Scaling Up Maximal k-plex Enumeration

Qiangqiang Dai
Beijing Institute of Technology
Beijing, China
qiangd66@gmail.com

Rong-Hua Li
Beijing Institute of Technology
Beijing, China
lironghuabit@126.com

Hongchao Qin
Beijing Institute of Technology
Beijing, China
qhc.neu@gmail.com

Meihao Liao
Beijing Institute of Technology
Beijing, China
mhliao@bit.edu.cn

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggrbit@126.com

ABSTRACT

Finding all maximal k-plexes on networks is a fundamental research problem in graph analysis due to many important applications, such as community detection, biological graph analysis, and so on. A k-plex is a subgraph in which every vertex is adjacent to all but at most k vertices within the subgraph. In this paper, we study the problem of enumerating all large maximal k-plexes of a graph and develop several new and efficient techniques to solve the problem. Specifically, we first propose several novel upper-bounding techniques to prune unnecessary computations during the enumeration procedure. We show that the proposed upper bounds can be computed in linear time. Then, we develop a new branch-and-bound algorithm with a carefully-designed pivot re-selection strategy to enumerate all k-plexes, which outputs all k-plexes in $O(n^2\gamma_k^n)$ time theoretically, where n is the number of vertices of the graph and γ_k is strictly smaller than 2. In addition, a parallel version of the proposed algorithm is further developed to scale up to process large real-world graphs. Finally, extensive experimental results show that the proposed sequential algorithm can achieve up to 2× to 100× speedup over the state-of-the-art sequential algorithms on most benchmark graphs. The results also demonstrate the high scalability of the proposed parallel algorithm. For example, on a large real-world graph with more than 200 million edges, our parallel algorithm can finish the computation within two minutes, while the state-of-the-art parallel algorithm cannot terminate within 24 hours.

CCS CONCEPTS

• **Theory of computation** → **Algorithm design techniques**.

ACM Reference Format:
Qiangqiang Dai, Rong-Hua Li, Hongchao Qin, Meihao Liao, and Guoren Wang. 2022. Scaling Up Maximal k-plex Enumeration. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557444

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557444

1 INTRODUCTION

Graphs are ubiquitous in real-world applications. It is often of great value to find cohesive subgraphs from a graph in both theoretical research and practical applications. A classic model of cohesive subgraph called clique, where vertices are pairwise connected, has attracted much attention in recent decades. Many algorithms of enumerating all maximal cliques in a graph were developed, such as the classic Bron-Kerbosch algorithm [5] and its pivot-based variants [14, 22, 29], the output-sensitive algorithms [6, 10], the parallel algorithms [12, 26, 32], and so on.

Real-world networks are often noisy or faulty [11, 35]. It may be overly restrictive to find cohesive subgraphs when using the notion of clique. Thus, a set of relaxed clique models has been proposed to overcome this issue [23]. One of an interesting relaxed clique models is the k-plex which was first introduced in [28]. In particular, given a graph G, a vertex set C is a k-plex if every vertex has a degree at least $|C| - k$ within the subgraph of G induced by C. A k-plex C is said to be maximal if there does not exist any other k-plex that contains C. The problem of enumerating all maximal k-plexes from a graph has been widely studied in the literature [4, 11, 30, 33, 35], which often arises in a large number of real-world applications, such as community detection in social networks [1], identifying protein complexes in protein-protein interaction networks [19], and serving as an alternative to cliques in biochemistry [13].

However, the problem of enumerating all maximal k-plexes is NP-hard [1], and the number of k-plexes in a real-world graph is often exponential to the size of the graph, resulting in that many existing algorithms for computing maximal k-plexes can only deal with very small graphs as reported in [9, 11]. Recently, a more desirable problem of computing all relatively-large maximal k-plexes (whose size is not less than a given parameter q, e.g., $q > 10$) has also been investigated [7, 9, 11], since small-size k-plexes are often no practical use in real-world applications. For instance, Conte et al. [9] first developed an efficient algorithm using clique and k-core [27] to reduce the search space. Then, Conte et al. [11] further proposed an improved algorithm with an effective pivoting technique which was originally used in many maximal clique enumeration algorithms [5, 14]. However, the worst-case time complexity of these two algorithms is $O(n^2 2^n)$. More recently, a theoretically faster branch-and-bound algorithm was proposed by Zhou et al. [35], which

was the first algorithm with the worst-case time complexity lower than $O(n^2 2^n)$ to our knowledge. Nevertheless, when setting $k \geq 3$, all existing algorithms often require several hours to enumerate all relatively-large maximal k-plexes on middle-size real-world graphs. To our knowledge, we further notice that only the algorithm proposed in [11] supports parallelism. As shown in their experiments, such a parallel algorithm still has difficulty in processing large graphs even with 20 CPU cores. This motivates us to develop an efficient, parallel, and scalable approach to enumerate all relatively-large maximal k-plexes on large real-world graphs.

To achieve this goal, we first develop two novel upper-bounding techniques to reduce the search space, and then present a branch-and-bound enumeration algorithm based on a novel pivot re-selection technique. To further improve the scalability, we also devise a parallel version of our enumeration algorithm. In summary, we make the following contributions.

Novel upper bounds. We propose two novel upper bounds for the k-plexes that contain a set C of vertices. The key idea of our upper bounds is based on the principle that any possible maximal k-plex C' containing C can not break the rule that every vertex in C must have at least $|C'| - k$ neighbors in C'. We show that with the help of our upper bounds, many unnecessary computations can be pruned in linear time.

Efficient algorithms. We develop a new branch-and-bound algorithm to enumerate all maximal k-plexes with size no less than q. Specifically, in our algorithm, we first propose a novel pivot re-selection technique to reduce the search space, and then apply the proposed upper-bounding techniques to further prune unpromising branches. Theoretically, we prove that the worst-case time complexity of our algorithm is bounded by $O(n^2 \gamma_k^n)$, where γ_k is the branching factor of the algorithm with respect to k which is strictly smaller than 2. Finally, an efficient parallel algorithm is further designed to process large real-world graphs.

Extensive experiments. We conduct extensive experiments on real-world massive graphs to test the efficiency of our algorithm. The experiments show that our sequential algorithm can achieve up to 2× to 100× speedup over the state-of-the-art sequential algorithm on most benchmark graphs. The results also show that the speedup ratio of our parallel algorithm is almost linearly w.r.t. (with respect to) the number of threads, indicating the high scalability of the proposed parallel algorithm. For example, on the enwiki-2021 graph (with more than 300 million edges), when $k = 2$ and $q = 50$, our parallel algorithm only takes 71.2 seconds to enumerating all desired k-plexes using 20 threads, while the state-of-the-art parallel algorithm cannot terminate the computation within 24 hours under the same parameter settings. For reproducibility purpose, the source code of this paper is released at https://github.com/qq-dai/kplexEnum.

2 PROBLEM STATEMENT

Given an undirected and unweighted graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Let $n = |V|$ and $m = |E|$ be the number of vertices and edges respectively. For each vertex v, the set of neighbors of v in G, denoted by $N_v(G)$, is defined as $N_v(G) \triangleq \{u \in V | (v, u) \in E\}$. The degree of a vertex v in G, denoted by $d_v(G)$, is the cardinality of $N_v(G)$, i.e., $d_v(G) = |N_v(G)|$. Further, we define the set of non-neighbors of $v \in V$ in G as $\overline{N}_v(G) = V \setminus N_v(G)$ (Note that $v \in \overline{N}_v(G)$ for each $v \in V$). Similarly, the size of the set of non-neighbors of v in G is denoted by $\overline{d}_v(G) = |\overline{N}_v(G)|$. Let $G(S) = (S, E(S))$ be an induced subgraph of G if $S \subseteq V$ and $E(S) = \{(u, v) | (u, v) \in E, u \in S, v \in S\}$. If the context is clear, we simply use $N_v(S)$ ($d_v(S)$) and $\overline{N}_v(S)$ ($\overline{d}_v(S)$) to denote $N_v(G(S))$ ($d_v(G(S))$) and $\overline{N}_v(G(S))$ ($\overline{d}_v(G(S))$), respectively. The definition of k-plex is given as follows.

Definition 1 (k-plex). *Given an undirected graph G, a set S of vertices is a k-plex if every vertex in the subgraph $G(S)$ induced by S has a degree no less than $|S| - k$.*

A k-plex S is said to be maximal if there is no other vertex set $S' \supset S$ in G such that S' is a k-plex. The problem of enumerating all maximal k-plexes has been widely studied in the literature [4, 30, 33, 35]. Among them, [35] proposed the first algorithm with a nontrivial worst-case time guarantee based on a branch-and-bound technique. This algorithm is also the state-of-the-art as far as we know. However, in real-world applications, when using k-plexes to detect communities, there may exist many maximal k-plexes with small size that are often no practical use [11]. Therefore, it is more useful to enumerate the relatively-large maximal k-plexes for practical applications [11]. Moreover, the relatively-large k-plexes are often much compact based on the following lemma.

Lemma 1 ([28]). *Given a k-plex S of G with the size of q, the diameter of the subgraph induced by S is at most 2 if $q \geq 2k - 1$.*

By Lemma 1, it is easy to see that any maximal k-plex with size no less than $2k - 1$ must be compactly connected and has a small diameter. Therefore, similar to [11], we aims to enumerate all maximal k-plexes with size no less than a given parameter $q \geq 2k-1$. More formally, we define our problem as follows.

Size-constraint maximal k-plex enumeration. Given an undirected graph G, a positive integer k, and a size constraint $q \geq 2k-1$, the goal of our problem is to list all maximal k-plexes in G with size no less than q.

3 THE PRUNING TECHNIQUES

Before developing the algorithm, some useful properties of k-plex can be applied to improve the performance of maximal k-plex enumerations. Below, we first describe two existing pruning techniques, and then we develop several novel techniques to prune branches in k-plex enumeration.

3.1 Existing Pruning Techniques

To reduce the search space, a widely used concept of k-core can be used to filter the vertices of G that are definitely not in the k-plexes with size of no less than q. In general, a k-core of G is a maximal subgraph in which every vertex has a degree no less than k within the subgraph [27]. Based on the definition of k-core, we can easily derive the following lemma. Due to the space limit, all proofs of this paper are given in the supplementary document.

Lemma 2. *All maximal k-plexes of G with size no less than q must be contained in the $(q - k)$-core of G.*

To compute the $(q - k)$-core of G, we can make use of a peeling algorithm developed in [2], which runs in $O(m+n)$ time. To further

reduce unnecessary vertices, a more interesting property of k-plex is also discovered recently which has been successfully used in [11, 34, 35].

Lemma 3 ([11]). *Given a k-plex S of G, for each pair of vertices u and v in S, we have:*

- *if $(u,v) \notin E(S)$, $|N_u(S) \cap N_v(S)| \geq |S| - 2k + 2$;*
- *if $(u,v) \in E(S)$, $|N_u(S) \cap N_v(S)| \geq |S| - 2k$.*

In the enumeration procedure, we let S be the k-plex to be extended in G, and let C be the set of candidate vertices of G that can be added to S to form a larger k-plex. Then, given the size constraint q, we can use Lemma 3 to prune unpromising candidate vertices in C. Specifically, for each vertex $u \in C$, we iteratively check whether there is a vertex $v \in S$ such that the vertices u and v conflict with Lemma 3 in the subgraph $G(S \cup C)$. If such a vertex $v \in S$ is found, it implies that the vertex u cannot be added to S to form a large k-plex. As a consequence, the vertex u can be safely removed from C without losing any result. Such an operation can be repeated until all vertices in C meet the conditions given in Lemma 3.

As shown in [34], the time complexity of removing vertices in G using Lemma 3 is $O(lm^{1.5})$ by using a triangle listing algorithm proposed in [17], where l is the number of iterations. Clearly, such a pruning technique is often very expensive when using in the recursive enumeration procedure. In the following, we will develop several novel and efficient pruning techniques to cut the branches in the recursive enumeration procedure.

3.2 Novel Upper-bounding Techniques

Here we develop several novel techniques to derive an upper bound of the size of a k-plex that contains some given vertices. Below, we start by giving a basic upper bound for the vertices in the candidate set C in the enumeration procedure.

Lemma 4. *Given a k-plex S and the candidate set C, for each $v \in C$, the upper bound of the k-plexes containing $S \cup \{v\}$ is $|S| + k - \overline{d}_v(S) + d_v(C)$, where $\overline{d}_v(S) = |S \setminus N_v(G)|$.*

Clearly, by Lemma 4, we can prune the vertex v if the upper bound of the k-plexes containing $S \cup \{v\}$ is smaller than the given parameter q. However, such an upper bound is often very loose, as it is dependent on the degree $d_v(C)$. To improve this, we next develop two novel upper bounds of the vertices in the candidate set. The first solution is as follows. Let $N_v(C) = \{v_1, \cdots, v_d\}$ be the neighbors of v in C sorted in non-decreasing order of the size of $\overline{N}_{v_i}(S)$ where $v_i \in N_v(C)$, i.e., $|\overline{N}_{v_i}(S)| \leq |\overline{N}_{v_{i+1}}(S)|$ for each $i \in [1, d)$. Let $T_v^i(C)$ be the first i vertices in $N_v(C)$. Denote by $sup(S) = \sum_{v \in S}(k - |\overline{N}_v(S)|)$ the support number of non-neighbors of the k-plex S. Then, a tighter upper bound can be obtained according to the following lemma.

Lemma 5. *Given a k-plex S and the candidate set C, let $\omega_k(S, C)$ be the maximum size of the k-plexes containing S in $G(S \cup C)$. We have $\omega_k(S \cup \{v\}, C) \leq |S| + k - \overline{d}_v(S) + max\{i \mid \sum_{u \in T_v^i(C)} |\overline{N}_u(S)| \leq sup(S)\}$, where $v \in C$.*

Proof. Let $|S| + k - \overline{d}_v(S) + r$ be the upper bound of the k-plexes containing $S \cup \{v\}$ with respect to Lemma 5. On the contrary, we assume that there is a k-plex S' with the size of $|S| + k - \overline{d}_v(S) + r'$ containing $S \cup \{v\}$, where $r' > r$. Thus, it is easily obtained that S' contains at least r' neighbors of v in C. However, there are at most r neighbors of v in C that can be added to S with the restriction of the definition of k-plex. Thus, such assumption is contradiction, and the lemma is established. □

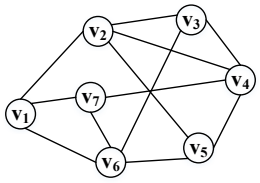

Figure 1: A running example.

It is easy to see that the upper bound given by Lemma 5 is tighter than that given by Lemma 4, and the following example further illustrates this point.

Example 1. *Given an undirected graph G shown in Fig. 1 and the parameter $k = 2$, let $S = \{v_1\}$ and $C = \{v_2, v_3, ..., v_7\}$ be the current k-plex and the candidate set in a recursion, respectively. In this example, we assume that the vertex v_2 is used to expand S. By Lemma 4, we can easily obtain that $\omega_k(S \cup \{v_2\}, C)$ is bounded by 6. However, when using Lemma 5, we have $\omega_k(S \cup \{v_2\}, C) \leq 5$. Because for the vertices in $N_{v_2}(C) = \{v_3, v_4, v_5\}$, at most two vertices can be added to $S \cup \{v_2\}$ to form a larger k-plex, which results a tighter bound on Lemma 5 compared to Lemma 4.*

Although Lemma 5 can efficiently prune many unnecessary branches in the recursive enumeration procedure, we observe that it only considers the overall acceptable non-neighbors of vertices in the k-plex S. However, when applying Lemma 5, there may exist some vertices in S that break the limit of only at most k non-neighbors. For example, given a k-plex S, a vertex $v \in C$, and a neighbor set $N_v(C) = \{u_1, u_2, u_3\}$ of v, assume that $\overline{N}_{u_1}(S) = \overline{N}_{u_2}(S) = \overline{N}_{u_3}(S) = \{v'\}$ and $sup(S)$ equals to 3 w.r.t. $k = 2$. According to Lemma 5, we cannot decrease the upper bound of the k-plexes containing $S \cup \{v\}$, while only one vertex in $N_v(C)$ may be added to $S \cup \{v\}$. Based on this analysis, we further develop an improved upper bound for each vertex in C. Let $sup(v, S)$ be the support number of non-neighbors of v in S, i.e., $sup(v, S) = k - \overline{d}_v(S)$ where $v \in S$. Then, we have the following lemma.

Lemma 6. *Given a k-plex S, the candidate set C, and $sup(w, S)$ for each $w \in S$, for each u in $N_v(C)$, where $v \in C$, we conduct the following operations: (1) get the vertex w from $\overline{N}_u(S)$ whose support number of non-neighbors in S is minimum, i.e., $sup(w, S) \leq sup(w', S)$ for each $w' \in \overline{N}_u(S)$; (2) if $sup(w, S) > 0$, we decrease $sup(w, S)$ by 1; (3) otherwise, we remove u from $N_v(C)$. Let D be the remaining vertices in $N_v(C)$, then we have $\omega_k(S \cup \{v\}, C) \leq |S| + k - \overline{d}_v(S) + |D|$.*

Proof. Suppose, on the contrary, that there is a k-plex S' containing $S \cup \{v\}$ with $|S'| > |S| + k - \overline{d}_v(S) + |D|$. Then, we can easily observe that there must be a set $D' \subseteq N_v(C)$ ($|D'| > |D|$) that can be added to $S \cup \{v\}$ to form a larger k-plex. Assume that $A = D' \setminus D$ and $B = D \setminus D'$. Then, we have that for each $u \in A$ there must exist a vertex $w \in \overline{N}_u(S)$ satisfying $sup(w, S) \leq 0$ and

Algorithm 1: Compute the upper bound

Input: a k-plex S, the candidate set C, and a vertex $v \in C$.
Output: The upper bound of k-plexes containing $S \cup \{v\}$.

```
1  for i = 0 to k − 1 do B[i] = ∅;
2  foreach u ∈ N_v(C) do B[d̄_u(S)].push(u) ;
3  foreach u ∈ S do sup(u, S) = k − d̄_u(S);
4  s ← Σ_{u∈S}(k − d̄_u(S));
5  ub ← |S| + k − d̄_v(S);
6  for i = 0 to k − 1 do
7      for u ∈ B[i] s.t. s ≥ i do
8          w ← arg min{sup(w, S)|w ∈ N̄_u(S)};
9          if sup(w, S) > 0 then
10             ub ← ub + 1; s ← s − i;
11             sup(w, S) ← sup(w, S) − 1;
12 return ub;
```

$w \in \overline{N}_{u'}(S)$ where $u' \in B$. Otherwise, the vertex u can be added to D according to Lemma 6. This indicates that the set A must be the empty set, which is contradiction to the hypothesis. Thus, the lemma is established. □

The following example further clarifies the idea of Lemma 6.

Example 2. *Given an undirected graph G shown in Fig. 1 and the parameter $k = 3$, let $S = \{v_1, v_7\}$ and $C = \{v_2, v_3, ..., v_6\}$ be the current k-plex and the candidate set of S in a recursion, respectively. Suppose that the vertex v_2 will be added to S in current recursion. The neighbor set $\{v_3, v_4, v_5\}$ of v_2 in C will be checked sequentially by Lemma 6. Before that, we first compute the support number of non-neighbors of each vertex in S (i.e., $sup(v_1, S) = 2, sup(v_7, S) = 2$). Then, when checking the first vertex v_3, we only decrease $sup(v_1, S)$ by 1, and for the second vertex v_4, $sup(v_1, S)$ also needs to be decreased, which leads to $sup(v_1, S) = 0$. However, when checking the third vertex v_5, it would be removed from the neighbor set. Finally, only two vertices are left. Thus, we have $\omega_k(S \cup \{v_2\}, C) \leq 6$. Here if we make use of Lemma 5 to generate the upper bound of $\omega_k(S \cup \{v_2\}, C)$, we can obtain that $\omega_k(S \cup \{v_2\}, C) \leq 7$. This example demonstrates that Lemma 6 works better than Lemma 5.*

We observe that a tighter upper bound can be further derived by combining Lemma 5 with Lemma 6. The key idea is that when checking the i-th vertex in $N_v(C)$ whether it can be added to S using Lemma 5, we also make use of Lemma 6 to detect the vertex whether there is a conflict with the vertices in $\overline{N}_v(S)$. Based on this idea, we propose an upper-bounding algorithm which is shown in Algorithm 1.

In Algorithm 1, it first makes use of a bucketing array B to store the vertices in $N_v(C)$, where $R[i]$ contains all vertices in $N_v(C)$ that have i non-neighbors in S (lines 1-2). Then, the algorithm computes the support number of non-neighbors of each vertex in S (line 3). After that, the algorithm sequentially checks each vertex in B whether it can be added to the current k-plex S based on Lemma 5 and Lemma 6 (lines 4-11). Specifically, when checking the vertex v from $B[i]$ (line 7), the algorithm first selects w from $\overline{N}_v(S)$ with the minimum support number of non-neighbors (line 8). If there is no conflict between w and u with respect to the definition of k-plex, the algorithm updates the corresponding values (lines 9-11). Otherwise, the vertex u will not be considered. The algorithm continues the computations until all vertices have been examined. It is easy to derive that the time complexity of Algorithm 1 is bounded by $O(|S|+\log k|C|) \approx O(|S|+|C|)$, as k is often a very small constant for most real-world applications (e.g., $k \leq 10$).

4 A NEW BRANCH-AND-BOUND ALGORITHM

In this section, we present the detailed framework of our branch-and-bound enumeration algorithm. Let S and C be the current k-plex and the candidate set in a recursion, respectively. Our idea is that we first select a pivot vertex v from $S \cup C$ according to the following rules: (i) the selected pivot vertex has the minimum degree in $G(S \cup C)$; and (ii) if there are multiple vertices that has the minimum degree in $G(S \cup C)$, we select the vertex among them that has maximum $\overline{d}_v(S)$. Then, we check whether the selected pivot vertex is contained in S. If so, we will re-choose a pivot vertex from the set $\overline{N}_v(C)$ using the same rules. Next, we make use of the (re-chosen) pivot vertex to conduct the branch-and-bound procedure. The detailed implementation of this algorithm is shown in Algorithm 2.

In Algorithm 2, it first computes the $(q-k)$-core G' of G, since all maximal k-plexes with size no less than q are contained in G' (Lemma 1). Then, the algorithm sorts the vertices in V' with the degeneracy ordering $\{v_1, ..., v_{n'}\}$ (i.e., the vertex-removing ordering in the peeling algorithm for k-core decomposition [2]) (line 1). Following the degeneracy ordering, the algorithm sequentially processes the vertices to enumerate all maximal k-plexes by invoking the procedure Branch, which admits five parameters: S, C, X, k and q (lines 2-6), where S, C, and X are the three disjoint sets, respectively. In particular, S is the current k-plex, C is the candidate vertex set in which the vertex can be used to expand S, and X is the set of excluded vertices that have already been explored in the previous recursions. Initially, C (X) is set as the set of 2-hop neighbors of v_i (the set of vertices whose distance from v_i is no larger than 2, denoted by $N^2_{v_i}(G')$) that come after (before) v_i (line 3-4), where v_i is the i-th vertex in the degeneracy ordering. This is because the diameter of any desired maximal k-plex is no larger than 2 by Lemma 1. Then, the algorithm applies the results in Lemma 3 to prune vertices in C and X (line 5). After that, the algorithm invokes the recursive procedure Branch to enumerate all maximal k-plexes containing v_i.

In Branch, if $C \cup X$ is an empty set and the size of S is no less than q, it outputs S as a result (lines 8-10). Otherwise, it executes the branch-and-bound procedure (lines 11-24). In particular, the algorithm first selects a vertex v with the maximum $\overline{d}_v(S)$ from the subset of $S \cup C$ that has minimum degree in $G(S \cup C)$ as a pivot vertex (lines 11-12). If such a pivot vertex is already in S (i.e., $v \in S$), the algorithm re-chooses a pivot vertex from $\overline{N}_v(C)$ using the same rules (lines 17-18). Then, the algorithm computes the upper bound of k-plexes containing $S \cup \{v\}$ by using Algorithm 1 (line 19). If such an upper bound is no less than q, the algorithm recursively invokes the Branch procedure by expanding S with v and updating C (X) such that for each $w \in C'$ ($w \in X'$), $S \cup \{v, w\}$ is a k-plex (lines 20-22). After that, the algorithm performs the other recursive

Algorithm 2: The branch-and-bound algorithm

Input: The graph G and two parameters k and $q \geq 2k-1$
Output: All maximal k-plexes in G with size no less than q

1 Let $G' = (V', E')$ be the $(q-k)$-core subgraph of G; and let $\{v_1, ..., v_{n'}\}$ be the degeneracy ordering of V';
2 **for** $i = 1$ *to* $n' - q + 1$ **do**
3 $C \leftarrow \{v_{i+1}, ..., v_{n'}\} \cap N^2_{v_i}(G')$;
4 $X \leftarrow \{v_1, ..., v_{i-1}\} \cap N^2_{v_i}(G')$;
5 Pruning vertices in C and X using Lemma 3;
6 Branch$(\{v_i\}, C, X, k, q)$;
7 **Procedure:** Branch(S, C, X, k, q)
8 **if** $C = \emptyset$ **then**
9 **if** $|S| \geq q$ *and* $X = \emptyset$ **then** Output S;
10 **return**;
11 Let D be the subset of $S \cup C$ whose degree is minimum in $G(S \cup C)$;
12 Pick a pivot vertex v from D with the maximum $\overline{d}_v(S)$;
13 **if** $d_v(S \cup C) \geq |S \cup C| - k$ **then**
14 **if** $S \cup C$ *is a maximal k-plex in G* **then**
15 **if** $|S \cup C| \geq q$ **then** Output $S \cup C$;
16 **return**;
17 **if** $v \in S$ **then**
18 Re-pick a pivot vertex u from $\overline{N}_v(C)$ using the same rules as used in lines 11-12; then set $v \leftarrow u$;
19 Compute the upper bound ub for the pivot vertex v by invoking Algorithm 1;
20 **if** $ub \geq q$ **then**
21 Get $C' \subset C$ and $X' \subseteq X$ such that $S \cup \{v, w\}$ is a k-plex for each $w \in C' \cup X'$;
22 Branch$(S \cup \{v\}, C', X', k, q)$;
23 Branch$(S, C \setminus \{v\}, X \cup \{v\}, k, q)$;

branch by moving v from C to X (line 23). Finally, the algorithm terminates until the candidate set is empty or the minimum degree of $G(S \cup C)$ is no less than $|S \cup C| - k$ (line 8 or line 13).

4.1 Complexity Analysis

Here we first analyze the time complexity of Algorithm 2, which is shown in Theorem 1.

Theorem 1. *Given a graph G and two parameters k and q, Algorithm 2 returns all maximal k-plexes with size no less than q in time $O(n^2\gamma_k^n)$, where $\gamma_k < 2$ is the maximum positive real root of $x^{k+2} - 2x^{k+1} + 1 = 0$ (e.g. $\gamma_1 = 1.618$, $\gamma_2 = 1.839$, and $\gamma_3 = 1.928$).*

Proof. It can be seen that the time complexity of Algorithm 2 is relative to the time taken for each recursion multiplied by the total number of recursions in the algorithm. Thus, the key problem is to detect bounds on the number of branches in the algorithm. The detailed analysis are as follows. Let $T(z, H)$ be the number of branches of Branch(S, C, X, q, k), where z is the size of $|C|$, and H is the subgraph of G induced by $S \cup C$. Denote by H_i $(i \geq 1)$ the induced subgraph of H that a pivot vertex in Branch$(S \cup \{v_1, ..., v_{i-1}\}, C \setminus \{v_1, ..., v_{i-1}\}, X, q, k)$ is removed, where $\{v_1, ..., v_{i-1}\} = \emptyset$ if $i = 1$. Then, we have the following recursive inequation:

$$T(z, H) \leq T(z-1, H) + T(z-1, H_1) \tag{1}$$

Clearly, the branching factor of Eq. (1) is 2, which means that the time cost of Algorithm 2 is bounded by $O(P(n)2^n)$, where $P(n)$ is the worst-case running time of each branch in Algorithm 2. However, a tighter bound can be obtained through the following analysis.

1. If $C = \emptyset$, it is easy to see that the time cost of $T(z, H)$ is a constant. Then, we consider the other situations after a pivot v is picked from $G(S \cup C)$.
2. If $d_v(S \cup C) \geq |S \cup C| - k$, it impels that $S \cup C$ is exactly a k-plex and the time cost of $T(n, H)$ is the maximal detection operations for $S \cup C$, which is bounded by $O(P(n))$.
3. If $v \in S$, a vertex u from $\overline{N}_v(C)$ is chosen for branching. Interestingly, we observe that a pivot vertex v in Branch(S, C, X, q, k) is still the pivot vertex in the sub-branch Branch$(S \cup \{u\}, C \setminus \{u\}, X, q, k)$ in worst-case. Then, Eq. (1) can be further revised with the following recursive inequation:
$$T(z, H) \leq T(z-1, H_1) + T(z-2, H_2) + T(z-2, H) \tag{2}$$
We can perform similar substitutions on the sub-branches of Branch$(S \cup \{u\}, C \setminus \{u\}, X, q, k)$, until the final branch Branch$(S \cup P, C \setminus P, X, q, k)$ is obtained, where $|P| + \overline{d}_v(S) = k$. Let $\overline{d}$ be the number of non-neighbors of v in C. The maximum size of the candidate set of $S \cup P$ is $z - \overline{d}$, since only the neighbors of v are left in this recursion. Finally, we have:
$$T(z, H) \leq \sum_{i=1}^{p} T(z-i, H_i) + T(z-\overline{d}, H') \tag{3}$$
where $p \leq k-1$ and $\overline{d} > p$, and H' the subgraph of H induced by $S \cup C \setminus \overline{N}_v(C)$.
4. If $v \notin S$, the vertex v can be immediately used for branching. According to Eq. (1), the vertex v can be the pivot vertex in Branch$(S \cup \{v\}, C \setminus \{v\}, X, q, k)$. Then, combining with Eq. (3), we have:
$$T(z, H) \leq \sum_{i=1}^{p+1} T(z-i, H_i) + T(z-\overline{d}-1, H') \tag{4}$$
where $p \leq k-1$ and $\overline{d} > p$.

Note that in the worst case, p and $\overline{d}$ are $k-1$ and k, respectively. Based on the theoretical result in [15], the branching factor γ_k of $T(z, H)$ is the largest real root of function $x^{k+2} - 2x^{k+1} + 1 = 0$. As a result, the total time cost of Algorithm 2 can be bounded by $O(P(n)\gamma_k^n)$. In each recursion, the time cost is dominated by the update operation (line 21) and the maximality checking operation (line 14), which are bounded by $O(n^2)$. Putting it all together, we have the time complexity of Algorithm 2. □

By Theorem 1, we notice that the time complexity of Algorithm 1 is the same as that in [35]. In contrast, the recursive inequation of our algorithm is quite implicit, which requires a more careful analysis. Then, we show the space complexity of the algorithm in the next theorem.

Theorem 2. *The space complexity of Algorithm 1 is bounded by* $O((\frac{\delta d_{max}}{q-2k+2})^2 + n + m)$.

Proof. We observe that the maximum size of the candidate set C is bounded by $O(\frac{\delta d_{max}}{q-2k+2})$ [11], where δ and d_{max} are the degeneracy and the maximum degree of an input graph G, respectively. Since Algorithm 2 runs with a DFS (depth-first search) manner, the total space consuming of all recursive calls can be bounded by $O(|C|^2)$. Thus, we proved the space complexity. □

Remark. By Theorem 2, the maximum size of the candidate set in Algorithm 2 is bounded by $\frac{\delta d_{max}}{q-2k+2}$, so the time complexity of Algorithm 2 can be further improved to $O(nn'^2\gamma_k^{n'})$, where $n' = min(\frac{\delta d_{max}}{q-2k+2}, n)$. Moreover, we notice that the time complexity of Algorithm 2 is dominated by $T(z, H)$, which can be further determined by p and $z - \overline{d}$ based on Eq. (3) and Eq. (4) when k is given. More specifically, the smaller p (or $z - \overline{d}$) yields a smaller branching factor of $T(z, H)$. In our pivot algorithm, the pivot vertex v has a minimum degree in $G(S \cup C)$ and also has the maximum $\overline{d}_v(S)$ among all vertices who have the minimum degree in $G(S \cup C)$, which makes both $z - \overline{d}$ and p as small as possible. As a result, our pivot technique can achieve better practical performance compared to the existing pivot method [35].

4.2 Parallel Enumeration Strategy

Here we propose a simple but effective parallelization strategy for our enumeration algorithm when running on the multi-core machines. To achieve this, we need to split the computations into multiple independent sub-tasks. In Algorithm 2, we can see that the branches that enumerate k-plexes containing the particular set S are completely independent. So, a general parallel computation scheme is to divide the entire task into n sub-tasks, and then we dynamically assign these sub-tasks to each thread to complete the computations (i.e., run lines 2-6 of Algorithm 2 in parallel). However, the computational workloads of some threads may be very large, leading to an inefficient parallel algorithm. The main reason for this is that the computational cost of each branch $\mathsf{Branch}(S, C, X, q, k)$ is very different, which is mainly determined by the size of the set C and the internal structural of the subgraph $G(S \cup C)$. Thus, in this paper, we develop an alternative solution to achieve a better load balancing. The details are as follows.

In our implementation, we first make use of the above-mentioned general scheme to start the parallel computation. Then, during the branching calculations, the algorithm also monitors the work status of each thread. If the algorithm finds that a thread to be idle, some threads would divide the task being executed into two sub-tasks. The first sub-task is the sub-branch that computes the maximal k-plexes containing the pivot vertex, and the other sub-task is to compute the maximal k-plexes excluding the pivot vertex. Note that these two sub-tasks are also independent of each other. Thus, the newly generated sub-task can be safely assigned to the idle thread. This task division scheme is performed iteratively until all threads finished the computations. As shown in the experiments, such a parallel algorithm can achieve a very good speedup ratio over our sequential algorithm.

5 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the efficiency of the proposed algorithm. Since this paper only focuses on improving the performance of maximal k-plex enumeration, we did not show the effectiveness testing for the maximal k-plex model. Below, we first describe the experimental setup and then report the results.

5.1 Experimental Setup

We implement our algorithm, called *FP*, in C++ to enumerate all maximal k-plexes, which combines the proposed upper bounding techniques and the pivot re-selection technique. For comparison, we use two state-of-the-art algorithms *D2K* [11] and *CPlex* [35] as baselines, since all the other existing algorithms [4, 9, 30] are less efficient than these two algorithms as shown in [11, 35]. The C++ codes of *D2K* and *CPlex* are provided by their authors, thus we use their original implementations in our experiments. All experiments are conducted on a PC with 2.2 GHz AMD CPU (20 cores) and 128GB memory running CentOS operating system.

Datasets. In the experiments, we use three sets of graphs to evaluate the efficiency of the proposed algorithms. The first set of graphs is the real-world massive graphs containing 139 undirected and simple graphs collected from the Network Repository [25]. The datasets can be downloaded from (http://lcs.ios.ac.cn/~caisw/graphs.html), and are widely used for evaluating the performance of k-plex search algorithms [7, 16]. The second set of graphs is the DIMACS graphs (http://archive.dimacs.rutgers.edu/pub/challenge/), which are the well-known benchmark graphs for the test of maximal clique enumeration. The last set of graphs is the large real-world graphs, which are detailed in Table 4.

Parameters. In all algorithms, there are two parameters: k and the size constraint q. In the experiments, we select the parameters k and q from the intervals of 2 to 5 and 10 to 30, respectively, as used in [35] for small or middle-size graphs. For large graphs, we adaptively set q to find relatively-large k-plexes.

5.2 Experimental Results

Exp-1: Efficiency of different sequential algorithms on 7 benchmark graphs. Here we compare the efficiency of different algorithms using 7 datasets selected from 139 benchmark graphs, because most of these 7 selected datasets have also been used as the benchmark datasets to evaluate different k-plex enumeration algorithms in [11, 35]. Table 1 shows the running time of each algorithm with varying k and q, where $k = 2, 3, 4$ and $q = 12, 20, 30$. If the algorithm can not terminate within 24 hours, we simply set its running time to "INF". From Table 1 , we can observe that our algorithm consistently outperforms *CPlex* on all datasets. In addition to a few results that are easy to be obtained by all algorithms, our algorithm is also much faster than *D2K*. In general, our algorithm can achieve 2× to 100× speedup over the state-of-the-art algorithms on most benchmark graphs. Moreover, the speedup of *FP* increases dramatically with the increase of k. This is because the proposed upper-bounding technique is the very effective in pruning unnecessary branches during the enumeration procedure. For instance, when $k = 3$ and $q = 30$, the speedups of *FP* over *CPlex*

Table 1: Running time of various sequential algorithms on 7 benchmark graphs.

Dataset (n,m)	k	q	#k-plexes	Running time (sec)			Dataset (n,m)	k	q	#k-plexes	Running time (sec)		
				FP	*CPlex*	*D2K*					*FP*	*CPlex*	*D2K*
DBLP (317080, 1049866)	2	12	12544	0.12	2.3	**0.1**	Slashdot (82144, 500480)	2	12	27208777	**61.95**	220.24	184.86
		20	5049	0.06	0.56	**0.05**			20	11411028	**25.39**	111.27	119.81
	3	12	3003588	**4.83**	17.66	7.77			30	453	**0.2**	5.45	14.90
		20	2141932	**3.33**	11.85	6.04		3	12	2807943240	**6564.55**	25721.92	19088.85
	4	12	610150817	**727.06**	2929.28	1760.42			20	1303148522	**2641.46**	14903.76	12045.24
		20	492253045	**576.8**	2489.0	1507.37			30	1679468	**14.78**	141.95	1854.576
		30	12088200	**21.82**	108.76	132.82		4	30	502699966	**1852.7**	26017.12	INF
EmEuAll (265214, 420045)	2	12	412779	**1.25**	4.61	9.23	WikiVote (8298, 100761)	2	12	2919931	**14.96**	41.11	101.09
		20	0	**0.1**	0.67	0.44			20	52	**0.26**	2.24	10.25
	3	12	32639016	**82.24**	337.79	835.58		3	12	458153396	**1948.39**	6885.94	15510.07
		20	2637	**0.26**	3.24	48.14			20	156727	**10.95**	149.58	1646.13
	4	12	1940182978	**5345.09**	31962.78	76634.4		4	20	46729532	**421.86**	30153.31	INF
		20	1707177	**10.68**	307.62	3388.93			30	0	**0.01**	2.24	0.05
Epinions (75879, 508837)	2	12	49823056	**211.86**	475.79	624.26	Pokec (1632803, 30622564)	2	12	7679906	**48.27**	1040.74	167.02
		20	613	**13.44**	56.75	158.64			20	94184	**7.19**	666.88	14.14
	3	20	548634119	**1746.34**	9637.53	28800.4			30	3	3.76	409.24	**2.78**
		30	16066	**3.37**	78.64	2172.89		3	12	520888893	**1442.79**	6861.47	12890.4
	4	30	13172906	**133.19**	20444.33	INF			20	5911456	**27.43**	820.87	847.06
Caida (26475, 53381)	2	12	5336	**0.04**	0.08	0.22			30	5	4.07	443.52	**3.87**
	3	12	281251	**0.74**	3.05	12.52		4	20	318035938	**885.44**	13090.41	INF
	4	12	15939883	**40.5**	185.16	724.91			30	4515	**4.46**	489.5	82.98

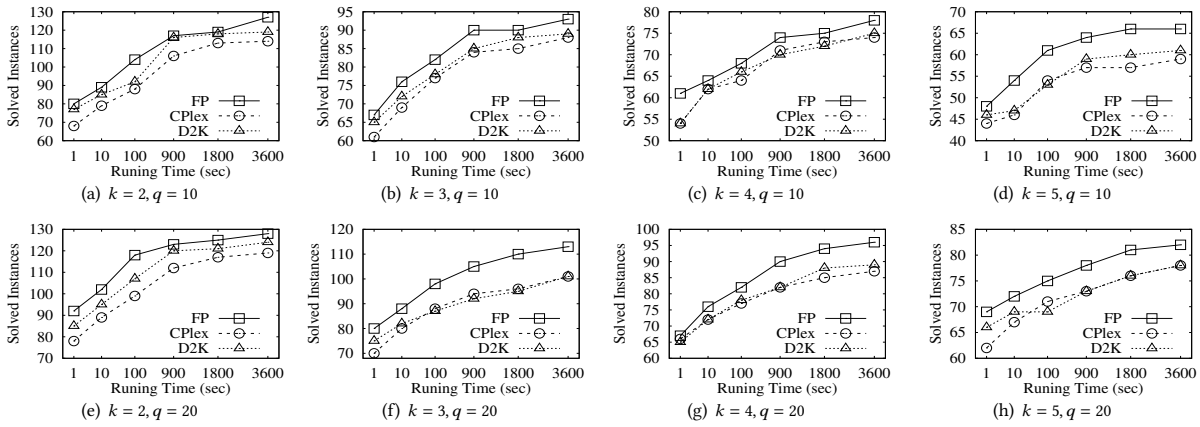

Figure 2: Number of solved instances on 139 real-world benchmark graphs under different time thresholds (in seconds).

and *D2K* are 23× and 644× respectively on Epinions. On the same dataset, when $k = 4$ and $q = 30$, the speedup of *FP* over *CPlex* increases to 153×, and *D2K* even cannot finish the computation within 24 hours. These results demonstrate the high efficiency of the proposed algorithm.

Exp-2: Efficiency of different sequential algorithms on real-world graphs. We test the number of solved instances of each algorithm on 139 real-world benchmark graphs to further compare the performance of different algorithms. Fig. 2 shows the experimental results under different time thresholds with varying k and q. As can be seen, *FP* solves the most number of instances among all algorithms with all parameter settings. When comparing with *CPlex* and *D2K*, we observe that *D2K* is usually superior to *CPlex* when $k \leq 3$, because *D2K* is tailored for processing sparse real-world graphs. This result is consistent with the result shown in Table 1. However, we can see that both *CPlex* and *D2K* still cannot keep up with our algorithm. This result further confirms that the proposed algorithm is very efficient to process real-world graphs. In addition, when computing large maximal k-plexes ($q = 20$), the gap in the number of solved instances between *FP* and the state-of-the-art algorithms becomes very large on most parameter settings. For example, when $q = 20$ and $k = 4$, *FP* solved 90 instances with a time limitation of 900 seconds, while both *CPlex* and *D2K* solved 82 instances. The reason behind it is that the proposed upper-bounding techniques can reduce a large number of unnecessary computations and the pivot re-selection technique can further reduce the size of the candidate set.

Exp-3: Efficiency of different sequential algorithms on DIMACS graphs. Here we also evaluate the performance of various algorithms on DIMACS graphs. Table 2 shows the experimental

Table 2: Running time of various sequential algorithms on DIMACS benchmarks (in seconds).

Dataset (n,m)	q	k = 2				k = 3			
		#k-plexes	*FP*	*CPlex*	*D2K*	#k-plexes	*FP*	*CPlex*	*D2K*
brock200-2	10	2002262	**22.24**	63.01	166.55	2668680146	**7030.18**	21218.15	41532.52
(200,19752)	20	0	**1.04**	7.93	41.92	0	**46.95**	787.89	10862.64
brock200-4	10	5775998682	**17048.28**	53907.15	36944.82	—	INF	INF	INF
(200,26178)	20	2	**215.63**	1097.01	9028.12	19227523	**85958.46**	INF	INF
c-fat200-5	10	5721	**0.13**	0.14	0.16	1086435	**5.28**	22.04	12.80
(200,16946)	20	5721	**0.09**	0.12	0.13	1086435	**3.66**	18.15	12.60
c-fat500-5	10	15642	0.44	0.34	**0.23**	3576858	**24.21**	44.97	26.94
(500,46382)	20	15642	0.4	0.28	**0.23**	3576858	**23.39**	41.24	25.8
c-fat500-10	10	31258	3.91	5.1	**3.47**	29552680	**574.5**	2585.94	1259.89
(500,93254)	20	31258	**2.61**	4.98	3.42	29552680	**524.66**	2533.09	1254.16
johnson8-4-4	10	16047210	**30.39**	76.71	63.14	2019800107	**5279.58**	17536.71	12888.40
(70,3710)	20	0	**0.17**	0.88	10.68	0	**26.1**	238.38	3650.27
MANN_a9	10	2160546	**2.51**	10.25	6.75	16619686	**44.05**	395.74	75.39
(45,1836)	20	1738656	**2.05**	7.06	6.72	16619686	**42.8**	397.75	74.83
p_hat300-1	10	24	**0.73**	3.3	7.97	382654	**52.27**	330.47	1022.41
(300,21866)	20	0	**0**	0.02	0.01	0	**0.22**	3.27	10.98
p_hat700-1	10	422470	**141.89**	545.85	1253.89	3475000381	**40311.47**	INF	INF
(700,121998)	20	0	**15.94**	225.37	505.94	0	**1343.98**	34751.76	INF

Table 3: Running time of *FPubs* and *FP* (in seconds).**

Datasets	q	k = 3		k = 4	
		*FP**ubs*	*FP*	*FP**ubs*	*FP*
EmEuAll	12	109.4	**82.24**	12219.2	**5345.09**
	20	0.49	**0.26**	50.51	**10.68**
Epinions	12	49621.4	**42414.6**	INF	INF
	20	2257.03	**1746.34**	INF	INF
	30	9.17	**3.37**	1892.27	**133.19**
WikiVote	12	3060.24	**1948.39**	INF	INF
	20	31.48	**10.95**	6315.26	**421.86**
Pokec	12	2069.68	**1442.79**	INF	INF
	20	36.62	**27.43**	2455.05	**885.44**
	30	4.45	**4.07**	5.23	**4.46**

results of *FP*, *CPlex* and *D2K* on 9 DIMACS benchmark graphs with $k = 2, 3$ and $q = 10, 20$. From Table 2, it is easy to see that our algorithm is much faster than *CPlex* and *D2K*, except for a few results that are easy to be obtained by all algorithms. Similar to the results in Exp-1, as k or q increase, the speedup ratio of our algorithm relative to the state-of-the-art algorithms also increases accordingly. More specifically, when $k = 2$ and $q = 20$, on brock200-2, *FP* runs 7.6 times and 40 times faster than *CPlex* and *D2K*, respectively, while when increasing the parameter k to 3, we observe that the speedup ratios of *FP* over *CPlex* and *D2K* on the same dataset are 16.7× and 230× respectively. These results further demonstrate the high efficiency of the proposed algorithm.

Exp-4: The effect of the proposed upper bound techniques. Here we conduct an ablation experiment to study the effect of the upper-bounding techniques used in our algorithm. Let *FP**ubs* be the proposed branch-and-bound algorithm without using our upper-bounding techniques (i.e., Lemma 5 and Lemma 6). Table 3 depicts the running time of *FP**ubs* and *FP* on four datasets with varying k and q. The results on the other datasets are consistent. From Table 3, we can see that the running time of *FP**ubs* is consistently higher than that of *FP* on all datasets. This is because the proposed upper bounds are very effective and easy to compute. Moreover, with the increase of q, the speedup of *FP* compared to *FP**ubs* usually increases, indicating that our upper bounds have a stronger pruning performance for a larger q. When comparing the results shown in Table 1, we can see that *FP**ubs* is consistently faster than *CPlex* on all datasets and also faster than *D2K* on most datasets with most parameter settings. This result indicates that the proposed pivot re-selection technique is indeed very effective as analyzed in Section 4.1.

Exp-5: The speedup ratio of our parallel algorithm. In this experiment, we evaluate the speedup ratio of our parallel algorithm on benchmark graphs. Fig. 3 shows the results on three benchmark datasets, and similar results can also be observed from the other datasets. As can be seen, the speedup ratio of our parallel algorithm is almost linear w.r.t. the number of threads used. More specifically, our parallel algorithm running with 20 threads is more than 15 times faster than the corresponding sequential algorithm (using one thread) with most parameter settings. For instance, when $k = 2$ and $q = 10$, on Epinions, Slashdot, and WikeVote, the parallel algorithm running with 20 threads is 19.03, 16.72, 17.66 times faster than the sequential algorithm, respectively. These results demonstrate the very high parallel performance of our algorithm.

Exp-6: Efficiency of the parallel algorithms on large real-world graphs. Here we compare the performance of two parallel algorithms: our parallel algorithm and the parallel version of *D2K* algorithm, in processing large real-world graphs. Note that *CPlex* does not support the parallelism, thus we exclude such an algorithm in this experiment. The detailed statistics of the large graphs are

Table 4: Running time of parallel algorithms on large graphs using 20 threads (in seconds).

Datasets	n	m	d	δ	$k=2$				$k=3$			
					q	#k-plexes	*FP*	*D2K*	q	#k-plexes	*FP*	*D2K*
hollywood	2180759	457971264	13107	1297	550	3697275	**49.61**	3209.48	600	57187773	**321.05**	45774.19
enwiki-2021	6261141	300249854	232410	178	50	360	**71.20**	INF	50	40997	**14911.83**	INF
soc-orkut	2997166	212698416	27466	230	40	17607659790	**10468.78**	82290.98	50	6276699468	**17088.40**	INF
fb-A-anon	3097165	47334788	4915	74	10	351549646	**124.41**	257.86	20	594505927	**142.29**	3456.51

shown in Table 4, where columns n, m, d, and δ denote the number of vertices, edges, maximum degree and the degeneracy number of a graph, respectively. Each graph can be downloaded from the Network Repository [25] or https://law.di.unimi.it/datasets.php. Table 4 reports the performance of these two parallel algorithms with 20 threads. We can clearly see that our parallel algorithm substantially outperforms the state-of-the-art parallel algorithm. On some datasets, our parallel algorithm can achieve up to 100 times faster than the parallel version of *D2K* (e.g., on hollywood with $k = 3$ and $q = 600$), which further indicates the superiority of our parallel algorithm in processing large graphs.

6 RELATED WORKS

Maximal clique enumeration. Enumerating all maximal cliques from a graph is a fundamental problem in graph analysis. As shown in [21], the number of maximal cliques could be exponential w.r.t. the graph size, i.e., its number is $O(3^{n/3})$ in the worst case. However, the number of cliques in real-world graphs are often much smaller than such a worst-case bound. Many practical algorithms, including the classic Bron-Kerbosch algorithm [5] and its pivot-based variants [14, 22, 29], work well on real-world graphs. [12, 26] developed parallel pivot-based maximal clique enumeration algorithms to handle large datasets. There also exist many output-sensitive maximal clique enumeration algorithms [6, 10, 20] which can achieve the polynomial-delay time complexity, while those algorithms usually show less efficient than the pivot-based enumeration algorithms.

Maximal relaxed clique enumeration. Since the constraint of clique is often very strictly for real-world community detection related applications, there are also many widely studied relaxed clique models including k-plex [23], s-defective cliques [23], r-clique [3], γ-quasi-clique [18, 24], and so on. To enumerate all these relaxed cliques, many practical algorithms have been proposed in recent years. For example, [3] borrowed the solutions of enumerating all maximal cliques to enumerate all r-cliques. [8] developed a general framework to list all subgraphs with hereditary property which can also be used to enumerate k-plex and s-defective clique, as these two models satisfy the hereditary property. However, such a general framework is inefficient for enumerating a specific subgraph instance (e.g., k-plex or s-defective clique). Liu and Wong [18] presented a so-called quick algorithm to enumerate all γ-quasi-cliques. In this work, we focus on developing efficient, parallel, and scalable solution to enumerate all relatively-large k-plexes. Recently, we notice that a work on the same research question was also published [31] during the review phase of our work. Upon careful examination, we found that these two works are independent of each other, and we proposed a completely different idea to address this problem.

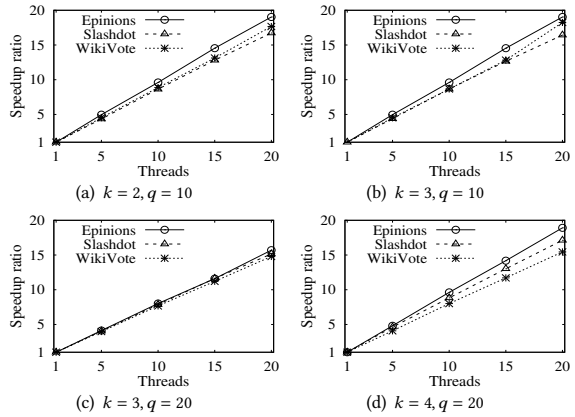

Figure 3: Speedup ratio of our parallel algorithm over the sequential algorithm.

7 CONCLUSION

In this paper, we study the problem of enumerating all maximal k-plexes in a graph with size no less than a given parameter q. To solve this problem, we first develop two efficient upper bounds of the k-plexes containing a set S of vertices. Then, we propose a branch-and-bound algorithm to enumerate maximal k-plexes based on a carefully-designed pivot re-selection technique and the proposed upper-bounding techniques. More importantly, we show that the worst-case time complexity of our algorithm is bounded by $O(n^2\gamma_k^n)$, where γ_k is strictly smaller than 2 (e.g. when setting k equal to 1, 2, and 3, γ_k corresponds to 1.618, 1.839, and 1.928, respectively). In addition, we also devise an effective parallelization strategy for the proposed enumeration algorithm. Extensive experimental results on more than 139 real-world graphs demonstrate the efficiency and scalability of the proposed techniques.

ACKNOWLEDGMENTS

This work was partially supported by (i) National Key Research and Development Program of China 2020AAA0108503, (ii) NSFC Grants 62072034 and U1809206, (iii) CCF-Huawei Populus Grove Fund, (iv) Guangdong Philosophy and Social Sciences Planning Project (GD21CYJ21), (v) The key planning project of education and scientific research of Shenzhen Institute of Education (ZD2021003), and (vi) The China Education Technology Association research project (XJJ202205023). Rong-Hua Li is the corresponding author of this paper.

REFERENCES

[1] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. 2011. Clique Relaxations in Social Network Analysis: The Maximum k-Plex Problem. *Oper. Res.* 59, 1 (2011), 133–142.

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

[3] Rachel Behar and Sara Cohen. 2018. Finding All Maximal Connected s-Cliques in Social Networks. In *EDBT*. 61–72.

[4] Devora Berlowitz, Sara Cohen, and Benny Kimelfeld. 2015. Efficient Enumeration of Maximal k-Plexes. In *SIGMOD*. 431–444.

[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, Jeffrey Xu Yu, and Lu Qin. 2013. Fast Maximal Cliques Enumeration in Sparse Graphs. *Algorithmica* 66, 1 (2013), 173–186.

[7] Peilin Chen, Hai Wan, Shaowei Cai, Jia Li, and Haicheng Chen. 2020. Local Search with Dynamic-Threshold Configuration Checking and Incremental Neighborhood Updating for Maximum k-plex Problem. In *AAAI*. 2343–2350.

[8] 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.

[9] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. 2017. Fast Enumeration of Large k-Plexes. In *KDD*. 115–124.

[10] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. 2016. Sublinear-Space Bounded-Delay Enumeration for Massive Network Analytics: Maximal Cliques. In *ICALP (LIPIcs)*, Vol. 55. 148:1–148:15.

[11] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. 2018. D2K: Scalable Community Detection in Massive Networks via Small-Diameter k-Plexes. In *KDD*. 1272–1281.

[12] Alessio Conte, Roberto De Virgilio, Antonio Maccioni, Maurizio Patrignani, and Riccardo Torlone. 2016. Finding All Maximal Cliques in Very Large Social Networks. In *EDBT*. 173–184.

[13] John C. Doyle, David L. Alderson, Lun Li, Steven Low, Matthew Roughan, Stanislav Shalunov, Reiko Tanaka, and Walter Willinger. 2005. The "robust yet fragile" nature of the Internet. *Proceedings of the National Academy of Sciences* 102, 41 (2005), 14497–14502.

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

[15] Fedor V. Fomin and Dieter Kratsch. 2010. *Exact Exponential Algorithms*.

[16] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. 2018. An Exact Algorithm for Maximum k-Plexes in Massive Graphs. In *IJCAI*. 1449–1455.

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

[18] Guimei Liu and Limsoon Wong. 2008. Effective Pruning Techniques for Mining Quasi-Cliques. In *PKDD*, Vol. 5212. 33–49.

[19] Feng Luo, Bo Li, Xiu-Feng Wan, and Richard H. Scheuermann. 2009. Core and periphery structures in protein interaction networks. *BMC Bioinform.* 10, S-4 (2009).

[20] Kazuhisa Makino and Takeaki Uno. 2004. New Algorithms for Enumerating All Maximal Cliques. In *SWAT*, Vol. 3111. 260–272.

[21] John W. Moon and Leo Moser. 1965. On cliques in graphs. *Israel journal of Mathematics* 3, 1 (1965), 23–28.

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

[23] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2013. On clique relaxation models in network analysis. *Eur. J. Oper. Res.* 226, 1 (2013), 9–18.

[24] Jian Pei, Daxin Jiang, and Aidong Zhang. 2005. On mining cross-graph quasi-cliques. In *KDD*. 228–238.

[25] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. 4292–4293. https://networkrepository.com

[26] Pablo San Segundo, Jorge Artieda, and Darren Strash. 2018. Efficiently enumerating all maximal cliques with bit-parallelism. *Comput. Oper. Res.* 92 (2018), 37–46.

[27] Stephen B. Seidman. 1983. Network structure and minimum degree. *Social Networks* 5, 3 (1983), 269–287.

[28] Stephen B. Seidman and Brian L. Foster. 1978. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology* 6, 1 (1978), 139–154.

[29] 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.

[30] Zhuo Wang, Qun Chen, Boyi Hou, Bo Suo, Zhanhuai Li, Wei Pan, and Zachary G. Ives. 2017. Parallelizing maximal clique and k-plex enumeration over graph data. *J. Parallel Distributed Comput.* 106 (2017), 79–91.

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

[32] 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.

[33] Bin Wu and Xin Pei. 2007. A Parallel Algorithm for Enumerating All the Maximal k -Plexes. In *PAKDD*, Vol. 4819. 476–483.

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

[35] Yi Zhou, Jingwei Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. 2020. Enumerating Maximal k-Plexes with Worst-Case Time Guarantee. In *AAAI*. 2442–2449.