G[S];
9  return G[R];
```

First, it gets the k-core G^k of the input graph G (Line 2). Then, it calls QkVCS to generate k-VCSs as seed subgraphs (Line 3). After that, QkVCS repetitively alternates between calling FBM and RME until no k-VCS can be further expanded or merged (Lines 4-7). It is worth noting that we first use the merging algorithm to combine seeds. This is because calling the merging part firstly helps avoid redundant computations during the expanding phase as some seeds can be merged. Finally, QkVCS returns k-VCCs in graph G.

Complexity Analysis. In the seeding phase, QkVCS calls BFS k times to find side groups in $O(k(n+m))$. Next, it calls Bron– Kerbosch algorithm to find maximal cliques with size $\geq k+1$, which costs $\mathrm{O}\left(dn3^{d/3}\right)$, where d is the degeneracy of the graph [9]. Finally, QkVCS invokes the LkVCS algorithm for each vertices that is not included in any seed subgraph, whose time complexity is $O\left(\alpha\left|\bar{m}\right|\right)$, where α is a threshold for enumerating combinations [25]. In fact, the number of remaining vertices is much less than n, which is denoted by $\tilde{n}$. For simplicity, We use $\bar{n}$ and $\bar{m}$ to represent the average number of vertices and edges of the seed subgraphs, respectively. Thus, the time complexity for QkVCS is $O(k(n+m)+dn3^{d/3}+\alpha\tilde{n}\left|\bar{m}\right|)$. In the merging phase, the time complexity of FBM is $O(|S|^2(\bar{n}\bar{m}^2))$, where $|S|$ is the number of seed subgraphs. In the expanding phase, the time complexity for calculating the maximal cliques is $O(|C|^{\frac{|C|}{3}})$, where set C is the one-hop neighbors of a seed subgraph, thus $|C| << n$. Determining and updating the neighbor set takes $O(\bar{m})$ time. Assuming I_2 iterations are needed, it runs in $O(I_2(|C|^{\frac{|C|}{3}}+\bar{m}))$ for one seed subgraph. Thus, the time complexity of RME is $O(|S|I_2(|C|^{\frac{|C|}{3}}+\bar{m}))$.

VI. EXPERIMENTS

In this section, we present our experimental results. All experiments are conducted on a machine with Intel(R) Xeon(R) CPU E5-2667 3.20GHz and 128GB memory running Linux. All algorithms are implemented in C++ and compiled with GCC with the -O3 optimization[1]. We achieve parallelization by OpenMP and utilize the compiler option -fopenmp. Algorithms are terminated if their running time exceeds 10^5 seconds.

TABLE II: Statistics of the real-world graphs ($\bar{d}$: average degree, k_{max}: maximum k such that a k-VCC exists)

Dataset	$\|V(G)\|$	$\|E(G)\|$	$\bar{d}$	k_{max}
ca-CondMat	23,133	93,497	8.08	25
uk-2005	129,632	11,744,049	180.19	499
arabic-2005	163,598	1,747,269	21.36	101
sc-shipsec	179,104	2,200,076	24.56	25
ca-citeseer	227,320	814,134	7.16	86
ca-dblp	317,080	1,049,866	6.62	113
ca-MathSciNet	332,689	820,644	4.93	24
it-2004	509,338	7,178,413	28.19	431
cit-patent	3,774,768	16,518,947	8.75	64
socfb-konect	59,216,211	92,522,017	3.12	16

Datasets. We use ten real-world graphs with different properties for evaluating the algorithms, which are downloaded from Network Repository[2] and Stanford Network Analysis Project[3]. Statistics of the graphs are shown in Table II. $|V(G)|$ denotes the number of vertices, and $|V(G)|$represents the number of edges. The average degree and the maximum k such that a k-VCC exists are also shown in the second last column and the last column, respectively.

Algorithms. We implement and compare four algorithms:

- VCCE-TD: Top-down algorithm based on [36].

[1]Our code is available at: https://github.com/Elssky/RIPPLE.

[2]https://networkrepository.com/

[3]http://snap.stanford.edu/data/index.html

- VCCE-BU: Bottom-up algorithm based on [25].
- RIPPLE: Algorithm composed of QkVCS, FBM, and RME (Section V).
- RIPPLE-ME: Replace RME with ME (Section IV).

Parameter Setting. For the dataset in Table II, we set the default k ($k \geq 5$) to be the minimum value that allows the VCCE-TD algorithm to be completed within 10^5 seconds. We set $\alpha := 10^3$ for VCCE-BU, RIPPLE and RIPPLE-ME. For RIPPLE-ME, we consider 1-hop neighbor of the seed for each iteration in RME.

Metrics. As discussed in Section 5, RIPPLE is a heuristic algorithm for extracting k-VCCs. There is no guarantee that it will always be correct. The VCCE-TD mines the exact k-VCCs, so we need to measure the accuracy of RIPPLE and VCCE-BU compared to VCCE-TD. For this purpose, We employ two metrics from [35] to evaluate the accuracy of the algorithms, F_{same} and J_{Index}, which are as follows.

Cross Common Fraction (F_{same}) compares each pair of k-VCCs, where one k-VCC is from the detected result, and the other is from the real result, to identify the maximal shared parts. Formally, it is defined as follows:

$$F_{same} = \frac{1}{2}\sum_{i=1}^{N}\max_{j}\left|S_i \cap S'_j\right| + \frac{1}{2}\sum_{j=1}^{N'}\max_{i}\left|S_i \cap S'_j\right| \quad (1)$$

where N and N' are the quantities of detected and real k-VCCs respectively, and S_i and S'_j are the i-th detected k-VCC and j-th real k-VCC respectively.

Jaccard Index (J_{Index}) is known for its sensitivity in handling cases where vertices from multiple communities in one result merge into a single community in another result, thereby overcoming the slight variability of the F_{same}. It can be defined as

$$J_{Index} = \frac{|V(S_t)|}{|V(S_t)| + |V(S_{f1})| + |V(S_{f2})|} \quad (2)$$

where $|V(S_t)|$ stands for the number of vertex pairs that are respectively classified into the same k-VCC in both results, $|V(S_{f1})|$ stands for the number of vertex pairs appearing in the same k-VCC in the algorithm-produced results, but in different k-VCCs in the truth, and $|V(S_{f2})|$ vice versa. However, Jaccard Index cannot detect missing communities.

A. Efficiency Evaluation

We evaluate the running time of RIPPLE on all the datasets and compare it with VCCE-TD and VCCE-BU. The results are shown in Figure 7. The VCCE-TD curves that fails to end within 10^5 seconds are not plotted. RIPPLE runs consistently faster than VCCE-TD and VCCE-BU. On uk-2005 with k=100, RIPPLE completes in 1,740 seconds, while VCCE-TD runs over 80,000 seconds, and VCCE-BU runs about 10,000 seconds. RIPPLE achieves a speedup of 46 times compared to VCCE-TD and 6 times compared to VCCE-BU.

VCCE-BU is faster than VCCE-TD in most cases except for it-2004, where over 30,000 seconds is spent on finding seed subgraphs in LkVCS. Furthermore, we vary the value of k to compare the running time changes among the three algorithms. As we can observe, with the increase of k, the running time of all three algorithms generally decreases, which is attributed to the reduction of the k-core. However, on uk-2005 and sc-shipsec, VCCE-TD has an increase of time due to the pruning efficiency is reduced [36]. On most datasets, RIPPLE and VCCE-BU exhibit a similar decreasing trend, as both utilize the seed expansion framework, which is also in line with our complexity analysis. Note that on ca-MathSciNet with k=13, VCCE-TD outperforms VCCE-BU and RIPPLE. We find it is because the seeding takes the major time of bottom-up framework in this setting, while there are few k-VCCs in the graph for top-down framework to traverse.

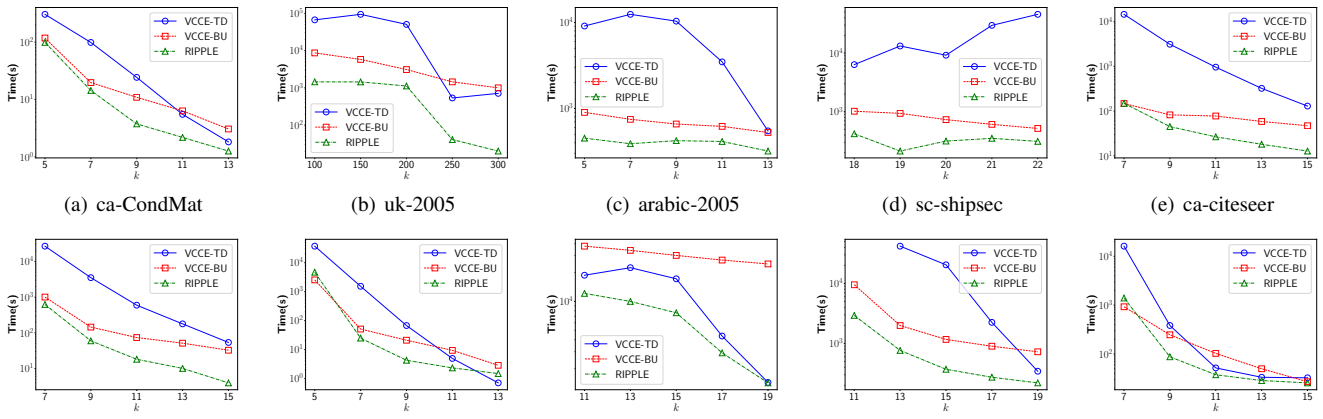
B. Accuracy Evaluation

In this section, we compare the accuracy of two heuristic algorithms, RIPPLE and VCCE-BU, on all datasets, with exact results obtained through VCCE-TD. Due to space limitations, we present results for the top three k values for each dataset. The results are shown in Table III.

TABLE III: Accuracy comparison

Dataset	k	F_{same}		J_{Index}	
		RIPPLE	VCCE-BU	RIPPLE	VCCE-BU
ca-CondMat	5	**97.97%**	91.62%	**90.33%**	87.81%
	7	**96.66%**	86.87%	**85.23%**	68.28%
	9	**96.61%**	87.15%	**84.78%**	66.53%
uk-2005	100	**100.00%**	100.00%	**100.00%**	100.00%
	150	**100.00%**	100.00%	**100.00%**	100.00%
	200	**100.00%**	100.00%	**100.00%**	100.00%
arabic-2005	5	**99.92%**	98.66%	**98.64%**	96.01%
	7	**100%**	98.94%	**99.99%**	99.31%
	9	**99.99%**	98.96%	**99.97%**	99.56%
sc-shipsec	18	**95.35%**	63.57%	**95.10%**	4.21%
	19	**92.12%**	64.55%	**89.50%**	2.43%
	20	**86.88%**	65.67%	**81.65%**	2.52%
ca-citeseer	7	**96.59%**	83.48%	**54.49%**	14.13%
	9	**98.08%**	90.06%	**61.82%**	60.09%
	11	**98.38%**	93.86%	**52.46%**	41.36%
ca-dblp	7	**97.52%**	86.10%	**84.46%**	65.59%
	9	**95.46%**	84.60%	**66.60%**	43.82%
	11	**97.14%**	88.70%	**59.39%**	20.57%
ca-MathSciNet	5	**97.17%**	85.37%	**91.44%**	73.73%
	7	**92.48%**	65.81%	**79.04%**	19.36%
	9	**91.68%**	68.56%	**59.68%**	12.09%
it-2004	11	**100.00%**	99.89%	**100.00%**	99.99%
	13	**100.00%**	100.00%	**100.00%**	100.00%
	15	**100.00%**	100.00%	**100.00%**	100.00%
cit-patent	13	**89.21%**	76.39%	**71.16%**	43.20%
	15	**83.44%**	76.31%	**44.90%**	34.30%
	17	**81.65%**	73.54%	**43.66%**	28.81%
socfb-konect	7	**91.12%**	55.07%	**81.17%**	7.68%
	9	**91.69%**	54.04%	**82.03%**	6.13%
	11	**75.95%**	52.84%	**36.28%**	9.57%

We observe that RIPPLE outperforms VCCE-BU in terms of accuracy based on both F_{same} and J_{Index}. The difference between the two algorithms is more pronounced in the J_{Index}. For example, on ca-dblp and ca-MathSciNet, the J_{Index} of RIPPLE is at least 20% higher than that of VCCE-BU. On sc-shipsec and scofb-konect, J_{Index} of VCCE-BU is even less than 10%. We investigate the reason and find this is mainly due to NBM's misjudgment as shown in Figure 3(a). Additionally, RIPPLE achieves F_{same} scores above 80% on all datasets, reaching 100% on uk-2005 and it-2004. We also observe a



(a) ca-CondMat (b) uk-2005 (c) arabic-2005 (d) sc-shipsec (e) ca-citeseer

(f) ca-dblp (g) ca-MathSciNet (h) it-2004 (i) cit-patent (j) socfb-konect

Fig. 7: Effect of k on the running time of VCCE-TD, VCCE-BU, RIPPLE

phenomenon of decreasing accuracy with increasing k values for both algorithms on ca-condmat, ca-MathSciNet, and cit-patent. This decrease can be attributed to an increase in the proportion of missed vertices during the expanding process.

TABLE IV: Comparison between RIPPLE and RIPPLE-ME

Dataset	k	RIPPLE			RIPPLE-ME		
		Time (s)	F_{same}	J_{Index}	Time (s)	F_{same}	J_{Index}
ca-CondMat	5	98.69	97.97%	90.33%	30014.60	99.75%	99.54%
	7	14.32	96.66%	85.23%	4205.27	99.19%	97.20%
	9	3.75	96.61%	84.78%	561.24	97.82%	91.50%
ca-dblp	7	623.55	97.52%	84.46%	time out	-	-
	9	59.71	95.46%	66.60%	23686.59	98.71%	89.64%
	11	18.16	97.14%	59.39%	499.72	98.58%	80.94%
ca-MathSciNet	5	4660.76	97.17%	91.44%	time out	-	-
	7	24.72	92.48%	79.04%	63880.08	99.32%	97.80%
	9	4.23	91.68%	59.68%	32.43	94.12%	67.52%
cit-patent	13	763.13	89.21%	71.16%	time out	-	-
	15	369.84	83.44%	44.90%	15211.05	87.19%	55.17%
	17	272.41	81.65%	43.66%	4272.34	94.23%	82.27%

In Table IV, we compare the performance of RIPPLE and RIPPLE-ME on four datasets. In line with theoretical expectations, RIPPLE-ME achieves higher accuracy than RIPPLE consistently. However, RIPPLE-ME is much slower as it takes more time on the max-flow calculation. For instance, on ca-dblp with $k = 7$, RIPPLE-ME cannot complete within 10^5 seconds, whereas RIPPLE runs in 10^3 seconds. Besides, as k decreases, the time taken by RIPPLE increases slowly, but the time taken by RIPPLE-ME increases much more rapidly.

C. Memory Usage Comparison

The main memory usage of the algorithms is demonstrated in Figure 8. RIPPLE and VCCE-BU consume significantly less memory compared to VCCE-TD on most datasets. For instance, on the ca-citeseer dataset, VCCE-TD occupies 24GB of memory, while RIPPLE and VCCE-BU only require approximately 100MB of memory. This discrepancy is caused by the utilization of graph partitioning in VCCE-TD to find k-VCCs, which entails storing all the subgraphs generated during each partitioning step in memory. Conversely, RIPPLE and VCCE-BU only retain the initial seed subgraphs and conduct expansions on them, this progress requires much lower memory. On socfb-konect, the space occupied by the two frameworks is close. This is because there is a large k-VCC in the graph, and the process of maintaining and updating its seed subgraph takes up most of the memory. When comparing RIPPLE and VCCE-BU, despite RIPPLE incorporating multi-vertex expansion and max-flow-based merging, both algorithms display comparable memory usage within the same order of magnitude. Remarkably, on uk-2005, RIPPLE's memory consumption is lower, which is attributed to QkVCS, by reducing the redundant seed subgraphs generation.

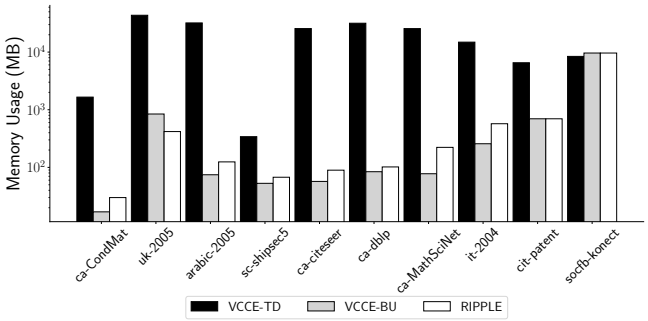


Fig. 8: Memory Usage: RIPPLE v.s. state-of-the-arts

D. Ablation Study

We conduct an ablation study on RIPPLE by replacing each module by the corresponding module in VCCE-BU. The result is shown in Table V. Firstly, we observed that RIPPLE has the highest accuracy on all datasets, indicating that each module can improve the accuracy. Secondly, we find that QkVCS contributes the most on the acceleration effect, while FBM and RME may trade time for accuracy. FBM and RME both contribute on improving the k-VCC detection accuracy.

E. Others

Time Proportion Analysis. We analyze the specific impact of each building block within RIPPLE on the overall running time once the graph is loaded. Figure 9 reports the proportion

TABLE V: Ablation Study of RIPPLE

Dataset	k	RIPPLE			RIPPLE-noQkVCS			RIPPLE-noFBM			RIPPLE-noRME		
		Time (s)	F_{same}	J_{Index}	Time (s)	F_{same}	J_{Index}	Time (s)	F_{same}	J_{Index}	Time (s)	F_{same}	J_{Index}
socfb-konect	9	**87.55**	**91.69%**	**82.03%**	232.88	54.38%	6.72%	95.44	91.69%	82.03%	100.36	90.13%	78.65%
ca-dblp	7	623.55	**97.52%**	**84.46%**	701.06	89.05%	79.41%	2023.19	97.48%	84.31%	**516.01**	96.76%	80.61%
sc-shipsec	18	420.73	**95.35%**	**95.10%**	1121.95	85.57%	94.49%	550.16	85.57%	94.49%	**178.03**	94.74%	94.57%
uk-2005	100	1740.39	**100.00%**	**100.00%**	11580.30	100.00%	100.00%	**752.96**	100.00%	100.00%	1590.99	100.00%	100.00%
it-2004	15	7617.46	**100.00%**	**100.00%**	32227.26	100.00%	100.00%	**4437.47**	100.00%	100.00%	6717.07	100.00%	100.00%

**RIPPLE-noQKVCS: Replacing QkVCS by LkVCS in RIPPLE; RIPPLE-noFBM: Replacing FBM by NBM in RIPPLE; RIPPLE-noRME: Replacing RME by UE in RIPPLE;*

of each part in the total running time: QkVCS (Algorithm 4), FBM (Algorithm 2) and RME (Algorithm 3 on all datasets. The FBM and RME is the most computationally expensive part (more than 90%) on most datasets. For cit-patent, the time used by verifying QkVCS takes the majority.

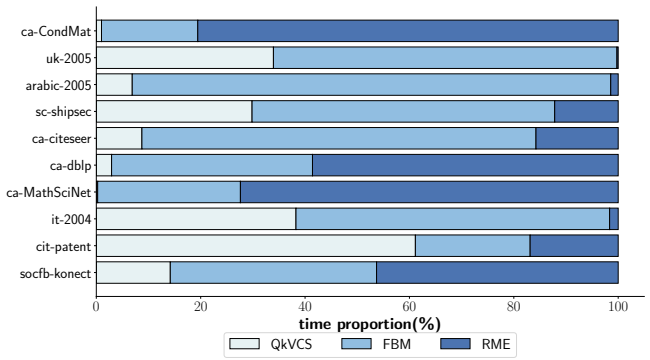


Fig. 9: Proportion of RIPPLE's each part in total running time

Evaluate Seeding Efficiency. In Table VI, we show the pruning effects of two algorithms: kBFS and BK-MCQ. On all datasets, QkVCS achieves a pruning efficiency of over 80% in total and a speedup of more than 4 times. BK-MCQ plays a more significant role in pruning and covers all vertices in uk-2005. The sum of vertices covered by kBFS and BK-MCQ exceeds the total number of covered vertices. This indicates that some vertices can be found as corresponding seeds in both algorithms. Considering the inherent nature of overlapping in k-VCSs, this phenomenon is reasonable.

TABLE VI: Evaluating QkVCS Efficiency

Dataset	k	Coverage ratio of vertices in G^k			Speed up
		kBFS	BK-MCQ	Total	
ca-CondMat	5	65.82%	80.31%	93.37%	12.0
	7	51.50%	73.81%	87.21%	8.6
	9	38.54%	69.00%	81.44%	5.4
uk-2005	100	40.24%	100.00%	100.00%	15.3
	150	4.51%	100.00%	100.00%	6.9
	200	2.36%	100.00%	100.00%	4.1
arabic-2005	5	79.17%	99.65%	99.68%	19.8
	7	72.48%	99.87%	99.87%	22.2
	9	66.37%	99.82%	99.82%	10.1
ca-citeseer	7	35.45%	93.88%	95.23%	8.3
	9	30.51%	97.09%	97.43%	10.7
	11	28.02%	98.92%	98.93%	15.3

**G^k is the k-core of input graph G; Speed up is compared to LkVCS*

Degree of Parallelism. Figure 10 shows the efficiency of RIPPLE running in parallel is dependent on the graph structure. The parallelization of the RIPPLE is accomplished in three steps. **At first**, in QKVCS, we use a paralleled BK algorithm to find maximal cliques and traverse vertices in $CandMaintain$ in parallel to find the k-vertex connected subgraphs. **Secondly**, for FBM, the pairwise combinations of seed subgraphs are traversed in parallel to determine whether they can be merged. **Thirdly**, RME is parallelized to expand each seed subgraph independently. Specifically, according to Figure 9, a smaller proportion of QkVCS and FBM leads to higher parallelism. The reason is that there is data contention for accessing seed subgraphs in these two algorithms, and resolving this contention through locking results in decreased parallel efficiency. We also observe that using 16 threads may lead to slower acceleration compared to using 8 threads due to the above phenomenon.

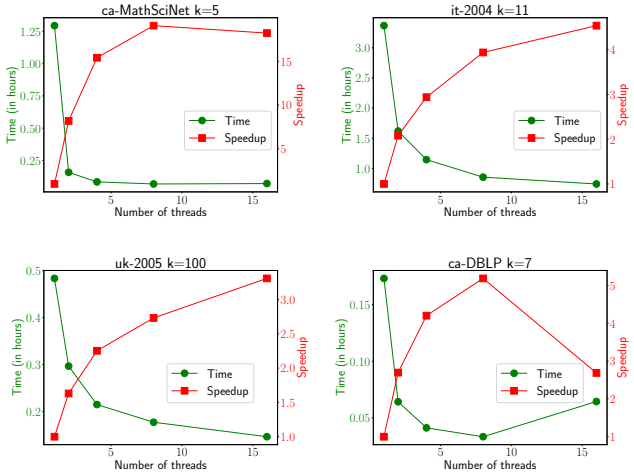


Fig. 10: Running time and speedup as a function of the number of threads for RIPPLE

VII. CONCLUSION

In this paper, we reveal the reason for the inaccuracy and inefficiency of existing VCCE-BU methods. We reformulate the local expansion problem of VCCE-BU as a multiple vertex collaborative expansion problem and propose an exact Multiple Expansion algorithm (ME) and an efficient Ring-based ME method, i.e., RME. We also propose a Flow-based Merging algorithm (FBM) and a Maximal Clique and BFS-based quick seeding algorithm: QkVCS. To seek accurate and efficient VCCE-BU, we propose RIPPLE, a parallel implementation of QkVCS and RME. We conduct extensive experiments using ten real datasets. The results show that RIPPLE is not only much more accurate but also a magnitude faster than the state-of-the-art VCCE-BU method. The proposed ME framework also provides the flexibility to select the local search step size based on the user's preference for accuracy or efficiency.

REFERENCES

[1] B. Balasundaram, S. Butenko, and I. V. Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011.

[2] J. A. Barnes. Graph theory and social networks: A technical comment on connectedness and connectivity. *Sociology*, 3(2):215–232, 1969.

[3] A. K. Baruah. Traffic control problems using graph connectivity. *International Journal of Computer Applications*, 86(11), 2014.

[4] J. Baumes, M. K. Goldberg, M. S. Krishnamoorthy, M. Magdon-Ismail, and N. Preston. Finding communities by clustering a graph into overlapping subgraphs. In *IADIS AC*, 2005.

[5] P. Bedi and C. Sharma. Community detection in social networks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(3):115–135, 2016.

[6] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang. Efficiently computing k-edge connected components via graph decomposition. In *Proceedings of the 2013 ACM SIGMOD international conference on management of data*, pages 205–216, 2013.

[7] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *2011 IEEE 27th International Conference on Data Engineering*, pages 51–62. IEEE, 2011.

[8] J. Cheriyan, M.-Y. Kao, and R. Thurimella. Scan-First Search and Sparse Certificates: An Improved Parallel Algorithm for k -Vertex Connectivity. *SIAM Journal on Computing*, 22(1):157–174, Feb. 1993.

[9] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs, 2011.

[10] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518, 1975.

[11] S. Forster, D. Nanongkai, L. Yang, T. Saranurak, and S. Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2046–2065. SIAM, 2020.

[12] H. N. Gabow. Using expander graphs to find vertex connectivity. *Journal of the ACM (JACM)*, 53(5):800–844, 2006.

[13] F. Gasparetti, G. Sansonetti, and A. Micarelli. Community detection in social recommender systems: a survey. *Applied Intelligence*, 51:3975–3995, 2021.

[14] M. Henzinger, S. Krinninger, and V. Loitzenbauer. Finding 2-edge and 2-vertex strongly connected components in quadratic time. In *International Colloquium on Automata, Languages, and Programming*, pages 713–724. Springer, 2015.

[15] M. R. Henzinger, S. Rao, and H. N. Gabow. Computing vertex connectivity: new bounds from old techniques. *Journal of Algorithms*, 34(2):222–250, 2000.

[16] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on computing*, 2(3):135–158, 1973.

[17] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1311–1322, 2014.

[18] X. Jia, D. Kim, S. Makki, P.-J. Wan, and C.-W. Yi. Power assignment for k-connectivity in wireless ad hoc networks. *Journal of Combinatorial Optimization*, 9:213–222, 2005.

[19] P. Kindlmann and F. Burel. Connectivity measures: a review. *Landscape ecology*, 23:879–890, 2008.

[20] I. M. Kloumann, J. Ugander, and J. M. Kleinberg. Block models and personalized pagerank. *Proceedings of the National Academy of Sciences*, 114:33 – 38, 2016.

[21] C. Komusiewicz. Multivariate algorithmics for finding cohesive subnetworks. *Algorithms*, 9(1):21, 2016.

[22] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11:033015, 2008.

[23] C. Lee, F. Reid, A. F. McDaid, and N. J. Hurley. Detecting highly overlapping community structure by greedy clique expansion. In *Knowledge Discovery and Data Mining*, 2010.

[24] N. Li and J. C. Hou. Flss: a fault-tolerant topology control algorithm for wireless networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 275–286, 2004.

[25] Y. Li, G. Wang, Y. Zhao, F. Zhu, and Y. Wu. Towards k-vertex connected component discovery from large networks. *World Wide Web*, 23(2):799–830, Mar. 2020.

[26] Y. Li, Y. Zhao, G. Wang, F. Zhu, Y. Wu, and S. Shi. Effective k-vertex connected component detection in large-scale networks. In *Database Systems for Advanced Applications: 22nd International Conference, DASFAA 2017, Suzhou, China, March 27-30, 2017, Proceedings, Part II 22*, pages 404–421. Springer, 2017.

[27] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

[28] D. Nanongkai, T. Saranurak, and S. Yingchareonthawornchai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 241–252, 2019.

[29] J. Pattillo, A. Veremyev, S. Butenko, and V. Boginski. On the maximum quasi-clique problem. *Discrete Applied Mathematics*, 161(1-2):244–257, 2013.

[30] J. Pattillo, N. Youssef, and S. Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.

[31] S. Rahiminejad, M. R. Maurya, and S. Subramaniam. Topological and functional comparison of community detection algorithms in biological networks. *BMC bioinformatics*, 20(1):1–25, 2019.

[32] N. N. Sein. Overlapping community detection using local seed expansion. *Int. J. Comput.*, 37(1):27–34, 2020.

[33] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[34] A. Veremyev, O. A. Prokopyev, V. Boginski, and E. L. Pasiliao. Finding maximum subgraphs with relatively large vertex connectivity. *European Journal of Operational Research*, 239(2):349–362, 2014.

[35] M. Wang, C. Wang, J. X. Yu, and J. Zhang. Community detection in social networks: an in-depth benchmarking study with a procedure-oriented framework. *Proceedings of the VLDB Endowment*, 8(10):998–1009, 2015.

[36] D. Wen, L. Qin, Y. Zhang, L. Chang, and L. Chen. Enumerating k-vertex connected components in large graphs. In *2019 IEEE 35th international conference on data engineering (ICDE)*, pages 52–63. IEEE, 2019.

[37] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using seed set expansion. *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013.

[38] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using neighborhood-inflated seed expansion. *IEEE Transactions on Knowledge and Data Engineering*, 28:1272–1284, 2015.

[39] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: on free rider effect and its elimination. *Proceedings of the VLDB Endowment*, 8(7):798–809, 2015.

[40] 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 *Proceedings of the 15th international conference on extending database technology*, pages 480–491, 2012.