The Power of Core Clique Removal for Exact Clique Enumeration

XIAOWEI YE, Beijing Institute of Technology, China
RONG-HUA LI, Beijing Institute of Technology, China
GUOREN WANG, Beijing Institute of Technology, China

Clique enumeration, including maximal clique enumeration and k-clique enumeration, is a fundamental problem in graph analysis. However, existing clique enumeration algorithms often struggle to efficiently handle complex real-world graphs. To address this issue, we propose a novel Core Clique Removal (CCR) technique that removes a large core clique from the original graph. After removing the core clique, the remaining graph is expected to become sparser. We show that the clique enumeration problem on the original graph can be reduced to the problem of enumerating cliques in the remaining sparser graph, thus reducing computational complexity. Building upon the CCR technique, we propose two innovative and efficient algorithms: CCRMCE for maximal clique enumeration and CCRKCE for k-clique enumeration. Both algorithms are non-trivial and offer substantial improvements over previous approaches. Our extensive experiments on 20 real-world graphs highlight the efficiency of our methods. CCRMCE demonstrates multiple times faster performance, while CCRKCE achieves an order of magnitude speedup compared to state-of-the-art algorithms.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**.

Additional Key Words and Phrases: Clique enumeration, Core Clique Removal, Graph algorithms

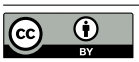
ACM Reference Format:
Xiaowei Ye, Rong-Hua Li, and Guoren Wang. 2025. The Power of Core Clique Removal for Exact Clique Enumeration. *Proc. ACM Manag. Data* 3, 4 (SIGMOD), Article 269 (September 2025), 27 pages. https://doi.org/10.1145/3749187

1 Introduction

Cliques, also known as complete subgraphs, are fundamental substructures in graph data management and analysis. A maximal clique is a clique that cannot be extended by adding any adjacent vertices, while a k-clique is a complete subgraph of size k. Clique enumeration problems are typically classified into two main types: the enumeration of maximal cliques and the enumeration of k-cliques for small values of k. Both types of enumeration have various applications in areas such as network analysis [4, 33, 52], web graph mining [23], bioinformatics [25], finance market analysis [20] and computational biology [1, 34].

Although clique enumeration is crucial for many applications, existing methods for enumerating cliques are still not very efficient when dealing with complex real-world graphs. For maximal clique enumeration, current state-of-the-art methods mainly rely on the pivot-based Bron-Kerbosch algorithm (BK) [5] and its optimizations [19, 21, 32, 39]. One key technique in the most efficient BK algorithms is degeneracy ordering, which aims to minimize the complexity of the induced subgraphs of out-neighbors of each vertex. Similarly, state-of-the-art k-clique enumeration algorithms also

Authors' Contact Information: Xiaowei Ye, yexiaowei@bit.edu.cn, Beijing Institute of Technology, Beijing, China; Rong-Hua Li, lironghuabit@126.com, Beijing Institute of Technology, Beijing, China; Guoren Wang, wanggrbit@gmail.com, Beijing Institute of Technology, Beijing, China.



ACM 2836-6573/2025/9-ART269
https://doi.org/10.1145/3749187

widely use degeneracy ordering or its variants [17, 26, 31, 49] to speed up computation. However, even with degeneracy ordering, there still exist induced subgraphs of the out-neighbors of vertices that are highly complex in real-world graphs, leading to inefficiencies in the algorithms. These induced subgraphs often contain a large number of cliques, and the cliques within them are heavily overlapping, making them computationally challenging to handle.

To overcome this challenge, we propose a novel technique called **Core Clique Removal (CCR)** to reduce the complexity of the induced subgraphs in degeneracy ordering, thereby improving the efficiency of clique enumeration algorithms. The **core clique** is a large clique that can be computed in linear time by iteratively removing the smallest degree vertex from the graph until only a clique remains. For a complex and dense graph $G(V, E)$, there often exists a large core clique C that acts as the central structure of the graph. Core Clique Removal operates by partitioning the set of vertices V into two disjoint sets: C and $V \setminus C$. Any clique K can then be classified into one of three types: (1) Clique entirely within C, i.e., $K \subseteq C$. For these cliques, no additional computation is required, and we can directly output the desired cliques in C using combinatorial methods. (2) Clique entirely within $V \setminus C$, i.e., $K \subseteq V \setminus C$. These cliques are easier to compute because, once the large core clique is extracted, the remaining subgraph is expected to be sparse. Enumerating cliques in a sparse graph is often significantly more efficient [21, 26, 31]. (3) Clique spanning both C and $V \setminus C$, i.e., $K \cap C \neq \emptyset$ and $K \cap (V \setminus C) \neq \emptyset$. For these cliques, the vertices in $K \cap C$ serve as common neighbors of the vertices in $K \cap (V \setminus C)$. This means that we only need to compute cliques of the second type, i.e., cliques contained within $V \setminus C$. In this way, the Core Clique Removal technique transforms the original problem of computing cliques in V into the problem of computing cliques in $V \setminus C$. By applying CCR to the out-neighbor induced subgraphs in degeneracy ordering, we can significantly reduce the computational complexity of the clique enumeration process. Moreover, our CCR method can be easily parallelized, as the clique enumeration computations within each out-neighbor induced subgraph are independent and can be processed in parallel.

Based on our CCR techinque, we develop two novel algorithms for the maximal clique enumeration and the k-clique enumeration problems respectively. Specifically, for the maximal clique enumeration problem, we propose a new BK-style algorithm called CCRMCE, which innovatively integrates the CCR technique to accelerate computation. CCRMCE incorporates a unique pivot selection technique that considers the core clique, significantly reducing the number of recursive calls and set intersection operations that are common in existing BK algorithms. As a result, CCRMCE outperforms previous algorithms in terms of speed, running notably faster than state-of-the-art algorithms on all 20 real-world graph datasets tested. It is important to note that the design of CCRMCE is both complex and non-trivial, as it must account for various factors. The original BK algorithm uses three parameters (R, P, X) which represent the already enumerated cliques, the candidate set, and the forbidden set of vertices, respectively. In contrast, CCRMCE introduces a new parameter C to represent the core clique. Consequently, CCRMCE must carefully manage the complex relationships between R, P, X, and C, requiring a detailed and thoughtful design of the algorithm.

For the k-clique enumeration problem, we propose another new algorithm called CCRKCE. This algorithm leverages the CCR technique to efficiently enumerate all k cliques in a graph. By removing the core clique and focusing on the remaining sparse subgraph, CCRKCE reduces the enumeration space and computational complexity, leading to significant performance improvements compared to existing methods. We show that the time complexity of CCRKCE is based on the arboricity [11] of the remaining graph, which is often much smaller than that of the original graph. Arboricity is a graph parameter used to measure the sparsity of a graph; it is defined as the minimum number of edge-disjoint forests into which the graph can be partitioned [11, 54]. The development of CCRKCE also involves careful consideration of the interactions between the core clique and the rest of the

graph. It employs an optimized strategy to handle the enumeration of k-cliques that span both the core clique and the remaining vertices, ensuring that no cliques are missed while maintaining efficiency.

In summary, our contributions are as follows:

A Novel Core Clique Removal Technique. We develop a novel Core Clique Removal (CCR) technique, which removes a large core clique from the graph and reduces the complexity of the remaining subgraph, thereby enhancing the efficiency of clique enumeration algorithms. Furthermore, our CCR technique is inherently parallelizable since the clique enumeration tasks across different out-neighbors are mutually independent.

Novel Maximal Clique Enumeration Algorithm. We propose a new algorithm for maximal clique enumeration, called CCRMCE. This algorithm integrates the CCR technique with pivot-based Bron-Kerbosch methods to develop a novel pivot strategy, achieving faster and more efficient performance.

Novel k-clique Enumeration Algorithm. We present a novel CCRKCE algorithm for k-clique enumeration, which leverages the CCR technique to improve performance by only enumerating sub-cliques on the remaining subgraph with the core clique removed. The time complexity of CCRKCE is dependent on the arboricity of the remaining subgraph after core clique removal, which is significantly lower than that of previous algorithms.

Extensive Experiments. We conduct extensive experiments on 20 real-world datasets, demonstrating that our algorithms achieve substantial speedups compared to state-of-the-art (SOTA) methods. For instance, on the Stanford dataset (with 281,903 vertices and 1,992,636 edges), our CCRMCE algorithm for maximal clique enumeration completes in just 0.67 seconds, whereas the SOTA methods [19, 32] require 2.01 and 3.44 seconds, respectively. This leads to our CCRMCE algorithm being approximately three times faster than the SOTA approaches. In the same dataset, for the k-clique enumeration problem with $k = 7$, our CCRKCE algorithm takes 0.32 seconds, while the SOTA method [26] consumes 3.34 seconds, demonstrating an order of magnitude speedup. On a particularly challenging dataset ComLiveJ (4,036,538 vertices and 34,681,189 edges), our CCRKCE algorithm enumerates all 7-cliques in only 4726 seconds, whereas all existing algorithms fail to produce results within 24 hours. To the best of our knowledge, this is the first time that all 7-cliques can be enumerated on ComLiveJ using a sequential algorithm.

Reproducibility. The source code of this paper is released at https://github.com/LightWant/coreCliqueRemoval.

2 Preliminaries

Consider an undirected graph $G = (V, E)$, where V denotes the set of vertices and $E \subseteq V \times V$ denotes the set of edges. Let n and m be the number of vertices and edges of G respectively. Denote by $N(u, S)$ the set of neighbors of u in the vertex set $S \subseteq V$, i.e. $N(u, S) = \{v | v \in S, \exists (u, v) \in E\}$. For a subset S of V, we denote by $G(S) = (S, E(S))$ the subgraph of G induced by S, where $E(S) = \{(u, v) \in E | u, v \in S\}$. We use $N(u)$ and S to replace $N(u, V)$ and $G(S)$ respectively if the context is clear.

Let $\vec{V} = \langle v_1, v_2, ..., v_n \rangle$ be a totally ordered set of V by a one-to-one map $\pi : V \rightarrow \vec{V}$ of a vertex in V to a vertex in $\vec{V}$. Based on such an order, we can obtain a directed acyclic graph (DAG) $\vec{G} = (\vec{V}, \vec{E})$ by orienting the edges of the undirected graph G. Specifically, for each undirected edge (u, v) in G, we obtain a directed edge $\langle u, v \rangle$ in $\vec{G}$ if $\pi(u) < \pi(v)$, otherwise we get a directed edge $\langle v, u \rangle$. Let the set of ordered edges be $\vec{E}$ and the neighbors of u in the DAG $\vec{G}$ be $N\langle u, \vec{G} \rangle = \{v | \langle u, v \rangle \in \vec{E}\}$.

(a) A graph

1	3 4 7 8 10
2	3 5 6 10
3	1 2 4 6 10
4	1 3 7 8 9 10
5	2 6 7 8 9
6	2 3 5 7 8 9 10
7	1 4 5 6 8 9 10
8	1 4 5 6 7 9 10
9	4 5 6 7 8 10
10	1 2 3 4 6 7 8 9

(b) The adjacency list

Fig. 1. An example graph and its adjacency list (although the graph is somewhat complex as an example, it can show the superiority and the key idea of our algorithms.)

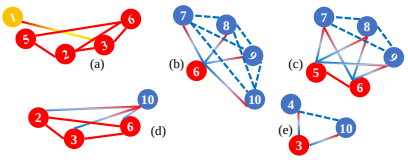


Fig. 2. Removing the core clique

We call $N\langle u, \vec{G}\rangle$ the set of out-neighbors of u. Given a set of ordered vertices $\vec{P}$ and a vertex u, let $N\langle u, \vec{P}\rangle$ be an ordered subset $\langle v | \langle u, v\rangle \in \vec{E}, v \in \vec{P}\rangle$.

A widely-used ordering heuristics for clique enumeration is the *degeneracy ordering* [31, 35]. The degeneracy ordering of vertices in G is defined as an ordering $\langle v_1, ..., v_n\rangle$ such that the degree of v_i is minimum in the subgraph induced by $\{v_i, ..., v_n\}$ for each v_i in G. The *degeneracy* of G, denoted by $\delta(G)$, is a measure of the sparsity of G [7]. A smaller $\delta(G)$ indicates a more sparse graph. The degeneracy of G is defined as $\delta(G) = \max_{v_i} |N(v_i, \{v_{i+1}, ..., v_n\})|$. It is evident that for any order-based DAG $\vec{G}$, it holds that $|\max_u N\langle u, \vec{G}\rangle| \geq \delta(G)$. Thus, the maximum size of the out-neighbors of each vertex is minimized in the degeneracy ordering [3]. It is well known that the degeneracy ordering of a graph can be computed in $O(m+n)$ time by iteratively removing the vertex with the smallest degree [31, 35]. The following example illustrates the concept of degeneracy ordering.

Example 2.1. The degeneracy ordering of the graph in Figure 1 is $\langle 2, 5, 3, 1, 6, 4, 7, 8, 9, 10\rangle$, where the degree of v_i is minimum in the subgraph induced by $\{v_i, ..., v_n\}$ for each v_i. For example, letting the undirected graph $G_3 = G(\{3, 1, 6, 4, 7, 8, 9, 10\})$, the vertex 3 has four neighbors $\{1, 4, 6, 10\}$ in G_3, while vertex 6 has five neighbors $\{3, 7, 8, 9, 10\}$ in the undirected graph G_3, vertex 9 has five neighbors $\{4, 6, 7, 8, 10\}$, and the other vertices have more neighbors. Therefore, the vertex 3 has the minimum number of neighbors and should be removed first from G_3.

The arboricity of a graph $G = (V, E)$, denoted by $\alpha(G)$, is defined as the minimum number of edge-disjoint forests that are needed to cover all the edges of the graph [11]. The arboricity $\alpha(G)$ is widely used for analyzing the worst-case time complexity of clique enumeration algorithms [11, 54]. Below, we give the concepts of the clique and maixmal clique.

Definition 2.2 (clique). A subgraph $G(S) = (S, E(S))$ is a clique if every pair of vertices is connected by an edge.

By Definition 2.2, we refer to a clique that contains k vertices as a k-clique.

Definition 2.3 (maximal clique). A subgraph $G(S) = (S, E(S))$ is a maximal clique if every pair of vertices is connected by an edge and $\forall u \in V \setminus S, G(S \cup \{u\})$ is not a clique.

With Definitions 2.2 and 2.3, we formulate our problems below.

The Maximal Clique Enumeration Problem (MCEP). Given a graph G, the maximal clique enumeration problem is to enumerate all maximal cliques in G.

The k-Clique Enumeration Problem (KCEP) Given a graph G and an integer k, the k-clique enumeration problem is to enumerate all k-cliques in G.

Note that both MCEP and KCEP are highly challenging problems, as no known polynomial-time exact algorithms exist to solve them [5, 17, 19, 21, 26, 31]. Existing practical algorithms for MCEP

Algorithm 1: Compute the Core Clique

Input: An undirected graph $G = (V, E)$
Output: The core clique of G
1 **while** *True* **do**
2 $u \leftarrow \arg\min_{v \in V} |N(v, V)|$;
3 **if** $|N(u, V)| = |V| - 1$ **then**
4 **return** V;
5 $V \leftarrow V \setminus \{u\}$;

and KCEP are often inefficient when applied to complex real-world graphs as discussed in Section 1. This motivates us to develop more efficient techniques capable of effectively handling such complex real-world graphs. In the following section, we develop a novel and general technique to address this challenge.

3 The Core Clique Removal Technique

In this section, we propose a novel **Core Clique Removal (CCR)** technique for clique enumeration problems. Below, we first give the definition of core clique.

Definition 3.1 (Core Clique). Given a graph $G = (U, V)$, the core clique of G is a clique which is obtained by iteratively removing the vertices with the smallest degree until the remaining subgraph forms a clique.

According to Definition 3.1, we can compute the core clique using Algorithm 1. When $|N(u, V)| = |V| - 1$ (line 3 of Algorithm 1), i.e., when every pair of vertices is connected, $G(V)$ is the core clique. Clearly, the time complexity of Algorithm 1 is $O(n + m)$.

To better understand the concept of the core clique, consider the following example, which demonstrates how the vertices are deleted iteratively to identify the core clique in a graph.

Example 3.2. As shown in Example 2.1, the removal order of the vertices in the graph depicted in Figure 1 is $\{2, 5, 3, 1, 6, 4, 7, 8, 9, 10\}$. The resulting final subgraph $\{4, 7, 8, 9, 10\}$, highlighted by the blue vertices in Figure 1, is the core clique.

3.1 The CCR-based Enumeration Framework

The key idea. Given a graph $G(V, E)$, let C be the set of vertices that form the core clique in G. The key idea of the CCR technique is to remove C from V, and then the cliques in the graph can be divided into three types:

- **Type (I): Cliques entirely within** C. Since C is a large clique, these cliques are subgraphs of C and can be trivially enumerated using a combinatorial method.
- **Type (II): Cliques entirely within** $V \setminus C$. After removing the core clique, the remaining subgraph $G(V \setminus C)$ is expected to be much sparser, making it easier to enumerate these cliques.

  Example 3.3. The subgraph in Figure 2(a) represents the remaining graph after removing the core clique from Figure 1. Compared to the whole graph, the remaining subgraph is much sparser, with $\{2, 3, 6\}$ and $\{2, 5, 6\}$ being the largest cliques.

- **Type (III): Cliques spanning both** C **and** $V \setminus C$. To find these cliques, we need to enumerate the partial-clique R in $V \setminus C$, and identify the common neighbors of R in C. The common neighbors, i.e., $\cap_{u \in R} N(u, C)$, naturally form a clique.

Algorithm 2: The CCR-based Enumeration Framework

Input: An undirected graph $G = (V, E)$
Output: All maximal cliques or k-cliques
1 Computing the degeneracy ordering;
2 $\vec{G} \leftarrow$ Orientating G by the degeneracy ordering;
3 **for** $u \in V$ **do**
4 $C \leftarrow$ the core clique of $G(N\langle u, \vec{G}\rangle)$;
5 $P \leftarrow N\langle u, \vec{G}\rangle \setminus C$;
6 **for** *each clique* K' *in* $G(P)$ **do** /*Enumerate each clique, and K' may be empty.*/
7 $C' \leftarrow \cap_{u \in K'} N(u, C)$; /*When K' is empty, C' is C */;
8 For MCEP: **if** $(K' \cup C' \cup \{u\}$ is maximal) **then** Report $K' \cup C' \cup \{u\}$ as a maximal clique;
9 For KCEP: Report $K' \cup K'' \cup \{u\}$ as a k-clique, $\forall K'' \subseteq C', |K' \cup K'' \cup \{u\}| = k$;

Example 3.4. The four maximal cliques in Figure 2(b), (c), (d), and (e) belong to this category. To find them, we only need to enumerate the red vertices in $V \setminus C$. For instance, in Figure 2(c), we enumerate $\{5, 6\}$ and their common neighbors in C form the sub-clique $\{7, 8, 9\}$.

Note that both Type (II) and Type (III) cliques only require enumerating the cliques in the remaining sparser subgraph $G(V \setminus C)$, making the enumeration process more efficient.

For large real-world graphs, instead of computing the core clique for the entire graph, we focus on the subgraphs of the out-neighbors of each vertex in the degeneracy ordering. For each subgraph, we remove the core clique in the subgraph and then enumerate the above three types cliques. The core clique of the entire graph is typically small compared to the graph size, as real-world graphs generally have low degeneracy, and the degeneracy is the upper bound of core clique size. Consequently, removing only one core clique from the entire graph does not significantly reduce its density. In contrast, the subgraphs induced by the out-neighbors in degeneracy ordering are often much denser, with relatively larger core clique. Removing the core cliques from these subgraphs often leads to substantially sparser remaining subgraphs.

The details of the CCR-based enumeration framework are outlined in Algorithm 2.

Let $\vec{G} = (V, \vec{E})$ be the DAG generated by the degeneracy ordering (line 2), where C represents the core clique of the subgraph $N\langle u, \vec{G}\rangle$, with $u \in V$ (line 4), and P denotes the remaining vertices after removing the core clique (line 5). The framework then enumerates the three types of desired cliques by only enumerating the cliques in P (line 6). For MCEP, we check the maximality of $K' \cup C' \cup \{u\}$ and report it. When K' is empty, C', the common neighbors, is the core clique C, where the framework enumerates the Type (I) maximal cliques. When C' is empty, i.e., when K' has no common neighbors in C, it enumerates the Type (II) maximal cliques, otherwise Type (III) maximal cliques (line 8). For KCEP, when K' is empty, it enumerates the Type (I) k-cliques. When K'' is empty, it enumerates Type (II) k-cliques. When both K' and K'' are not empty, it enumerates Type (III) k-cliques (line 9). As shown in the framework, no matter which type, it only needs to enumerate the cliques in P, the remaining subgraph.

Correctness of the framework. Each maximal clique or k-clique is enumerated exactly once in our CCR-based framework, and no clique will be missed. For a clique K, let u be the vertex with the smallest order in K. The set $K \setminus \{u\}$ is all the out-neighbors of u. Thus, K will be enumerated only among the out-neighbors of a single vertex (line 3 of Algorithm 2). Within the out-neighbors of u, $K \setminus \{u\}$ belongs to one of the three types discussed earlier. Type (I) cliques are enumerated exactly once when K' is empty. Type (II) cliques are enumerated only once, specifically in line 6. For Type (III) cliques, their part in P is enumerated only once in line 6, while the remaining part

is found among the common neighbors (line 7). Therefore, each Type (III) clique is enumerated exactly once.

Challenge in implementation. The CCR-based framework offers a high-level solutions for clique enumeration problems. However, it is neither straightforward nor trivial to design specific algorithms based on this framework. In practice, directly enumerating all subsets of P in line 6 is inefficient. We must carefully design search strategies and pruning techniques to avoid redundant computations. For example, in the MCEP, there are some cliques K' for which $K' \cup C'$ is not maximal—how can we avoid such cases? Similarly, for KCEP, how to enumerate the cliques efficiently while consider both the remaining subgraph and the core clique? To address these challenges, we propose a novel BK-style algorithm for MCEP and a new enumeration-based algorithm for KCEP.

Note that the most time-consuming step of Algorithm 2 is line 6, which often exhibits exponential time complexity due to the potential existence of an exponential number of desired cliques. Clearly, this complexity is much higher than the total time complexity for computing all core cliques for all vertices (lines 3-4), as indicated in the following theorem. Due to the space limits, all the missing proofs can be found in the Appendix.

Theorem 3.5. *Given a graph $G = (V, E)$, the time complexity of computing all core cliques for all subgraphs induced by out-neighbors in Algorithm 2 is $O(\alpha(G)|E|)$, where $\alpha(G)$ is the arboricity of the G.*

Proof. Clearly, for each out-neighbors induced subgraph, the time complexity is linear with respect to the size of the out-neighbors. Thus, the total time complexity is

$$\sum_{u \in V} |E(N\langle u \rangle)| = \sum_{u \in V} \frac{1}{2} \sum_{v \in N\langle u \rangle} |N(v, N\langle u \rangle)| \leq \frac{1}{2} \alpha(G)|E|. \tag{1}$$

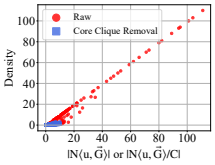
□

Remark. It is worth noting that, in addition to the core clique, our framework is also applicable to any other heuristically large clique [6]. For any large clique, all the desired cliques can be classified into three types similar to our core clique technique. Our framework can clearly handle these three types in the same way. In this work, we specifically select the core clique for our approach due to the following reasons: (1) the core clique is typically very large and often serves as a lower bound in maximum clique computation [6, 43]; (2) it can be computed in linear time; and (3) our experiments further demonstrate the high efficiency and effectiveness of core clique removal-based methods.

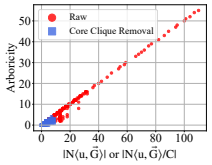
In the worst case, our CCR technique does not have a theoretical guarantee to make the remaining graph have a smaller arboricity. The reasons are as follows. First, the core clique may not belong to the densest subgraph, meaning its removal does not guarantee a reduction in graph arboricity, since arboricity depends on the densest subgraph [38]. Second, as the core clique is a greedy maximal clique rather than the maximum clique, larger cliques may remain after its removal. Since arboricity upper bounds depend on maximum clique size [11], this further explains why we cannot theoretically guarantee arboricity reduction. However, despite this theoretical limitation in the worst case, our empirical results (Figures 3 and 4) in the following section demonstrate that CCR effectively reduces arboricity in practice for real-world graphs.

3.2 Effectiveness Analysis of our Framework

To demonstrate the effectiveness of the CCR-based framework, we show that removing the core clique from a subgraph effectively reduces both its density and arboricity on complex real-world networks. Here, both the density (the number of edges divided by the number of nodes) and arboricity are measure of the cohesiveness and structure complexity of graphs.



(a) Density



(b) Arboricity

Fig. 3. The size and the density/arboricity of the subgraphs without/with CCR on the DBLP network

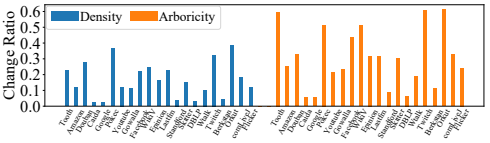


Fig. 4. The average change ratio of the density and arboricity by CCR (the smaller, the better)

For a node u in the degeneracy-ordered graph $G(\vec{V}, \vec{E})$, $G(N\langle u, \vec{G}\rangle)$ is the induced subgraph of the out-neighbors of u, and $G(N\langle u, \vec{G}\rangle \setminus C_u)$ is the remaining subgraph with the core clique removed, where C_u is the core clique in $G(N\langle u, \vec{G}\rangle)$.

In Figure 3(a), each red circle labeled "Raw" represents a vertex $u \in V$ with $x_u = |N\langle u, \vec{G}\rangle|$ and $y_u^d = \frac{|E(N\langle u, \vec{G}\rangle)|}{|N\langle u, \vec{G}\rangle|}$, while each blue square labeled "Core Clique Removal" represents a vertex u with $x_u = |N\langle u, \vec{G}\rangle \setminus C_u|$ and $y_u^{d,ccr} = \frac{|E(N\langle u, \vec{G}\rangle \setminus C_u)|}{|N\langle u, \vec{G}\rangle \setminus C_u|}$. In Figure 3(b), each red circle labeled "Raw" represents a vertex u with $x_u = |N\langle u, \vec{G}\rangle|$ and $y_u^{\alpha} = \alpha(G(N\langle u, \vec{G}\rangle))$, while each blue square labeled "Core Clique Removal" represents a vertex u with $x_u = |N\langle u, \vec{G}\rangle \setminus C_u|$ and $y_u^{\alpha,ccr} = \alpha(G(N\langle u, \vec{G}\rangle \setminus C_u)))$. In the y-axis of Figures 3(a) and 3(b), both the density and arboricity decrease after the core clique removed, suggesting that the remaining subgraphs become significantly sparser. In the x-axis, after removing the core cliques, the remaining subgraphs have significantly smaller size.

Figure 4 illustrates the average change ratio of the density and arboricity across 20 datasets. The average change ratios for density and arboricity in a graph $G(V, E)$ are computed as follows:

$$avg_{\text{density}} = \frac{1}{n} \sum_{u \in V} \frac{y_u^{d,\text{ccr}}}{y_u^d} \quad \text{and} \quad avg_{\alpha} = \frac{1}{n} \sum_{u \in V} \frac{y_u^{\alpha,\text{ccr}}}{y_u^{\alpha}},$$

respectively. Here, $y_u^{d,\text{ccr}}$ is no larger than y_u^d and $y_u^{\alpha,\text{ccr}}$ is no larger than y_u^{α}, ensuring that both avg_{density} and avg_{α} fall within the range $[0, 1]$. A smaller value of avg_{density} and avg_{α} indicates better performance of the CCR technique. The results reveal that both the density and arboricity decrease significantly across all datasets, demonstrating the effectiveness of our approach.

Based on such a powerful CCR-based framework, we can design efficient algorithms for both MCEP and KCEP. The CCR technique enables us to focus on sparser subgraphs, which leads to faster enumeration of cliques.

3.3 Parallelization of Framework

Our CCR-based framework is inherently well-suited for parallel implementation. In parallel computing, achieving finer granularity—where each thread handles smaller subtasks—often leads to better load balancing and enhanced parallelism. In the clique enumeration problem, each subtask can naturally be defined as "enumerating cliques containing a specific vertex" (line 3 of Algorithm 2). Our CCR technique can simplify these subtasks by enumerating on the sparser remaining graphs, facilitating more straightforward parallelization.

When parallelizing code, it is advisable to decompose a complex loop into multiple simpler loops whenever possible. This approach helps reduce dependencies, improve parallel efficiency, and minimize resource contention. For a more efficient implementation, we split the original "for"

loop (line 3 of Algorithm 2) into two distinct phases: (1) Core clique computation: For each vertex, we compute its core cliques (line 4 of Algorithm 2) and store them in main memory, which requires at most $O(m)$ space; (2) Parallel clique enumeration: Using the precomputed core cliques, we then compute the cliques for each vertex in parallel.

We employ a vertex-weight-based work allocation scheme combined with a work-stealing strategy. Based on the degeneracy ordering $\langle v_1, v_2, \ldots, v_n \rangle$, each vertex v_i is assigned a weight of $|N\langle v_i \rangle|$. Each thread t is allocated a contiguous integer interval $[i, j]$, meaning that thread t processes vertices $v_i, v_{i+1}, \ldots, v_j$. The intervals are designed such that: (1) The sum of weights within each interval is equal to ensure load balancing; (2) All intervals collectively cover every vertex in V. The contiguous intervals improves computational locality for each thread, while the equal total weight per interval guarantees balanced workloads. If a thread finishes its assigned work early, it sequentially steals tasks from other unfinished threads to further maintain load balancing. Experimental results in Section 6 for both MCEP and KCEP demonstrate the high effectiveness of parallelization within our CCR-based framework.

4 Maximal Clique Enumeration by CCR

In this section, we propose a novel BK-style algorithm for maximal clique enumeration based on our CCR technique. Below, we first briefly introduce existing BK-style algorithms and discuss their limitations.

4.1 Existing BK Algorithms and Their Defects

Existing state-of-the-art maximal clique enumeration algorithms [19, 21, 32, 39] are all based on the classic Bron-Kerbosch (BK) framework [5, 47] with the degeneracy ordering trick [21]. The pseudo-code of this framework is shown in Algorithm 3. It employs a recursive procedure that iterates over three sets: the set of candidates P, the current partial clique R, and the forbidden vertices X (line 4 of Algorithm 3). All vertices of the $P \cup X$ are all neighbors of every vertex in R. The goal of $\mathsf{BK}(P, R, X)$ is to enumerate all the maximal cliques contains all vertices in R, partial vertices in P, and no vertices in X. If $P \cup X$ becomes empty, the algorithm reports R as a maximal clique (lines 5-6). In each recursive call, a pivot vertex u_p is chosen from $P \cup X$ to reduce the number of recursive calls (line 7). Since all maximal cliques must contain a non-neighbor of u_p (considering u_p itself) [47], the recursive calls can be restricted to $P \setminus N(u_p, P)$ (line 8). In each recursive call, a vertex u from P is added to the partial clique R, and P and X include only neighbors of u (lines 9-10). The worst-case time complexity of $\mathsf{BK}(P, R, X)$ is $O(3^{(|P|+|X|)/3})$ [47]. By utilizing the degeneracy ordering, BK is called for each vertex $u \in V$, with P being the out-neighbors of u, $R = \{u\}$ and X being the in-neighbors of u (line 3). Algorithm 3 has a complexity of $O(d_{max} n 3^{d_{max}/3})$, where $d_{max} = \max_{u \in V} |N(u)|$ [21]. There are several studies that aim to improve the efficiency of Algorithm 3, but they still follow the general framework of Algorithm 3 [19, 32, 39].

Despite its efficiency in handling real-world sparse graphs, Algorithm 3 still has some notable drawbacks. First, this framework may suffer from large search branches, leading to an excessive number of recursive calls. As the algorithm explores all possible combinations of vertices to form maximal cliques, the enumeration space can grow exponentially, making it computationally expensive for large dense graphs or graphs with many maximal cliques. Second, set intersections, which are crucial for updating the P and X during the recursive process (line 9 of Algorithm 3), can become a significant bottleneck. In fact, a previous study [24] found that set intersection operations account for 73.6% of the total running time. These factors can limit the scalability of the BK algorithm for large-scale real-world applications.

Algorithm 3: Pivot-based Bron-Kerbosch Framework

```
Input: An undirected graph G = (V, E)
Output: All maximal cliques in G
1  G⃗ ← an ordered G by the degeneracy ordering;
2  for u ∈ V do
3      P ← N⟨u, G⃗⟩; BK (P, {u}, N(u) \ P);
4  Procedure BK (P, R, X)
5      if P ∪ X = ∅ then
6          report R;
7      u_p ← arg max_{u∈P∪X} |N(u, P)|;
8      for u ∈ P \ N(u_p, P) do
9          BK (N(u, P), R ∪ {u}, N(u, X));
10         P ← P \ {u}; X ← X ∪ {u};
```

4.2 A Sub-problem: CCR-based MCEP

The original BK algorithm, as outlined in Algorithm 3, uses three parameters: P, R, and X. To accommodate the introduction of the core clique, we introduce an additional parameter C, and carefully examine the relationships among P, R, X, and C. This gives rise to the following sub-problem:

The CCR-based MCEP sub-problem: *Given four sets of vertices C, P, R, and X, where $C \cup P \cup X$ are all neighbors of every vertex in R, and both R and C are cliques, we seek to enumerate all maximal cliques containing R as a sub-clique, excluding vertices in X, while potentially including some vertices from $P \cup C$.*

In this sub-problem, the overlaps between maximal cliques become more complex than the problem solved by the BK algorithm, presenting a non-trivial challenge for algorithm design. The primary challenges here include selecting pivot vertices considering all the vertices in C, P, X and ensuring that each maximal clique is enumerated exactly once. These challenges are addressed by analyzing the properties of the sub-problem, which are formalized in the following lemmas and theorems. Based on these lemmas and theorems, we can design our novel CCRMCE algorithm for MCEP.

Properties of the Sub-Problem. Firstly, Lemma 4.1 establishes that $C \cup R$ is a clique.

Lemma 4.1. *$C \cup R$ is a clique.*

Proof. Since both C and R are cliques, and C consists of common neighbors for all vertices in R, the union $C \cup R$ must also form a clique. □

The following Lemma 4.2 provides a condition under which $C \cup R$ is a maximal clique. A maximal clique is considered *legal* if it does not contain any vertices from X.

Lemma 4.2. *If for all $u \in P \cup X$, $|N(u, C)| < |C|$, then $C \cup R$ is a legal maximal clique.*
Proof Sketch: If $|N(u, C)| < |C|$, $|N(u, C \cup R)| < |C \cup R|$.

We provide Example 4.3 to illustrate Lemma 4.1 and Lemma 4.2.

Example 4.3. In Figure 5(a), X is empty and no vertex in P is connected to all vertices in C, so the subgraph $R \cup C$ forms a maximal clique. In Figure 5(b), the vertex 1 in X is connected to all vertices in C, so the subgraph $R \cup C = \{3, 4, 7, 10\}$ is not a maximal clique, as vertex 1 can be added to form a larger clique.

Lemma 4.4. *Let u_p be a vertex in $C \cup P \cup X$. Any maximal clique in $G(C \cup P \cup X)$ must contain at least one non-neighbor of u_p (note that u_p is a non-neighbor of itself).*

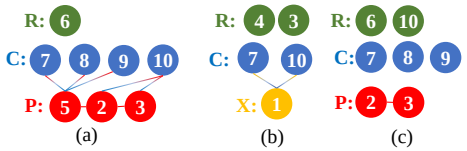


Fig. 5. Three subgraphs of Figure 1. The edges connected to R and the edges between C are omitted.

Proof Sketch: Otherwise by adding u_p to this clique, we would create a larger clique.

According to Lemma 4.4, we only need to consider the non-neighbors of an vertex u_p when extending maximal cliques. This is the way of utilizing pivot vertex in the classic BK algorithms (Algorithm 3). However, in our CCR-based algorithms, we can consider more cases to prune unnecessary search branches as stated in Lemma 4.5 and Theorem 4.6.

Lemma 4.5. *Any legal maximal clique in $G(C \cup P \cup X)$ must contain at least one vertex from P (excluding C if C is a legal maximal clique).*

Proof Sketch: Otherwise, it would either be a sub-clique of C or violate maximality by excluding all vertices in P.

Based on Lemmas 4.4 and 4.5, we can prove the following important theorem.

Theorem 4.6. *Let u_p be a vertex in $C \cup P \cup X$. Let $T_1 = \{u \mid u \in C \backslash N(u_p, C), N(u, P) \cap N(u_p, P) \neq \emptyset\}$ and $T_2 = P \backslash N(u_p, P)$. Any legal maximal clique (except C if C is a legal maximal clique) in $G(C \cup P \cup X)$ must contain at least one vertex from $T_1 \cup T_2$.*

Proof Sketch: In all cases ($u_p \in C$, $u_p \in P$, $u_p \in X$), $T_1 \cup T_2$ contains the necessary vertices to extend the maximal cliques .

Theorem 4.6 establishes a crucial condition for the algorithm used to recursively enumerate all legal maximal cliques in $G(C \cup P \cup X)$. In the classic BK algorithm, all the non-neighbors of u_p should be used to extend the maximal cliques (line 8 of Algorithm 3). Now we only need to extend the vertices in $T_1 \cup T_2$, which is a subset of the non-neighbors of u_p. Therefore, this theorem is important for reducing the enumeration space and guiding the algorithm's recursive exploration.

To illustrate Theorem 4.6, we provide an example that demonstrate how the sets T_1 and T_2 are derived.

Example 4.7. In Figure 5(a), let $u_p = 10$. The non-neighbor of 10 in C is 10 itself, and 10 has neighbors in P, thus $T_1 = \{10\}$, and $T_2 = \{5\}$.

In Figure 5(c), let $u_p = 7$. Since 7 has no neighbors in P, we have $T_1 = \emptyset$ and $T_2 = \{2, 3\}$. Clearly, the maximal clique $\{2, 3, 6, 10\}$ must contain vertices from T_2.

In Figure 5(c), let $u_p = 2$. The non-neighbors of 2 are $\{2\}$ and $\{7, 8, 9\}$. Since none of $\{7, 8, 9\}$ is connected to 2's neighbors in P, namely 3, we have $T_1 = \emptyset$ and $T_2 = \{2\}$. Thus, the maximal clique $\{2, 3, 6, 10\}$ must contain vertices from $T_1 \cup T_2 = T_2 = \{2\}$.

4.3 The Proposed CCRMCE Algorithm

Based on the CCR-based framework and the lemmas and theorem presented in the previous subsection, we propose a novel and efficient algorithm, CCRMCE, for MCEP.

The details of the CCRMCE algorithm are outlined in Algorithm 4. The algorithm follows the structure of the CCR-based framework in Algorithm 2, where it leverages degeneracy ordering and enumerates maximal cliques within the out-neighbors of each vertex, and for the subgraph induced by the out-neighbors, a core clique is first removed to improve efficiency. The CCRMC procedure in line 5 of Algorithm 4 is an implementation of lines 6-8 of Algorithm 2, by utilizing a novel pivoting technique in Theorem 4.6 to recursively generate candidate sets.

The recursive procedure CCRMC includes two stopping conditions (line 7). When the first condition meets, i.e., $fromP$ = true, there is a vertex from P has been added into R (line 12). The

Algorithm 4: CCRMCE

Input: A graph $G = (V, E)$
Output: All maximal cliques in G

1 $\overrightarrow{G} \leftarrow$ an ordered G by the degeneracy ordering;
2 **for** $u \in \vec{V}$ **do**
3 $\quad C \leftarrow$ the core clique of $G(N\langle u, \overrightarrow{G}\rangle)$;
4 $\quad P \leftarrow N\langle u, \overrightarrow{G}\rangle \setminus C$;
5 $\quad$ CCRMC $(C, P, \{u\}, N(u) \setminus N\langle u, \overrightarrow{G}\rangle$, true);
6 **Procedure** CCRMC $(C, P, R, X, fromP)$
7 $\quad$ **if** $fromP = true$ *and* $\forall u \in P \cup X, |N(u, C)| < |C|$ **then**
8 $\quad\quad$ Report $C \cup R$;
9 $\quad u_p \leftarrow \arg\max_{u \in P \cup C \cup X} |N(u, P \cup C)|$;
10 $\quad$ Compute $T = T_1 \cup T_2$ by Theorem 4.6.
11 $\quad$ **for** $u \in T$ **do**
12 $\quad\quad$ CCRMC $(N(u, C), N(u, P), R \cup \{u\}, N(u, X), 1[u \in P])$;
13 $\quad\quad P \leftarrow P \setminus \{u\}; C \leftarrow C \setminus \{u\}; X \leftarrow X \cup \{u\}$;

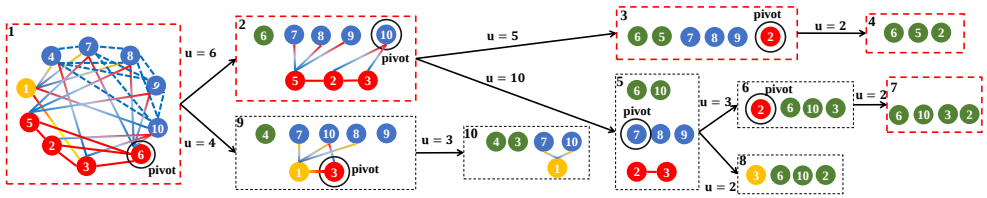


Fig. 6. The enumeration tree of the CCRMC procedure in CCRMCE for enumerating the maximal cliques of the graph in Figure 1, excluding vertex 1 (the yellow vertex in Figure 1). The enumeration tree consists of 10 nodes, labeled from 1 to 10. The vertices within the red-dotted boxes represent those for which $C \cup R$ is a reported legal maximal clique. In each tree node, the vertex in the black cycle is the pivot vertex u_p, the red vertices represent P, the blue vertices represent C, the green vertices represent R, and the yellow vertices represent X. Edges between the vertices in C (i.e., the blue vertices) and edges connected to R are omitted for clarity. A detailed description is provided in Example 4.9.

$fromP = true$ condition is based on Lemma 4.5 and ensures that if no vertex from P has been added to the clique, it will not be reported, thus avoiding repeated enumeration. Another explanation of the $fromP$ parameter is the observation of line 8 of Algorithm 2, where a maximal clique must contain nodes in P because $K' \subseteq P$. Lemma 4.8 shows the correctness of the parameter $fromP$.

Lemma 4.8. *For a node in the enumeration tree of the* CCRMC *procedure, let its parameters be denoted as C_1, R_1, and $fromP = false$, while the father node's parameters are denoted as C_2 and R_2. It holds that $C_1 \cup R_1 = C_2 \cup R_2$.*

Proof. When a node u is from C, it has $N(u, C) = C \setminus \{u\}$. Therefore, we have $R_1 = R_2 \cup \{u\}$ and $C_1 = C_2 \setminus \{u\}$, which implies that $C_1 \cup R_1 = C_2 \cup R_2$. □

This condition is intuitive because if only a vertex is moved from C to R, the overall clique $C \cup R$ remains unchanged. Consequently, for a legal maximal clique without repeated enumeration, there must exist at least one vertex from P.

The second condition in line 7, as stated in Lemma 4.1 and 4.2, ensures that the clique $C \cup R$ is maximal. If both conditions are satisfied, the current clique $R \cup C$ is reported.

To minimize unnecessary branches, the pivot vertex u_p is chosen (line 9) and the candidate set $T = T_1 \cup T_2$ is computed based on Theorem 4.6 (line 10).We choose the vertex that has the smallest number of non-neighbors as the pivot vertex (line 9). Note that the previous studies use the non-neighbors of the pivot vertex as the candidate set (denoted by T'), while our candidate set

T avoids the vertices in C that only connect to the non-neighbors of the pivot vertex in P, according to Theorem 4.6. Therefore, we have $T \subseteq T'$, a smaller candidate set. As shown in Example 4.9 and our experiments, our CCRMCE has smaller recursive enumeration tree than the previous BK algorithm in Algorithm 3.

During the recursive exploration, each vertex u in the candidate set T is added to the partial clique R, and the parameters C, P, and X are updated accordingly (line 12). Once processed, u is added to X (line 13) to prevent repeated enumeration of the same clique. When $u \in T_1$ (line 11), we can get $N(u, C) = C \setminus u$ directly, which reduce the computation of set intersection.

To illustrate the CCRMC procedure and how it enumerates maximal cliques efficiently, we provide the following example to describe the enumeration tree in Figure 6. The enumeration tree consists of 10 vertices, whereas the pivot-based BK algorithm produces a enumeration tree with 20 vertices for the same problem.

Example 4.9. Figure 6 shows the enumeration tree of the CCRMC *procedure, which takes as input* $C = \{4, 7, 8, 9, 10\}, P = \{2, 3, 5, 6\}$, *and* $X = \{1\}$, *representing the vertices of the example graph in Figure 1. The labeling details are provided in the caption of the figure. For example, in tree node-2, vertex* 10 *has the maximum number of neighbors in* $P \cup C$ *(line 9 of Algorithm 4). Thus,* 10 *is selected as the pivot vertex, and its non-neighbors are* $T = \{5, 10\}$. *Consequently, tree node-2 has two child vertices: node-3(*$u = 5$*) and node-5(*$u = 10$*).*

In tree node-1, 6 *is selected as the pivot vertex* u_p, *and the set of non-neighbors is* $T = \{6, 4\}$. *Therefore, tree node-1 has two child vertices: node-2, which adds vertex* 6 *into* R, *and node-9, which adds vertex* 4 *into* R. *node-9 would normally add* 6 *into* X *because* 6 *has been searched in tree node-2, but it is removed because* 6 *is not a neighbor of vertex* 4.

Both tree node-2 to tree node-5 have $C \cup R = \{6, 7, 8, 9, 10\}$, *which is a legal maximal clique. However, the* $fromP$ *parameter of node-5 is false, so only node-2 reports the legal maximal clique. Similarly, the* $fromP$ *parameter of node-9 is also false, and its legal maximal clique has already been reported by node-1.*

Node-6 does not report $C \cup R$ *because there is a vertex in* P *that connects to all vertices in* C *(here* C *is empty). Likewise, node-8 and node-10 do not report* $C \cup R$ *because there is a vertex in* X *that connects to all vertices in* C.

4.4 Analysis of Our CCRMCE Algorithm

Lemma 4.10 proves the correctness of the CCRMCE algorithm.

Lemma 4.10. *The* CCRMCE *algorithm correctly solves the MCEP.*

Proof Sketch: Theorem 4.6 guarantees that an arbitrary maximal clique M contains at least one vertex from T. M is reported exactly once by CCRMCE.

Theorem 4.11 provides the worst-case time complexity of CCRMCE.

Theorem 4.11. *The time complexity of* CCRMCE *is* $O(d_{max} n 3^{d_{max}/3})$, *where* d_{max} *is the maximum degree of the graph.*

Proof Sketch: Following the result established in [47], we can easily derive the above time complexity.

Note that the worst-case time complexity of our CCRMCE algorithm is identical to that of the previous BK-style algorithms [21, 47]. Since the number of maximal cliques in a graph $G = (V, E)$ can reach up to $O(3^{n/3})$ [47], both our algorithm and previous BK-style algorithms [21, 47] achieve the theoretically optimal worst-case time complexity. However, it is important to emphasize that, despite the same theoretical complexity, the practical performance of our algorithm is significantly better due to two key reasons: (1) It reduces the number of search branches and recursive calls by utilizing a novel pivot technique, as described in Theorem 4.6; and (2) It minimizes

Algorithm 5: kClist

```
Input: A graph G = (V, E), an integer k
Output: The k-cliques in G
1  G⃗ ← an ordered G by the degeneracy ordering;
2  for u ∈ V do
3      kClist (N⟨u, G⃗⟩, {u});
4  Procedure kClist (P⃗, R)
5      if |R| = k − 2 then
6          for e(u, v) ∈ E(P⃗) do Report R ∪ {u, v};
7          return;
8      for u ∈ P⃗ do
9          kClist (N⟨u, P⃗⟩, R ∪ {u});
```

the computation of set intersections by pruning unnecessary intersections involving the core clique. These improvements enable CCRMCE to efficiently enumerate maximal cliques with fewer operations. Experimental results show that CCRMCE achieves multiple times speedups compared to state-of-the-art algorithms, demonstrating its superior efficiency in real-world scenarios.

5 k-Clique Enumeration by CCR

In this section, we propose a novel algorithm for KCEP based on our CCR-based framework. Below, we start by reviewing the existing enumeration solutions for KCEP.

5.1 Existing Solutions for KCEP

The state-of-the-art algorithms for KCEP are also based on the ordering heuristic techniques [17, 26, 31, 49]. For degeneracy ordering, the problem of enumerating k-cliques changes into enumerating the $(k-1)$-cliques in $G(N\langle u, \vec{G}\rangle)$ for all $u \in V$. We introduce the kClist algorithm [17] in Algorithm 5, which is based on the degeneracy ordering. The algorithm begins by reordering the graph G (line 1) to form $\overrightarrow{G}$, ensuring vertices are processed in degeneracy order. For each vertex $u \in V$ (line 2), it invokes the recursive procedure kClist$(\vec{P}, R)$, where $\vec{P}$ is the set of out-neighbors of u, and $R = \{u\}$ represents the current partial clique under construction (line 3). The recursive procedure kClist$(\vec{P}, R)$ checks if the size of R has reached $k-2$ (line 4); if so, it reports all edges $(u, v) \in E(\vec{P})$ as extensions of R to form complete k-cliques (line 5). Otherwise, it iterates over each vertex $u \in \vec{P}$ in order (line 8), recursively exploring further extensions of R by calling kClist$(N\langle u, \vec{P}\rangle, R \cup \{u\})$ (line 9). This systematic exploration ensures that every k-clique is identified exactly once, making the algorithm efficient for k-clique enumeration in large and sparse graphs. The algorithm's worst time complexity is bounded as $O(km(\alpha(G) - \frac{1}{2})^{k-2})$, where $\alpha(G)$ is the arboricity of the input graph G (note that $\alpha(G) \leq \delta(G) < 2\alpha(G)$).

There are alternative ordering techniques for k-clique enumeration, such as color ordering [31], edge ordering, color edge-ordering [49]. While these approaches offer varying degrees of optimization, they often face limitations when applied to large real-world networks, where the complexity and scale of the data can hinder their efficiency and scalability.

The pivot-based method Pivoter [26] is originally designed for k-clique counting, but it can also be applied to the k-clique enumeration problem (simply changing the counting operator into reporting). Although Pivoter is efficient on most of the datasets for large value of k, it is not as efficient as the algorithms like kClist for small value of k. Most real-world applications need to enumerate k-clique with a small value of k because small size motifs are building blocks of the real-world networks [36]. Furthermore, the worst case time complexity of Pivoter is bounded by

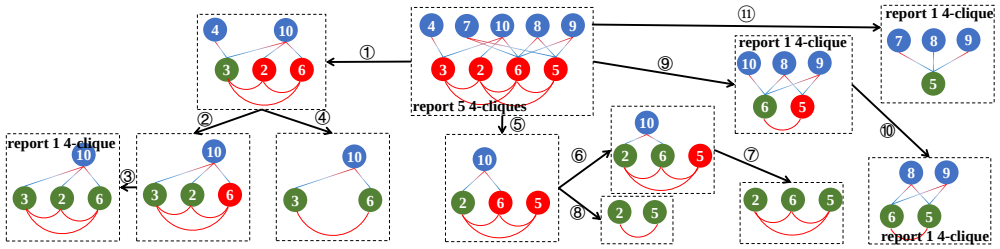


Fig. 7. The enumeration tree of the CCRKC procedure in CCRKCE.

the number of maximal cliques. As a result, Pivoter struggles to efficiently handle graphs with a large number of maximal cliques (e.g., the ComLiveJ dataset used in our experiments).

5.2 The Proposed CCRKCE Algorithm

Key Idea. As shown in the CCR-based framework in Algorithm 2, the k-cliques in the graph can be divided into two sub-cliques: K'' in C and K' in P. We design an algorithm to enumerate all cliques in P. Each x-clique in P can be expanded into a k-clique by adding $k-x$ vertices from the common neighbors in C of the x vertices. We provide the following example to illustrate this idea.

Example 5.1. The clique $\{5,6,7,8,9\}$ from the example graph in Figure 1 is a 4-clique that has $\{7,8,9\}$ in C and $\{5,6\}$ in $V \setminus C$. To find this clique, we only need to enumerate the vertices $\{5,6\}$ in $V \setminus C$, and the common neighbors of $\{5,6\}$ are $\{7,8,9\}$.

Empirical observation. In real-world networks, an observation is that the k-cliques in the core cliques (a core clique C in line 4 of Algorithm 2 contains $\binom{|C|}{k-1}$ k-cliques) account for a large proportion of the total k-cliques. The ratio of the k-cliques in core cliques among the total the k-cliques is 81.2%, 85.5%, 99.99%, 95.97%, 98.6% for the Amazon, Google, DBLP, Berkstan,Flicker datasets (Table 1 in Section 6), respectively. Therefore, removing the core cliques can reducing the enumeration space significantly.

The following lemma provides the count of the possible k-cliques that contain a partial clique R in $V \setminus C$.

Lemma 5.2. *Let R be a partial clique in the remaining subgraph, and let $C' \subseteq C$ be the set of common neighbors of all vertices in R. There are $\binom{|C'|}{k-|R|}$ k-cliques containing R.*

Proof Sketch: $C' \cup R$ forms a clique and choose $k-|R|$ vertices from the set C'.

This lemma is crucial for efficiently enumerating the cliques in the intersection of C and P. By leveraging this result, we can enumerate the k-cliques in the sparse remaining graph $G(P)$ and output the k-cliques in batches.

The CCRKCE algorithm. The algorithm firstly follows the CCR-based framework (lines 1-5 of Algorithm 2). Then, CCRKCE invokes the recursive procedure CCRKC$(C, \vec{P}, R)$ (line 5) as the implementation of lines 6-9 of Algorithm 2, where C is the core clique, $\vec{P}$ is the set of candidate vertices, and R is the partial clique under construction.

The CCRKC procedure works recursively to enumerate all cliques in the subgraph induced by $\vec{P}$, similar to Algorithm 5. The key difference between our CCRKCE and Algorithm 5 is that the common neighbors of the vertices in the partial clique R are maintained in the parameter C (line 13). When a new vertex is added to R, the core clique C is updated accordingly (line 13). Additionally, for each partial clique R and its common neighbors in C, the algorithm computes all combinations of $k-|R|$ vertices from C to form a complete k-clique, as stated in Lemma 5.2 (line 11). When R reaches size $k-1$, each vertex in C and $\vec{P}$ can be added to R to complete a k-clique, resulting in

Algorithm 6: CCRKCE

```
Input: A graph G = (V, E), an integer k
Output: The k-cliques in G
1  G⃗ ← an ordered G by the degeneracy ordering;
2  for u ∈ V do
3      C ← the core clique of G(N(u, G⃗));
4      P⃗ ← the degeneracy ordering of G(N⟨u, G⃗⟩ \ C);
5      CCRKC (C, P⃗, {u});
6  Procedure CCRKC (C, P⃗, R)
7      if |R| = k − 1 then
8          Report the |P⃗| + |C| k-cliques;
9          return;
10     if |C| + |R| ≥ k then
11         Report the (|C| choose k−|R|) k-cliques;
12     for u ∈ P⃗ do
13         CCRKC (N(u, C), N⟨u, P⃗⟩, R ∪ {u});
```

$|\vec{P}|+|C|$ distinct k-cliques (line 8). This approach efficiently enumerates all k-cliques while ensuring that each clique is considered exactly once.

Example 5.3. Figure 7 illustates the enumeration tree of the CCRKC *procedure in* CCRKCE *for counting* 4*-cliques. In each tree node (each dotted box), red vertices represent the candidate set* $\vec{P}$*, blue vertices represent the core clique* C*, and green vertices represent the partial-clique* R*, respectively. The edges between the vertices in the core clique* C*, i.e. the blue vertices, are omitted. The cycled number is the searching order of the search tree. The enumeration tree starts with* $C = \{4, 7, 10, 8, 9\}$, $\vec{P} = \langle 3, 2, 6, 5\rangle$*, and* $R = \emptyset$*. This tree node reports* $\binom{|C|}{k-|R|} = \binom{5}{4-0} = 5$ *4-cliques (line 11 of Algorithm 6). This tree node has* $|\vec{P}| = 4$ *children nodes (line 12 of Algorithm 6). To go to the first children (by ①), the vertex* 3 *is put into* R*, i.e. be green in the figure, and* C *and* $\vec{P}$ *become* $N(3, C) = \{4, 10\}$ *and* $N\langle 3, \vec{P}\rangle = N(3) \cap \vec{P} = \{2, 6\} \cap \langle 3, 2, 6, 5\rangle = \langle 2, 6\rangle$ *respectively. By ②, the graph node* 2 *is put into* R *and* 4 *is removed as the non-neighbor of* 2*. At last, report a* 4*-clique by ③, where* $|R| = 3 = k - 1$ *(line 7 of Algorithm 6).*

5.3 Analysis of Our CCRKCE Algorithm

The following lemma proves the correctness of our CCRKCE algorithm.

Lemma 5.4. CCRKCE *can solve KCEP correctly.*

Without loss of generality (w.l.g), we analyze the time complexity of the CCRKC procedure for a randomly selected vertex $u \in V$. Given a graph $G = (V, E)$, for a vertex $u \in V$ w.l.g, denote by C the core clique of the out-neighbors of u and $\vec{P}$ the remaining vertices $N\langle u, \vec{G}\rangle \setminus C$. Define $\alpha' = \alpha(G(\vec{P}))$, which is the arboricity of the remaining sparser graph.

Lemma 5.5. *Let* $f_1 = k(\alpha(G) - |C|)(\alpha' + |C|)\alpha'^{k-3}$ *and* $f_2 = k(\alpha(G) - |C|)\sum_{i=0}^{k-3} \alpha'^i \binom{|C|}{k-2-i}$*. Both* f_1 *and* f_2 *are smaller than* $k\alpha(G)^{k-1}$*.*

Proof Sketch: The proof can be established by showing that $\alpha(G) - |C|$, $\alpha' + |C|$, α', and $|C|$ are all bounded by $\alpha(G)$.

Theorem 5.6. *The time complexity of the* CCRKC *procedure is* $O(f_1 + f_2)$*, where* f_1 *and* f_2 *are defined in Lemma 5.5.*

Proof. Let $T(C, \vec{P}, r)$ represent the time complexity of the CCRKC procedure with parameter $C, \vec{P}$ and $|R| = r$. Note that $\alpha'|\vec{P}|$ is the upper bound of $|E(\vec{P})|$. We have

$$\begin{cases} T(C, \vec{P}, k-1) = k(|C| + |\vec{P}|) \\ T(C, \vec{P}, r) = k\binom{|C|}{k-r} + \sum_{u \in \vec{P}} \left(|C| + |N\langle u, \vec{P}\rangle|\right) \\ \qquad\qquad + \sum_{u \in \vec{P}} T(N(u, C), N\langle u, \vec{P}\rangle, r+1) \end{cases} \tag{2}$$

We prove that $T(C, \vec{P}, r) \leq k|\vec{P}|^{k-r} + |\vec{P}||C|\frac{|\vec{P}|^{k-1-r}-1}{|\vec{P}|-1} + \alpha'^{k-r-2}|E(\vec{P})| + k\sum_{i=0}^{k-r-1}|\vec{P}|^i\binom{|C|}{k-r-i}$. When $r = k-1$, the inequality holds clearly because the right part is $k(|\vec{P}| + |C|) + \alpha'^{-1}|E(\vec{P})|$, which is larger than $k(|\vec{P}| + |C|)$. Then, we prove it by induction:

Let $\vec{P}_u$ represent $N\langle u, \vec{P}\rangle$, and we have

$$\begin{aligned} T(C, \vec{P}, r) \leq\ & k\binom{|C|}{k-r} + |\vec{P}||C| + |E(\vec{P})| + \sum_{u \in \vec{P}} (k|\vec{P}_u|^{k-r-1} + |\vec{P}_u||C|\frac{|\vec{P}_u|^{k-2-r}-1}{|\vec{P}_u|-1} + \alpha'^{k-r-3}|E(\vec{P}_u)| \\ & + k\sum_{i=0}^{k-r-2}|\vec{P}_u|^i\binom{|C|}{k-r-1-i}) \leq k|\vec{P}|^{k-r} + |\vec{P}||C|(1 + |\vec{P}|\frac{|\vec{P}|^{k-2-r}-1}{|\vec{P}|-1}) + \\ & \alpha'^{k-r-3}\sum_{u \in \vec{P}}|E(\vec{P}_u)| + k\sum_{i=0}^{k-r-1}|\vec{P}|^i\binom{|C|}{k-r-i} \\ =\ & k|\vec{P}|^{k-r} + |\vec{P}||C|\frac{|\vec{P}|^{k-1-r}-1}{|\vec{P}|-1} + \alpha'^{k-r-2}|E(\vec{P})| + k\sum_{i=0}^{k-r-1}|\vec{P}|^i\binom{|C|}{k-r-i}. \end{aligned} \tag{3}$$

So we have

$$T(C, \vec{P}, 2) \leq k|\vec{P}|^{k-2} + |\vec{P}||C|\frac{|\vec{P}|^{k-3}-1}{|\vec{P}|-1} + \alpha'^{k-4}|E(\vec{P})| + k\sum_{i=0}^{k-3}|\vec{P}|^i\binom{|C|}{k-2-i}. \tag{4}$$

Since $|\vec{P}|$ is bounded by α' (when $r \geq 2$) and $|E(\vec{P})|$ is bounded by $\alpha'^2/2$, we have

$$T(C, \vec{P}, 2) \leq k\alpha'^{k-2} + \alpha'^{k-3}|C| + \alpha'^{k-2}/2 + k\sum_{i=0}^{k-3}\alpha'^i\binom{|C|}{k-2-i} = k\alpha'^{k-3}((1 + \frac{1}{2k})\alpha' + |C|) + k\sum_{i=0}^{k-3}\alpha'^i\binom{|C|}{k-2-i}. \tag{5}$$

At last, we have $T(C, \vec{P}, 1) = \sum_{u \in \vec{P}} T(C_u, \vec{P}_u, 2) = f_1 + f_2$.

□

An explanation of Theorem 5.6. The time complexity of the algorithm is divided into two parts: f_1 and f_2. $O(f_1)$ represents the time complexity of searching through the enumeration tree (the summary of the lines 12-13 for all recursive calls), while $O(f_2)$ corresponds to the time complexity of reporting the cliques that include vertices from the core clique C. Specifically, f_2 involves reporting the $\binom{|C|}{k-|R|}$ k-cliques in line 10 and the $|C|$ k-cliques in line 7 of Algorithm 6.

The worst-case time complexity of Algorithm 6 is in the order of $O(n(f_1 + f_2)) < O(kn\alpha(G)^{k-1})$, which matches the worst-case time complexity of Algorithm 5, where $m\alpha(G)^{k-2}$ is asymptotically equivalent to $n\alpha(G)^{k-1}$. However, as discussed in Section 3.1, α', which represents the complexity of the remaining subgraph after removing the core clique, is typically much smaller than the arboricity of the original graph $\alpha(G)$ for real-world graphs. As a result, the worst-case time complexity of Algorithm 6 is significantly lower than $O(kn\alpha(G)^{k-1})$ in practice.

Table 1. The datasets

Name	δ	n	m	Type
Tooth	7	78,136	452,591	DIMACS10
Amazon	10	403,394	2,443,408	Co-purchased
Douban	15	154,908	327,162	Social
Caida	22	26,475	53,381	Autonomous system
Google	44	916,428	4,322,051	Web
Pokec	47	1,632,803	22,301,964	Social
Gowalla	51	196,591	950,327	Social
Youtube	51	1,157,827	2,987,625	Social
Facebook	52	63,731	817,090	Social
WikiV	53	7,115	100,762	Who-votes-whom
Epinion	67	75,879	405,740	Social
Lastfm	70	1,191,805	4,519,330	Users-songs
Stanford	71	281,903	1,992,636	Web
Skitter	111	1,696,415	11,095,298	Internet topology
DBLP	113	425,957	1,049,866	Collaboration
Wtalk	131	2,394,385	4,659,565	Communication
Twitch	149	168,114	6,797,557	Social
Berkstan	201	685,230	6,649,470	Web
Orkut	253	3,072,627	117,185,083	Social
Flicker	573	105,938	2,316,948	Social

Table 2. Running time of various algorithms for MCEP

Datasets	CCRMCE	RMCE [19]	BKdegen [21]	BKrcd [32]
Tooth	**0.16**	0.17	0.41	0.22
Amazon	**0.89**	1.55	4.47	1.85
Douban	**0.06**	0.12	0.28	0.12
Caida	**0.01**	0.06	0.03	0.02
Google	**1.28**	4.12	10.50	4.50
Pokec	**22.55**	40.80	76.63	39.70
Gowalla	**0.94**	1.30	2.31	1.69
Youtube	**1.85**	4.07	6.89	3.76
Facebook	**1.59**	1.73	3.00	3.07
WikiV	**0.36**	0.37	0.54	0.67
Epinion	**1.29**	1.44	2.26	2.94
Lastfm	**3.04**	5.90	10.76	6.20
Stanford	**0.67**	3.44	4.56	2.01
Skitter	**27.46**	99.39	69.86	122.56
DBLP	**0.26**	0.61	1.97	0.75
Wtalk	**57.74**	68.23	101.73	227.75
Twitch	**240.03**	304.05	473.22	1128.65
Berkstan	**2.54**	71.47	9.97	3.19
Orkut	**1662.48**	2276.00	3164.99	6760.31
Flicker	**113.23**	151.74	285.06	717.74

Table 3. The size of the enumeration tree

Datasets	CCRMCE	BKdegen [21]	BKrcd [32]
Tooth	**474502**	763849	609701
Amazon	**1619569**	2760585	2094504
Douban	322109	491875	**279986**
Caida	**48872**	89537	49645
Google	1847410	4379484	**1648039**
Pokec	**30945772**	38680802	38146255
Gowalla	**1936331**	2840414	2592429
Youtube	**4330118**	6319853	4680116
Facebook	**2855243**	4102585	4063212
WikiV	**806068**	1058176	1052477
Epinion	**3158723**	4222883	4116704
Lastfm	**6811309**	9130563	7840932
Stanford	**1292635**	2610493	1452388
Skitter	**63519537**	103339439	101873996
DBLP	487557	1102858	**302553**
Wtalk	**163813878**	217879243	214051494
Twitch	**651444497**	913030632	913021580
Berkstan	**4072377**	7968835	4460373
Orkut	**4250966883**	5791024136	5790736949
Flicker	**353480677**	562174498	562150277

6 Experiments

All experiments were conducted on a server equipped with an AMD Ryzen Threadripper 3990X processor, 256GB of RAM, running Linux CentOS 7. All reported execution times are measured in seconds.

Algorithms for MCEP. For the MCEP, we implement our CCRMCE algorithm, as described in Algorithm 4. We compare it against three SOTA algorithms, including the BKdegen algorithm (Algorithm 3) [21], the RMCE algorithm [19], and the BKrcd algorithm [32]. The RMCE algorithm builds on BKdegen with additional graph reduction techniques to enhance efficiency. The BKrcd algorithm hybrids a top-down approach that iteratively removes vertices with non-neighbors from the candidate set. Other algorithms, such as those presented in [39] and [27], are not included in our comparison, as previous work [19] demonstrated that these methods are less efficient than the selected baselines. All algorithms are implemented in C++, and the codes for the baseline methods are downloaded from open-source repositories [19, 21, 32].

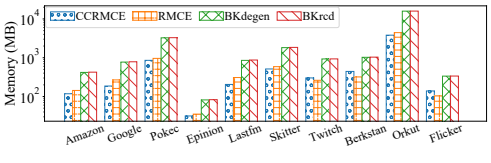


Fig. 8. Comparison of memory costs for MCEP

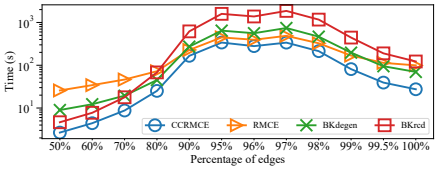


Fig. 9. Scalability of various algorithms for MCEP

Table 4. Running time of different algorithms for MCEP with the results reported

Datasets	CCRMCE	RMCE [19]	BKdegen [21]	BKrcd [32]
Amazon	**1.25**	2.38	4.12	2.13
Google	**1.83**	5.54	9.85	4.76
Stanford	**1.27**	3.94	4.92	2.58
Skitter	**87.34**	138.15	129.47	185.35
DBLP	**0.34**	0.68	1.78	0.96

Algorithms for KCEP. For the KCEP, we compare our CCRKCE algorithm with several SOTA methods, including kClist [17] as described in Algorithm 5 and Pivoter [26] (which enumerates k-cliques from large cliques). We also utilize the EBBkC algorithm [49] as the baseline, which is a method that enumerates edges to identify cliques. All algorithms are implemented in C++, and the code for the baseline methods is sourced from open-source repositories [17, 26, 49].

Datasets. Table 1 shows the datasets used in our experiments. These datasets were downloaded from the SNAP repository [30] and the Network Repository [42]. These datasets represent a wide variety of network types, including social networks, web graphs, and communication systems, providing a comprehensive set of benchmarks for evaluating the performance of our algorithms.

6.1 Experimental Results for MCEP

Running time. Table 2 presents the running times of the algorithms across a wide range of datasets. Our CCRMCE algorithm consistently demonstrates superior performance compared to the state-of-the-art methods, including RMCE, BKdegen, and BKrcd. Specifically, CCRMCE is faster across all datasets, with notable speedups observed in several cases. Moreover, we can observe that the speedup is not sensitive to the value of degeneracy. For example, on the Google dataset, CCRMCE executes in 1.28 seconds, which is 3.2× faster than RMCE (4.12s), 8.2× faster than BKdegen (10.50s), and 3.5× faster than BKrcd (4.50s), showcasing a significant speedup.

For each baseline, our CCRMCE algorithm is faster by up to 5× in many cases. For instance, CCRMCE is faster than RMCE 28.1× on the Berkstan dataset and 5.1× on the Stanford dataset. On the Google dataset, CCRMCE is 8.2× faster than BKdegen, and on the Flicker dataset, it outperforms BKrcd by 6.3×.

Overall, the result clearly demonstrates that CCRMCE provides a substantial advantage in terms of execution speed. This makes it a highly efficient and effective algorithm for solving the MCEP across a diverse range of real-world networks.

Enumeration tree size. Table 3 shows the number of the recursive calls during the enumeration of maximal cliques. We compare our CCRMCE with BKdegen and BKrcd. RMCE is not considered because it costs more operations before each tree vertex to reduce the graph size, whose enumeration tree size is smaller but each tree node costs more time [19]. As shown in Table 3, our CCRMCE has the smallest number of recursive calls on most cases, further demonstrating the effectiveness of our CCR-based framework (Algorithm 2) and novel pivot strategy (Theorem 4.6).

Memory cost. Figure 8 shows the memory cost of the algorithms for MCEP. We measure the maximum memory usage via the "ps command" in Linux. We choose 10 representative datasets and

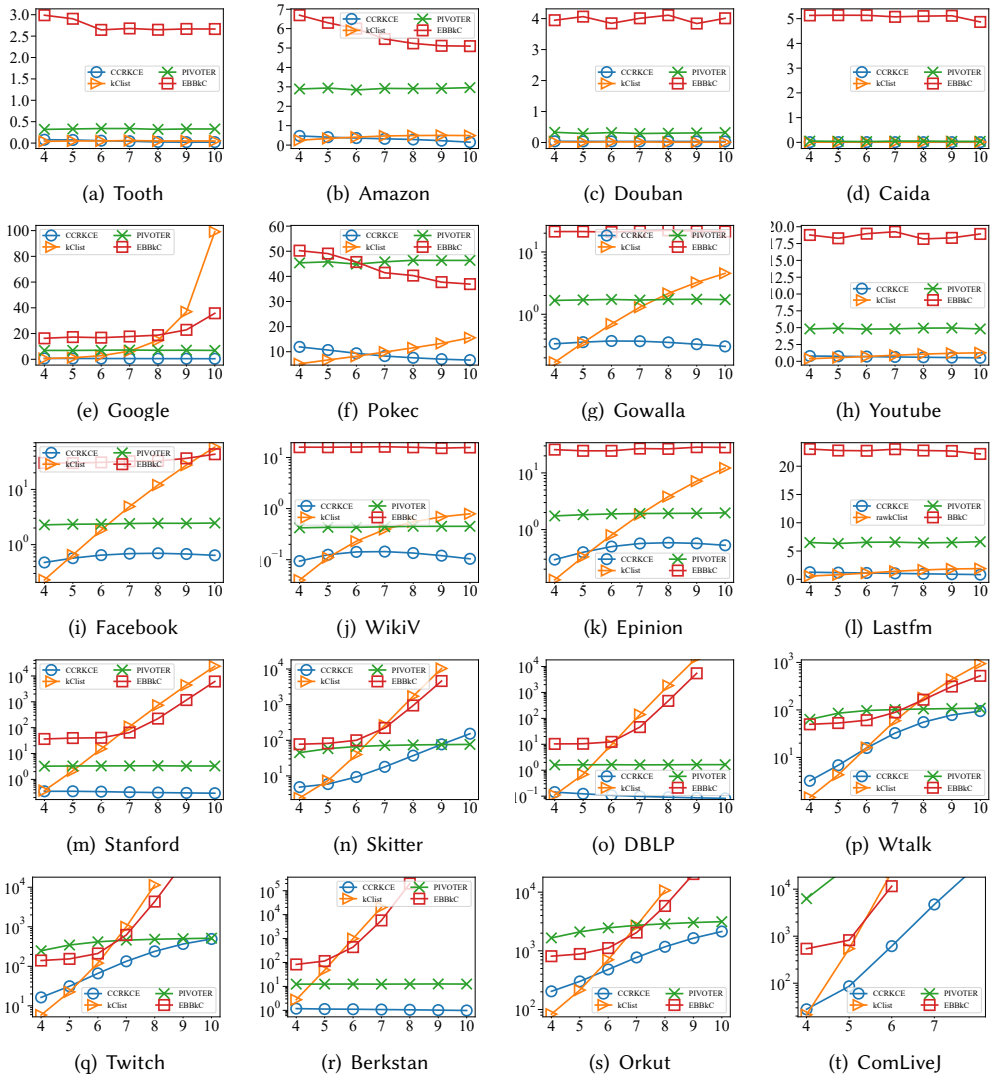
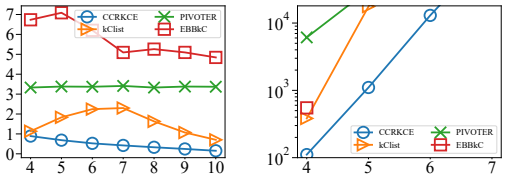


Fig. 10. Running time of different algorithms for KCEP (seconds)

the results on the other datasets are similar. As shown in Figure 8, CCRMCE, RMCE, BKdegen and BKrcd has similar memory costs in the same order of magnitude. This is because they following the same framework of BK algorithms. These results demonstrate that our algorithm is space efficient.

Scalability test. The scalability of the algorithms for MCEP is evaluated on the Skitter dataset, as shown in Figure 9. In this experiment, the x-axis represents that edges are sampled uniformly from the original Skitter dataset. As the graph size increases, the number of maximal cliques initially increases and then decreases. Specifically, when 50%, 96%, and 99.5% of the edges are sampled, the number of maximal cliques becomes 5.3×10^6, 5.4×10^8, and 5.6×10^7, respectively. This phenomenon occurs because after deleting certain edges, a maximal clique may split into many smaller maximal cliques. As a result, the number of maximal cliques can increase when a percentage of edges is removed.



(a) Amazon (b) ComLiveJ

Fig. 11. Running time of different algorithms for KCEP with the results reported ($k = 7$)

As shown in Figure 9, the running time of all algorithms follows a similar trend to the number of maximal cliques. For graphs where less than 70% of the edges are sampled, BKrcd is more efficient than both RMCE and BKdegen. However, as the graph complexity increases, RMCE becomes more efficient than BKdegen and BKrcd. Notably, our CCRMCE algorithm consistently outperforms all other algorithms. For instance, when 50% of the edges are sampled, CCRMCE takes only 57.3% of the running time of BKrcd. When 96% of the edges are sampled, CCRMCE requires just 69.4% of the running time of RMCE. These results demonstrate the high scalability and robustness of the CCRMCE algorithm.

6.2 Experimental Results for KCEP

Running time. Figure 10 compares the running times of our CCRKCE algorithm with other state-of-the-art methods. In this experiment, we include an additional dataset ComLiveJ, which consists of $n = 4,036,538$ vertices, $m = 34,681,189$ edges, and has a degeneracy of $\delta = 360$. As shown in previous studies [26], ComLiveJ is particularly challenging for k-clique enumeration tasks, especially for algorithms like Pivoter, which struggle to handle it efficiently. Therefore, we also incorporate ComLiveJ into our evaluation to assess the performance of CCRKCE under such demanding scenarios.

As shown in Figure 10, our CCRKCE consistently outperforms all baselines in most cases. Notably, as the degeneracy of the network increases, CCRKCE performs even better, significantly surpassing the other methods. For example, when $k = 6$, our algorithm achieves the fastest running time on datasets with a degeneracy greater than 52 (see Figure 10(i) to Figure 10(t)). Additionally, on datasets such as Stanford, DBLP, Berkstan, and ComLiveJ, CCRKCE is an order of magnitude faster than other algorithms when $k \geq 5$.

Compared to kClist, our CCRKCE is much faster when k is large. For example, when $k = 10$, CCRKCE is at least an order of magnitude faster than kClist on 12 datasets. There exists datasets that CCRKCE and kClist exhibit similar performance. However, both CCRKCE and kClist have nearly zero seconds on these datasets. For example, on the Tooth dataset, both CCRKCE and kClist take approximately 0.ss seconds. Additionally, CCRKCE substantially outperforms Pivoter for smaller values of k or when the network contains a large number of maximal cliques. On all datasets with degeneracy larger than 50 (from Figures 10(g) to Figure 10(t)), when $k = 4$, CCRKCE is at least an order of magnitude faster than Pivoter. Especially on the challenging dataset ComLiveJ, CCRKCE is two orders of magnitude faster. This is because our CCR technique can significantly reduce the complexity of the graph. These results demonstrate that CCRKCE provides significant speedups over existing algorithms, especially in graphs with high degeneracy, making it an efficient choice for k-clique enumeration.

Memory cost. The memory consumption of the algorithms for KCEP is shown in the Figure 12. CCRKCE demonstrates competitive memory efficiency compared to other algorithms. For most

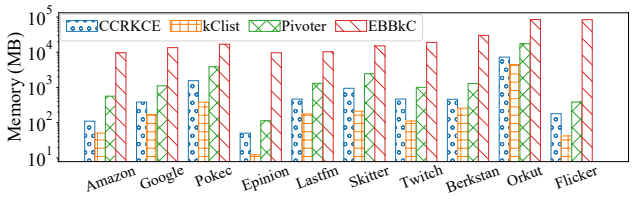
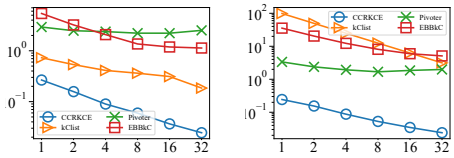
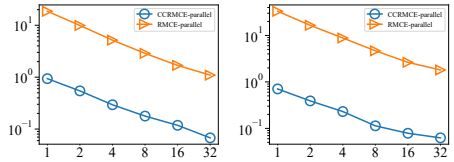


Fig. 12. Comparison of memory costs for KCEP



(a) Amazon (b) Stanford

Fig. 13. Runtime of parallel algorithms for KCEP ($k = 7$)



(a) Amazon (b) Stanford

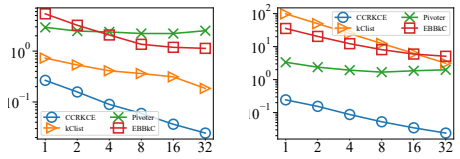
Fig. 14. Runtime of parallel algorithms for MCEP (seconds)

datasets, CCRKCE consumes significantly less memory than Pivoter and EBBkC, which show much higher memory usage, especially on large datasets such as Orkut and Flicker. For example, on the Orkut dataset, CCRKCE uses 7,266.33 MB of memory, while Pivoter requires 17,464.94 MB, and EBBkC requires a staggering 84,638.79 MB. Similarly, CCRKCE outperforms kClist and Pivoter in terms of memory usage across other datasets, such as Amazon, Google, and Pokec, where the memory consumption of CCRKCE remains consistently lower. Although kClist costs less memory than CCRKCE, they are in the same order. For example, on the Orkut dataset, CCRKCE requires 7266.33 MB and kClist requires 4474.54 MB. This indicates that, in addition to providing significant computational speedups, CCRKCE is also more memory-efficient.

6.3 Further Experimental Results

Running time of different algorithms for clique reporting. Most prior studies on clique enumeration evaluate performance by counting cliques rather than explicitly enumerating them [17, 19, 21, 32, 49]. To ensure fair comparison, we adopt the same counting mode in our experiments—incrementing a counter for each discovered clique without outputting the cliques themselves. However, to measure the runtime overhead of explicit enumeration, we introduce additional loops to report cliques for each algorithm. Table 4 and Figure 11 present the running time of various algorithms, with a representative subset of datasets shown due to space constraints (results on omitted datasets follow similar trends). Our algorithms significantly outperform state-of-the-art (SOTA) methods in both maximal clique enumeration (MCEP) and k-clique enumeration (KCEP). For instance, on the Skitter dataset (Table 4), CCRMCE reports all maximal cliques in 87.34 seconds, compared to 129.47 seconds for the SOTA algorithm. On the ComLiveJ dataset (Figure 11(b)), CCRKCE enumerates all k-cliques an order of magnitude faster than the SOTA method. These results demonstrate the high efficiency of our approach for explicit clique enumeration tasks.

Performance studies of parallel algorithms. As discussed in Section 3.2, our CCR framework is inherently parallelizable. In this experiment, we implement parallel versions of our CCRMCE and CCRKCE algorithms, and compare them against the SOTA methods for both MCEP and KCEP. For MCEP, the SOTA algorithm is the parallel RMCE algorithm [19]. For KCEP, we evaluate against three SOTA parallel algorithms: Pivoter [26], kClist [17, 31], and EBBkC [49]. With the exception of



(a) Amazon (b) Stanford

Fig. 15. Runtime of parallel algorithms for KCEP ($k = 7$)

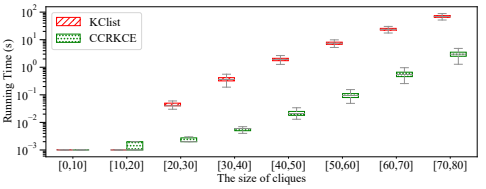


Fig. 16. Permance comparison between kClist and CCRKCE on large-clique-size controlled random graphs

parallel Pivoter, which we implemented ourselves due to the lack of an open-source implementation available, all baseline algorithms employ their original author-provided parallel implementations.

Figure 14 highlights the superior performance of parallel CCRMCE. Note that an unusual case is that the single-threaded parallel implementation of RMCE is significantly slower than its sequential counterpart (also observed in [19]). For example, on Amazon, the sequential RMCE completes in 1.55 seconds, whereas the parallel version takes 18.6 seconds. This slowdown arises from the complexity of RMCE, which relies on a sophisticated parallel hashmap [19]. In contrast, as shown in Figure 14, parallel CCRMCE remains highly efficient, requiring just 0.06 second on Amazon with 32 threads. Figure 15 compares the performance of parallel CCRKCE, Pivoter, kClist, and EBBkC for $k = 7$. Similar results can also be observed for other k values. As can be seen, our algorithm substantially outperforms all the SOTA algorithms in terms of both speed-up and running time. Notably, the weak parallel scalability of Pivoter aligns with prior observations in [26]. These results demonstrate that our CCR-based framework delivers highly efficient parallel performance.

Random graphs with controlled large cliques. To further evaluate the effectiveness of our approach, we conducted experiments on synthetic random graphs. We begin by generating a graph with 500 nodes. Given two integers, l and r, we sample 10 clique sizes from the range $[l, r]$. For each clique size $k \in [l, r]$, we randomly select k nodes to form a clique. Each remaining non-clique node is then connected to a random number of clique nodes. Finally, we generate 10^4 additional edges using the ER model. Figure 16 presents a box plot where each box represents results from over 30 random graphs. The x-axis denotes the values of l and r. When the graphs do not contain large cliques ($r \leq 20$), kClist and CCRKCE exhibit similar performance. However, as clique size increases, CCRKCE achieves significant speedups over kClist. These results further demonstrate the high effectiveness of our CCR technique in graphs containing large cliques.

Discussions. On extremely sparse graphs, where clique enumeration is inherently trivial due to the limited number of cliques, existing algorithms already achieve excellent performance, leaving little room for improvement. In such cases (e.g., the Tooth graph with degeneracy 7 in Figure 10(a)), our CCRMCE and CCRKCE algorithms demonstrate comparable efficiency to state-of-the-art methods like kClist, though without significant advantage. However, the true advantage of our approach becomes evident in more challenging real-world graphs. For small k values (e.g., $k = 4$ on ComLiveJ with degeneracy 360), CCRKCE incurs minor overhead from core clique computation, performing slightly slower than kClist but remaining within the same order of magnitude (Figure 10(t)). For larger k ($k \geq 5$), our algorithm consistently outperforms SOTA methods by substantial margins. Our experiments confirm that CCRKCE achieves significant speedups in practical scenarios where clique enumeration is non-trivial.

7 Related Work

Maximal (relaxed) clique enumeration. The BK algorithm [5] is a classic backtracking algorithm used to find all maximal cliques in an undirected graph. The pivoting strategy [47] is a optimized

version of the BK algorithm, which is particularly known for its ability to avoid unnecessary computations by pruning the search space. Eppstein et al. [21] introduce the degeneracy order and Li et al. [32] propose a top-to-down approach which repeatedly peels the vertex with the smallest degree until a clique is left. There are also studies for the parallel BK algorithm [2, 51], for the specific maximal clique types [53] and for other graph types like bipartite graphs [8, 41], uncertain graphs [13, 37] and dynamic graphs [18, 46]. The proposed CCRMCE algorithm differs significantly from all these existing techniques. Although it is based on the BK framework, it incorporates a novel CCR technique and a new pivoting strategy. These advancements make CCRMCE distinct from traditional methods and contribute to its superior performance in maximal clique enumeration tasks.

Since the clique model can be too restrictive in many applications, several relaxed models have been proposed, including the s-defective clique and s-plex [9, 10, 14, 16, 22, 55]. Similar to the BK algorithm, several pivot-based solutions have been developed for enumerating maximal s-defective cliques and maximal s-plexes, which generalize the classic pivoting technique [14, 16, 50]. Additionally, there are general algorithms designed to enumerate both maximal s-defective cliques and s-plexes [12, 15].

k-clique enumeration. The classic CN algorithm proposed in [11] is the first practical algorithm for listing all k-cliques. CN is a backtracking algorithms with worst case time complexity of $O(km\alpha(G)^{k-2})$. Danisch et al. [17] propose the kClist algorithm, which optimize the CN algorithm by degeneracy ordering. kClist has the worst case time complexity of $O(km(\alpha(G)-\frac{1}{2})^{k-2})$, which is slightly lower than than of CN. Li et al. [31] further optimize the kClist algorithm based on a color-based vertex ordering technique, but the worst case time complexity remains $O(km(\alpha(G)-\frac{1}{2})^{k-2})$. The EBBkC algorithm [49] is an edge-ordering algorithm, with lower worst case time complexity of $O(km(\tau/2)^{k-2})$, where τ represents the truss number [48] and $\tau/2 < \alpha(G) - \frac{1}{2}$. The proposed CCRKCE algorithm differs significantly from existing algorithms due to its novel CCR technique. In our CCRKCE algorithm, most k-cliques are enumerated within the core clique in a combinatorial manner, thus substantially improving the efficiency. There is also a special case for k-clique listing problem with $k = 3$, namely triangle enumeration. Triangle enumeration is a well studied problem and has many applications [11, 28, 29, 40, 44, 45]. The current algorithms for triangle enumeration problem also follows a vertex ordering based framework [28, 40, 44, 45], with the worst case time complexity of $O(\alpha(G)m)$.

8 Conclusion

In this paper, we propose a novel Core Clique Removal (CCR) technique for clique enumeration problems. Based on the CCR technique, we develop a new maximal clique enumeration algorithm, namely CCRMCE, with a carefully-designed pivoting technique. We also present a new k-clique enumeration algorithm, namely CCRKCE, based on the CCR technique. Our CCR technique can significantly reduce the complexity of induced subgraphs in degeneracy ordering, enabling both CCRMCE and CCRKCE to achieve faster and more scalable performance. Extensive experiments on 20 real-world datasets demonstrate the high efficiency of our approaches, achieving notable speedups in both maximal and k-clique enumeration tasks.

9 Acknowledgments

This work is supported by NSFC Grants U2241211, 62427808 and U24A20255. Rong-Hua Li and Guoren Wang are the corresponding authors of this paper.

References

[1] Faisal N Abu-Khzam, Nicole E Baldwin, Michael A Langston, and Nagiza F Samatova. 2005. On the relative efficiency of maximal clique enumeration algorithms, with applications to high-throughput computational biology. In *International Conference on Research Trends in Science and Technology*. 1–10.

[2] Mohammad Almasri, Yen-Hsiang Chang, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-mei W. Hwu. 2023. Parallelizing Maximal Clique Enumeration on GPUs. In *PACT*. IEEE, 162–175.

[3] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. *CoRR* cs.DS/0310049 (2003).

[4] Kamanashis Biswas, Vallipuram Muthukkumarasamy, and Elankayer Sithirasenan. 2013. Maximal clique based clustering scheme for wireless sensor networks. In *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, Australia, April 2-5, 2013*. IEEE, 237–241.

[5] Coenraad Bron and Joep Kerbosch. 1973. Finding All Cliques of an Undirected Graph (Algorithm 457). *Commun. ACM* 16, 9 (1973), 575–576.

[6] Lijun Chang. 2020. Efficient maximum clique computation and enumeration over large sparse graphs. *VLDB J.* 29, 5 (2020), 999–1022.

[7] Lijun Chang and Lu Qin. 2019. Cohesive Subgraph Computation Over Large Sparse Graphs. In *ICDE*.

[8] Jiujian Chen, Kai Wang, Ronghua Li, Hongchao Qin, Xuemin Lin, and Guoren Wang. 2024. Maximal Biclique Enumeration: A Prefix Tree Based Approach. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 2544–2556.

[9] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. 2021. Computing maximum k-defective cliques in massive graphs. *Comput. Oper. Res.* 127 (2021), 105131.

[10] Qihao Cheng, Da Yan, Tianhao Wu, Lyuheng Yuan, Ji Cheng, Zhongyi Huang, and Yang Zhou. 2025. Efficient Enumeration of Large Maximal k-Plexes. In *EDBT*. OpenProceedings.org, 53–65.

[11] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223.

[12] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. 2008. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *J. Comput. Syst. Sci.* 74, 7 (2008), 1147–1159.

[13] Qiangqiang Dai, Rong-Hua Li, Meihao Liao, Hongzhi Chen, and Guoren Wang. 2022. Fast Maximal Clique Enumeration on Uncertain Graphs: A Pivot-based Approach. In *SIGMOD*. ACM, 2034–2047.

[14] Qiangqiang Dai, Rong-Hua Li, Hongchao Qin, Meihao Liao, and Guoren Wang. 2022. Scaling Up Maximal k-plex Enumeration. ACM, 345–354.

[15] Qiangqiang Dai, Rong-Hua Li, Xiaowei Ye, Meihao Liao, Weipeng Zhang, and Guoren Wang. 2023. Hereditary Cohesive Subgraphs Enumeration on Bipartite Graphs: The Power of Pivot-based Approaches. *Proc. ACM Manag. Data* 1, 2 (2023), 138:1–138:26.

[16] Qiangqiang Dai, Rong-Hua Li, Meihao Liao, and Guoren Wang. 2023. Maximal Defective Clique Enumeration.. In *SIGMOD*.

[17] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-cliques in Sparse Real-World Graphs. In *WWW*.

[18] Apurba Das, Michael Svendsen, and Srikanta Tirthapura. 2019. Incremental maintenance of maximal cliques in a dynamic graph. *VLDB J.* 28, 3 (2019), 351–375.

[19] Wen Deng, Weiguo Zheng, and Hong Cheng. 2024. Accelerating Maximal Clique Enumeration via Graph Reduction. *Proc. VLDB Endow.* 17, 10 (2024), 2419–2431.

[20] Xiaoxi Du, Ruoming Jin, Liang Ding, Victor E. Lee, and John H. Thornton Jr. 2009. Migration motif: a spatial - temporal pattern mining approach for financial markets. In *SIGKDD*. ACM, 1135–1144.

[21] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing All Maximal Cliques in Large Sparse Real-World Graphs. *ACM J. Exp. Algorithmics* 18 (2013).

[22] Jian Gao, Zhenghang Xu, Ruizhi Li, and Minghao Yin. 2022. An Exact Algorithm with New Upper Bounds for the Maximum k-Defective Clique Problem in Massive Sparse Graphs. AAAI Press, 10174–10183.

[23] David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering Large Dense Subgraphs in Massive Graphs. In *VLDB*. 721–732.

[24] Shuo Han, Lei Zou, and Jeffrey Xu Yu. 2018. Speeding Up Set Intersections in Graph Algorithms using SIMD Instructions. In *SIGMOD*. ACM, 1587–1602.

[25] Leonidas D Iasemidis, Deng-Shan Shiau, Wanpracha Chaovalitwongse, J Chris Sackellares, Panos M Pardalos, Jose C Principe, Paul R Carney, Awadhesh Prasad, Balaji Veeramani, and Kostas Tsakalis. 2003. Adaptive epileptic seizure prediction system. *IEEE transactions on biomedical engineering* 50, 5 (2003), 616–627.

[26] Shweta Jain and C. Seshadhri. 2020. The Power of Pivoting for Exact Clique Counting. In *WSDM*.

[27] Yan Jin, Bowen Xiong, Kun He, Yangming Zhou, and Yi Zhou. 2022. On fast enumeration of maximal cliques in large graphs. *Expert Syst. Appl.* 187 (2022), 115915.

[28] Sofiane Lagraa and Hamida Seba. 2016. An efficient exact algorithm for triangle listing in large graphs. *Data Min. Knowl. Discov.* 30, 5 (2016), 1350–1369.

[29] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.* 407, 1-3 (2008), 458–473.

[30] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[31] Ronghua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering Heuristics for k-clique Listing. *Proc. VLDB Endow.* 13, 11 (2020), 2536–2548.

[32] Yinuo Li, Zhiyuan Shao, Dongxiao Yu, Xiaofei Liao, and Hai Jin. 2019. Fast Maximal Clique Enumeration for Real-World Graphs. In *DASFAA (Lecture Notes in Computer Science, Vol. 11446)*. Springer, 641–658.

[33] Zhenqi Lu, Johan Wahlström, and Arye Nehorai. 2018. Community detection in complex networks via clique conductance. *Scientific reports* 8, 1 (2018), 5982.

[34] Tsutomu Matsunaga, Chikara Yonemori, Etsuji Tomita, and Masaaki Muramatsu. 2009. Clique-based data mining for related genes in a biomedical database. *BMC Bioinform.* 10 (2009).

[35] David W. Matula and Leland L. Beck. 1983. Smallest-Last Ordering and clustering and Graph Coloring Algorithms. *J. ACM* 30, 3 (1983), 417–427.

[36] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[37] Arko Provo Mukherjee, Pan Xu, and Srikanta Tirthapura. 2015. Mining maximal cliques from an uncertain graph. In *ICDE*. IEEE Computer Society, 243–254.

[38] C St JA Nash-Williams. 1961. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society* 1, 1 (1961), 445–450.

[39] Kevin A. Naudé. 2016. Refined pivot selection for maximal clique enumeration in graphs. *Theor. Comput. Sci.* 613 (2016), 28–37.

[40] Mark Ortmann and Ulrik Brandes. 2014. Triangle Listing Algorithms: Back from the Diversion. In *ALENEX*. SIAM, 1–8.

[41] Zhe Pan, Xu Li, Shuibing He, Xuechen Zhang, Rui Wang, Yunjun Gao, Gang Chen, and Xian-He Sun. 2024. AMBEA: Aggressive Maximal Biclique Enumeration in Large Bipartite Graph Computing. *IEEE Trans. Computers* 73, 12 (2024), 2664–2677.

[42] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*.

[43] Ryan A. Rossi, David F. Gleich, and Assefaw Hadish Gebremedhin. 2015. Parallel Maximum Clique Algorithms with Applications to Network Analysis. *SIAM J. Sci. Comput.* 37, 5 (2015).

[44] Thomas Schank. 2007. *Algorithmic Aspects of Triangle-Based Network Analysis*. Ph. D. Dissertation. Karlsruhe University, Germany.

[45] Thomas Schank and Dorothea Wagner. 2005. In *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3503)*. Springer, 606–609.

[46] Shengli Sun, Yimo Wang, Weilong Liao, and Wei Wang. 2017. Mining Maximal Cliques on Dynamic Graphs Efficiently by Local Strategies. In *ICDE*. IEEE Computer Society, 115–118.

[47] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.* 363, 1 (2006), 28–42.

[48] Jia Wang and James Cheng. 2012. Truss Decomposition in Massive Networks. *Proc. VLDB Endow.* 5, 9 (2012), 812–823.

[49] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2024. Efficient k-Clique Listing: An Edge-Oriented Branching Strategy. *Proc. ACM Manag. Data* 2, 1 (2024), 7:1–7:26.

[50] Zhengren Wang, Yi Zhou, Mingyu Xiao, and Bakhadyr Khoussainov. 2022. Listing Maximal k-Plexes in Large Real-World Graphs. In *WWW*. 1517–1527.

[51] Yi-Wen Wei, Wei-Mei Chen, and Hsin-Hung Tsai. 2021. Accelerating the Bron-Kerbosch Algorithm for Maximal Clique Enumeration Using GPUs. *IEEE Trans. Parallel Distributed Syst.* 32, 9 (2021), 2352–2366.

[52] Xuyun Wen, Wei-Neng Chen, Ying Lin, Tianlong Gu, Huaxiang Zhang, Yun Li, Yilong Yin, and Jun Zhang. 2017. A Maximal Clique Based Multiobjective Evolutionary Algorithm for Overlapping Community Detection. *IEEE Trans. Evol. Comput.* 21, 3 (2017), 363–377.

[53] Qi Zhang, Rong-Hua Li, Minjia Pan, Yongheng Dai, Qun Tian, and Guoren Wang. 2023. Fairness-Aware Maximal Clique in Large Graphs: Concepts and Algorithms. *IEEE Trans. Knowl. Data Eng.* 35, 11 (2023), 11368–11387.

[54] Yalong Zhang, Ronghua Li, Qi Zhang, Hongchao Qin, Lu Qin, and Guoren Wang. 2024. Efficient Algorithms for Pseudoarboricity Computation in Large Static and Dynamic Graphs. *Proc. VLDB Endow.* 17, 11 (2024), 2722–2734.

[55] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. 2021. Improving Maximum k-plex Solver via Second-Order Reduction and Graph Color Bounding. AAAI Press, 12453–12460.

Received January 2025; revised April 2025; accepted May 2025