Enumerating Maximal k-Plexes with Worst-Case Time Guarantee

Yi Zhou,[1] Jingwei Xu,[1] Zhenyu Guo,[1] Mingyu Xiao,[1] Yan Jin[2]*
[1]University of Electronic Science and Technology of China,
[2]Huazhong University of Science and Technology
zhou.yi@uestc.edu.cn, {Jingw.Xu, Harry.Guo}@outlook.com, myxiao@gmail.com, jinyan@mail.hust.edu.cn.

Abstract

The problem of enumerating all maximal cliques in a graph is a key primitive in a variety of real-world applications such as community detection and so on. However, in practice, communities are rarely formed as cliques due to data noise. Hence, k-plex, a subgraph in which any vertex is adjacent to all but at most k vertices, is introduced as a relaxation of clique. In this paper, we investigate the problem of enumerating all maximal k-plexes and present *FaPlexen*, an enumeration algorithm which integrates the "pivot" heuristic and new branching schemes. To our best knowledge, for the first time, *FaPlexen* lists all maximal k-plexes with provably worst-case running time $\mathcal{O}(n^2\gamma^n)$ in a graph with n vertices, where $\gamma < 2$. Then, we propose another algorithm *CommuPlex* which non-trivially extends *FaPlexen* to find all maximal k-plexes of prescribed size for community detection in massive real-life networks. We finally carry out experiments on both real and synthetic graphs and demonstrate that our algorithms run much faster than the state-of-the-art algorithms.

Introduction

The analysis of *cohesive groups* also known as *communities* (or *clusters*) has received a lot of attention from researchers of different areas like social and computer science, biology, economics, physics and discrete mathematics. Clique, a subgraph where vertices are pairwise connected, is perhaps the earliest and most studied community model. A large body of literature has emerged in mining cliques with respect to different goals, e.g., finding cliques from general graphs (Xiao and Nagamochi 2017), sparse graphs (Chang, Yu, and Qin 2013; Eppstein and Strash 2011) and uncertain graphs (Mukherjee, Xu, and Tirthapura 2016), fitting the algorithm in main memory (Cheng et al. 2012) and optimizing the running time as a function of the output (Conte et al. 2016).

In real-life applications, due to the existence of data noice, large and closely linked cohesive groups can rarely appear as cliques (Conte et al. 2017; 2018; Balasundaram, Butenko, and Hicks 2011). So other forms of *relaxed clique* are resorted to. For example, the *k-core* (Cheng et al. 2011), *k-club* (Pajouh, Balasundaram, and Hicks 2016) and *quasi-clique* (Veremyev et al. 2016) are such clique-like graphs which relax the vertex degree, pairwise distance of vertices and edge density in induced subgraph, respectively. In this paper, we mainly study the *k-plex*, a relax clique model which has been receiving increasing popularity in recent years (Gao et al. 2018; Conte et al. 2018; Xiao et al. 2017; Conte et al. 2017).

A k-plex (k is a positive integer), first appeared in (Seidman and Foster 1978), is a subgraph such that the degree of each vertex is at least $n - k$, n being the vertex number of the subgraph. Intuitively, the k-plex mimics the clique but each vertex can miss at most $k - 1$ links to the other vertices. A k-plex is said to be *maximal* if it is not a subgraph of any larger k-plex. To effectively use k-plexes in detecting communities, a fundamental problem is to enumerate all maximal k-plexes.

Most existing maximal k-plex enumeration algorithms originate from the celebrated Bron-Kerbosch algorithm by (Bron and Kerbosch 1973), which was initially designed for enumerating all maximal cliques. Wu and Pei (2007) were the first ones of adapting the Bron-Kerbosch algorithm to maximal k-plex ($k \geq 1$) enumeration. Then, Wang et al. (2017) improved the performance by integrating more heuristic pruning rules and using multi-thread parallelization technique. It is also worth mention that Conte et al. (2018) modified the Bron-Kerbosch algorithm for finding communities, i.e., maximal k-plexes with specified connectivity and cardinality constraints and Bentert et al. (2018) extended the algorithm from static graphs to temporal graphs where edges can appear and disappear in different time intervals.

Aside from Bron-Kerbosch families, Berlowitz, Cohen, and Kimelfeld (2015) designed another type of k-plex enumeration algorithm, which optimizes the runtime between the output of two consecutive solutions. Conte et al. (2017) then integrated this algorithm into their framework which reduces larger graphs via efficient k-core and clique computation.

Though we have observed a considerable number of algorithms for enumerating maximal k-plexes, this problem is still intrinsically hard since the numbers of maximal k-plexes can be exponential to the size of a given graph. Indeed, even finding a k-plex of specified size is NP-hard (Balasundaram, Butenko, and Hicks 2011). As demonstrated in this paper, existing Bron-Kerbosch-based algorithms enumerate all the maximal k-plexes in running time $\mathcal{O}(n^2 2^n)$

*Corresponding author.

for an n-vertex graph in the worst-case. To some extent, this result is frustrating as it suggests that the algorithms can potentially degrade to the brute-force enumeration of all subsets of vertices. However, in a specific condition where $k = 1$, Bron-Kerbosch algorithm can list all 1-plexes (cliques) with worst-case running time $O(1.414^n)$ by a simple pivot heuristic (Tomita, Tanaka, and Takahashi 2006). So, we may naturally raise the following questions. Is it possible to improve the worst-case running time of Bron-Kerbosch-based algorithm for $k > 1$? Does there exist the "pivot" techniques which Bron-Kerbosch-based algorithm breaks the trivial exponential bound of 2^n for $k > 1$? We will answer the questions positively in this paper by proposing a new algorithm with specified "pivot" heuristic.

From a practical point of view, it is also of great importance to apply the enumeration algorithms to mining communities in large real-life networks. In the paper, we study the community detection problem defined in (Conte et al. 2018) in which a community is mapped to a maximal k-plex of specified constraints. Since the real-life graphs are often massive, e.g., the *DBLP* co-authorship networks have up to 317,080 nodes and more than 1 million edges (Leskovec and Krevl 2014), we present our techniques for enumerating large communities from networks with millions even billions of vertices in the reminder.

In summary, in this paper, we make the following main contributions.

1. We propose an efficient algorithm called *FaPlexen* for enumerating maximal k-plexes with novel pivot heuristic and branching scheme. Theoretically, in a graph with n vertices, *FaPlexen* enumerates all maximal k-plexes within provably worst-case running time $\mathcal{O}(n^2\gamma_k^n)$ where $\gamma_k < 2$ is a value related to k, e.g., $\gamma_2 = 1.839$. We conduct empirical tests and confirm that for randomly generated graphs and hard DIMACS benchmark graphs, *FaPlexen* runs much faster than other recent best-known algorithms.
2. For community detection purpose, we extend *FaPlexen* to find maximal k-plexes with prescribed size and obtain a refined algorithm, *CommuPlex*. *CommuPlex* enjoys good scalability as it breaks the input graph into small parts and each part is independently enumerated via our proposed algorithm. In massive real-life graphs, *CommuPlex* competes favourably with the *D2K* (Conte et al. 2018), the best-known algorithm dedicated to this problem.
3. All the codes and data are publicly available at https://github.com/aaai20-id9699/faplex.

Notations and problem statements

Let $G = (V, E)$ be a simple and undirected graph, where V and E are the vertex and edge set, respectively. For a vertex set $S \subseteq V$, $G[S]$ is the *subgraph induced by* S. For any vertex v, let $N_G(v)$ be the set of *neighbors* of v, i.e., the set of vertices adjacent to v, $N_G^2(v)$ be the *2-hop neighbors* of v, i.e., the set of neighbors of neighbors of v, except vertex v and vertices of $N_G(v)$ themselves. $|N_G(v)|$ is the *degree* of v in G. Besides, we use $\overline{N}_G(v)$ to denote the *non-neighbors* of v, that is, the set $V \setminus N_G(v)$. Note that $v \in \overline{N}_G(v)$.

The *distance* between two connected vertices u and v in G, which is the shortest length of path between u and v in G, is denoted by $dist_G(u, v)$. The *diameter* of graph G, $diam(G[S])$, is defined as the maximum distance among all the pairs of vertices in G.

A graph G is a *clique* if G is complete. A graph $G = (V, E)$ is a k-plex where k is a positive number, if, for any vertex $v \in V$, $|N_G(v)| \geq |V| - k$. k-plex is a generalization of clique as a 1-plex is a clique.

Given a k-plex (clique) $G[S]$ from G, $G[S]$ is a *maximal k-plex (clique)* in G if there is no any other set S' such that $S \subset S'$ and $G[S']$ is a k-plex (clique). Intuitively, a maximal k-plex is not contained in a larger one. For all the maximal k-plexes in a graph, the ones with largest number of vertices are *maximum k-plexes*.

In this paper, we mainly investigate the maximal k-plex enumeration problem which is formulated as follows.

Problem 1. *Given an undirected graph $G = (V, E)$, a positive integer k, list all the maximal k-plexes induced from G.*

It is known that maximal k-plexes can represent communities in complex networks (Xiao et al. 2017; Balasundaram, Butenko, and Hicks 2011; Pattillo, Youssef, and Butenko 2013). In real-life, a community is a set of cohesively connected members and the size of a community is not trivial, e.g. a community with one of two members is insignificant in applications. In contrast, a maximal k-plex can be a small, disconnected graph, e.g., an independent set of k vertices. To bridge the gap between maximal k-plexes and real communities, Conte et al. (2018) recently recast the community detection problem as the following constrained k-plex enumeration problem.

Problem 2 (Community detection)**.** *Given an undirected graph (network) $G = (V, E)$, detecting all the communities (subset) $S \subseteq V$ such that (1)$G[S]$ is a maximal k-plex and the $|S|$ is not smaller than the threshold q ($q > 0$ is an integer), (2)$G[S]$ is connected and (3) the diameter of $G[S]$ is not larger than 2, i.e., $diam(G[S]) \leq 2$.*

By conclusions of (Xiao et al. 2017), if $G[S]$ is a k-plex and $|S| \geq 2k - 2$, then $G[S]$ must be a connected graph and $diam(G[S]) \leq 2$. Therefore, assume $q \geq 2k - 2$, Problem 2 can be reformulated in a more compact way.

Problem 3. *Given an undirected graph (network) $G = (V, E)$, listing all the maximal k-plexes with at least q vertices where $q \geq 2k - 2$.*

Enumerating all maximal k-plexes

Conventional Algorithms

Existing algorithms for maximal k-plex enumeration such as these in (Conte et al. 2018; Wang et al. 2017; Wu and Pei 2007) stem from the Bron-Kerbosch algorithm which was originally designed from maximal cliques enumeration (Bron and Kerbosch 1973). We first sketch this algorithmic framework before further discussion. Let us name it as *Plexen* as shown in Alg. 1

In general, *Plexen* starts from calling recursive procedure *Plexen-Rec*. *Plexen-Rec* provides three disjoint sets of vertices, P, $Cand$ and $Excl$ as arguments, where P induces a k-plex and $Cand \cup Excl$ are the vertices, each of which can form a k-plex with P. However, $Cand$ are *candidate vertices* that will be considered to be added to P, while $Excl$ are *exclusive vertices* that must be excluded from the k-plex. With these three sets, *Plexen-Rec* lists all maximal k-plexes in the subgraph induced by $P \cup Cand$ such that for each of these maximal k-plexes P',

- $P \subseteq P'$ and,
- $\forall v \in Excl, G[\{v\} \cup P']$ is not a k-plex in G.

Given a graph $G = (V, E)$ and an integer $k > 0$, *Plexen-Rec* is initialized with $P = Excl = \emptyset$ and $Cand = V$. In lines 6-9, if both $Cand$ and $Excl$ become empty, then $G[P]$ is a solution not being enumerated yet. If $Excl$ is not empty, then for each $v \in Excl$, $P \cup \{v\}$ induces a k-plex and thus P is not maximal. In each iteration in lines 10-14, *Plexen-Rec* chooses a vertex u from $Cand$, and calls itself with u moved from $Cand$ to P. The iteration stops when $Cand$ becomes empty. The *update* procedure prunes vertices which cannot form a k-plex with P from $Cand$ and $Excl$.

Algorithm 1: The recursion scheme of conventional k-plex enumeration

```
1  Plexen(G, k)
2  begin
3      Plexen-Rec(G, k, ∅, V, ∅)
4  Plexen-Rec(G, k, P, Cand, Excl)
5  begin
6      if Cand = ∅ then
7          if Excl = ∅ then
8              emit P
9      else
10         for u ∈ Cand do
11             Cand ← Cand \ {u}
12             Cand', Excl' ←
                 update(G, k, P ∪ {u}, Cand, Excl)
13             Plexen-Rec(G, k, P ∪ {u}, Cand', Excl')
14             Excl ← Excl ∪ {u}
15 update(G, k, P, Cand, Excl)
16 begin
17     Cand' ← {v ∈ Cand : P ∪ {v} is a k-plex}
18     Excl' ← {v ∈ Excl : P ∪ {v} is a k-plex}
19     return Cand',Excl'
```

Complexity analysis For complexity analysis in this paper, the following lemma is important.

Lemma 1. *The function* update$(G, k, P, Cand, Excl)$ *runs in time* $\mathcal{O}(|P|^2 + |P|(|Cand| + |Excl|))$.

Proof. We justify the correctness of this lemma by showing that *update* can be implemented by two steps. First, find all vertices u from P such that $|N_G(v) \cap P| = |P| - k$. This step can be finished within running time $\mathcal{O}(|P|^2)$. Second, for each vertex in $v \in Cand \cup Excl$, determine if v can form a k-plex with P. When $G[P]$ is a k-plex, $P \cup \{v\}$ is a k-plex if and only if v satisfies both the two following conditions. (1) v is adjacent to all the vertices u which satisfies $u \in P$ and $|N_P(u)| = |P| - k$. (2) $|N_G(v) \cap P| \geq |P| + 1 - k$. Thus, one can decide if $v \in Cand \cup Excl$ should be pruned in time $O(|P|)$. Finally, the whole running time of *update* is $\mathcal{O}(|P|^2 + |P|(|Cand| + |Excl|))$. □

Now, consider the case where the input graph of *Plexen*, $G = (V, E)$, is a k-plex itself. Due to the hereditary property of k-plex (Pattillo, Youssef, and Butenko 2013), any induced subgraph of G is a k-plex. Thus, for each $P \subseteq V$, *update* will not remove any vertex from $Cand$. With this in mind, one can notice that *Plexen-Rec* will be called for each P equals to a different subset of V, i.e., for a total number of $2^{|V|}$ times. Combining with Lemma 1, the running time of *update* can be assumed as $O(|V|^2)$. Hence, the whole running time of *Plexen* in this case is $\mathcal{O}(|V|^2 2^{|V|})$. It is the worst-case running time complexity of *Plexen*.

In the above case, there is only one maximal k-plex in G, i.e., G itself. It is natural to raise the question that if one can improve the worst-case time complexity of *Plexen* by avoiding the above case. That is to say, when $G[P \cup Cand]$ is a k-plex, stops *Plexen-Rec* and emit $P \cup Cand$ if $G[P \cup Cand]$ is maximal. This is implemented in (Wang et al. 2017). However, so far, there is still no evidence that this algorithm is better than $O(2^{|V|})$ in the worst-case.

A new approach, *FaPlexen*

Suppose $G, k, P, Cand$ and $Excl$ are the input arguments of *Plexen-Rec* in Alg. 1. As we just mentioned, if $G[P \cup Cand]$ is a k-plex, then there is no need to do further recursive search. But what if this is not the case? We have the following observation.

Lemma 2. *If $G[P \cup Cand]$ is not a k-plex, then there exists $u_p \in P \cup Cand$ such that $|N_{G[P \cup Cand]}(u_p)| < |P| + |Cand| - k$, meanwhile, for any maximal k-plex containing P, either u_p is not in the maximal k-plex, or at most $k - (|\overline{N_G}(u_p) \cap P|)$ vertices from $Cand \setminus N_G(u_p)$ is in the maximal k-plex.*

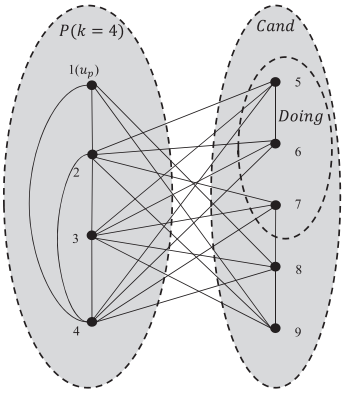
The statement can be justified by the definition of k-plex. It inspires us a *pivot* heuristic to improve *Plexen*. Let us call the improved algorithm *FaPlexen*. *FaPlexen* also starts with calling a recursive procedure, namely *FaPlexen-Rec*, which receives the same input parameters $P, Cand$ and $Excl$ as *Plexen-Rec*. However, the search procedure of *FaPlexen-Rec* is quite different. First, *FaPlexen-Rec* finds a *pivot vertex* $u_p \in P \cup Cand$ which has the minimum degree in $G[P \cup Cand]$, then it continues the search with the following steps. For brevity, let us denote $G' = G[P \cup Cand]$.

- If $|N_{G'}(u_p)| \geq |P| + |Cand| - k$, then G' is a k-plex, we emit G' if it is maximal.
- Otherwise, the search continues with respect to the two disjoint cases: $u_p \in P$ and $u_p \notin P$.

(a) If $u_p \in P$, then we can move at most $k - (|\overline{N_G}(u_p) \cap P|)$ vertices from $Cand \setminus N_G(u_p)$ to P without missing any k-plex by Lemma 2. Let us denote $Doing = Cand \setminus N_G(u_p) = \{v_1, ..., v_d\}$ where d is the size of $Doing$, $p = k - (|\overline{N_G}(u_p) \cap P|)$. Note that $p < k$ by the definition of p and $p \leq d - 1$ since $p = d$ implies that G' is a k-plex. *FaPlexen-Rec* generates $p + 1$ branches, each of which computes P', $Cand'$ and $Excl'$ as follows.

1. In the first branch, $P' \leftarrow P$, $Excl' \leftarrow Excl \cup \{v_1\}$ and $Cand' \leftarrow Cand \setminus \{v_1\}$.
2. For $i \in \{2, ..., p\}$, in the ith branch, $P' \leftarrow P \cup \{v_1, ..., v_{i-1}\}$, $Excl' \leftarrow Excl \cup \{v_i\}$ and $Cand' \leftarrow Cand \setminus \{v_1, ..., v_i\}$.
3. In the last, i.e.$(p+1)$th, branch, $P' \leftarrow P \cup \{v_1, ..., v_p\}$, $Excl' \leftarrow Excl$ and $Cand' \leftarrow Cand \setminus \{v_1, ..., v_d\}$.

Then, in each branch, *FaPlexen-Rec* is recursively called with input argument P',$Cand'$ and $Excl'$. Fig. 1 shows an example of generating branches.



a) The input graph G, arguments $P, Cand$ and $Excl$ for $FaPlexen - Rec$ are shown. The pivot vertex is 1, $k = 4$, so $p = 4 - 2 = 2$, $Doing = \{5,6,7\}$

Vtx	1st	2nd	3rd
1	P	P	P
2	P	P	P
3	P	P	P
4	P	P	P
5	E	P	P
6	C	E	P
7	C	C	X
8	C	C	X
9	C	C	X

b) Each column represents a branch. P, E, C and X indicate that the vertex is in set P', $Excl'$, $Cand'$ and none of them, respectively.

Figure 1: An example of generating branches.

Now, let us justify that the above branches neither miss a maximal k-plex with u_p nor output the same maximal k-plex multiple times.

Suppose S' is a maximal k-plex with $u_p \in S'$ and S' is not visited before. Let us denote $C' = S' \cap Doing$. We first consider the case $C' = Doing$. In this case, S' is emitted in the $(p + 1)$th branch without doubt. Meanwhile, in any other branches, at least one vertex of $Doing$ is moved to $Excl'$, ensuring that S' cannot be emitted in these branches, thus there is no duplication. Then, consider $C' \subset Doing$. Let j be the smallest index such that $v_j \in C'$ while $v_{j+1} \notin C'$. Then, we notice that in the ith branch where $i \leq j$, *Plexen-Rec* puts one vertex of $v_1, ..., v_j$ in $Excl'$, while in the lth branch where $l > j+1$, *Plexen-Rec* puts v_{j+1} in P', both prohibiting the reproduction of S'. Therefore, except the $(j + 1)$th branches, all the other branches do not produce the S^*. Hence, we can make sure that the above branching strategy correct.

(b) If $u_p \in Cand$, we bipartition the search by an invocation of *FaPlexen-Rec* with pivot vertex u_p moved from $Cand$ to P and another invocation of *FaPlexen-Rec* with u_p moved from $Cand$ to $Excl$.

We show the whole algorithm *FaPlexen-Rec* in Alg. 2. In lines 3-5, we check the maximality P like *Plexen-Rec*. After u_p is found in lines 7, *FaPlexen-Rec* checks if $G[P \cup Cand]$ is already a k-plex in line 8-11. If not, as we have explained, *FaPlexen-Rec* generates $p+1$ branches if $u_p \in P$ or 2 branches otherwise.

Algorithm 2: The new recursion scheme of *FaPlexen*

```
1  FaPlexen-Rec(G, k, P, Cand, Excl)
2  begin
3      if Cand = ∅ then
4          if Excl = ∅ then
5              emit P
6      else
7          u_p ← a vertex of minimum degree in
               G' = G[P ∪ Cand]
8          if |N_G'(u_p)| ≥ |P| + |Cand| − k then
9              Cand', Excl' ←
                   update(G, k, P ∪ Cand, ∅, Excl)
10             if Excl' = ∅ then
11                 emit P ∪ Cand
12         else if u_p ∈ P then
13             Doing ← Cand \ N_G(u_p). Suppose
                   Doing = {v_1, ..., v_d} where d is the size
                   of Doing.
14             p ← k − |N̄_G(u_p) ∩ P|
15             for i ← 1, ..., p + 1 do
16                 Build the ith P', Cand' and Excl'
17                 if G[P'] is a k-plex then
18                     Cand', Excl' ←
                           update(G, k, P', Cand', Excl')
19                     FaPlexen-
                           Rec(G, k, P', Cand', Excl')
20         else
21             FaPlexen-
                   Rec(G, k, P, Cand \ {u_p}, Excl ∪ {u_p})
22             Cand', Excl' ← update(G, k, P ∪
                   {u_p}, Cand \ {u_p}, Excl)
23             FaPlexen-
                   Rec(G, k, P ∪ {u_p}, Cand', Excl')
```

Complexity analysis Given a graph $G = (V, E)$ and an integer $k \geq 1$, we evaluate the worst-case running time of *FaPlexen*(G, k) which is equivalent to the worst-case running time of *FaPlexen-Rec*$(G, k, \emptyset, V, \emptyset)$. We begin with the following definitions.

- Let $T(m, n)$ be the worst-case running time of *FaPlexen-Rec*$(G, k, P, Cand, Excl)$ where $m = |Cand|$, $n = |P \cup Cand|$. $(|P| \leq m - n, |Excl| \leq |V| - n)$.

- Consider a nonrecursive procedure *FaPlexen-Rec*$_0(G, k, P, Cand, Excl)$ that is obtained by replacing recursive calls at lines 19, 21 and 23 by *FaPlexen-Rec*$(G, k, P, \emptyset, Excl)$. It is appropriate to assume that the running time of *FaPlexen-Rec*$_0(G, k, P, Cand, Excl)$ is dominated by the *update* operations at lines 9, 18 and 22. Hence we can set the running time of *update* as $c_1|V|^2$ where $c_1 > 0$ is a constant by Lemma 1. Let us assume that $update$ procedure reduces no vertex in the worst-case.

  We now analyze $T(m, n)$ via a case-by-case manner.

1. If $Cand = \emptyset$ (lines 3-5), then
$$T(m, n) = T(0, n) \leq c_2 \tag{1}$$
where c_2 is a constant.

2. If G' is a k-plex, then there is no further branches as shown in lines 8-11. So $T(m, n)$ is dominated by *update* operation, i.e.
$$T(m, n) \leq c_1|V|^2. \tag{2}$$

3. If $u_p \in P$ (lines 12-19), we call $p + 1$ times *FaPlexen-Rec*, each given a smaller $Cand$ set. Specifically, we have
$$\begin{aligned} T(m, n) \leq & \sum_{i=1}^{p} T(m - i, n - 1) \\ & + T(m - d, n - d + p) + c_1|V|^2. \end{aligned} \tag{3}$$

4. If $u_p \notin P$, we have $T(m, n) \leq c_1|V|^2 + T(m - 1, n - 1) + T(m - 1, n)$ due to bipartition search in lines 21-23. However, since no vertex is reduced from $Cand \setminus \{u_p\}$ by *update* in line 22 in the worst-case, we can image that in the next recursive call to *FaPlexen-Rec* in line 22, u_p is still the vertex of minimum degree in G'. So, the recurrence relation can be expanded as
$$\begin{aligned} T(m, n) &\leq c_1|V|^2 + T(m - 1, n - 1) + T(m - 1, n) \\ &\leq c_1|V|^2 + T(m - 1, n - 1) \\ &\quad + \max\{c_2, c_1|V|^2, \sum_{i=1}^{p} T(m - 1 - i, n - 1) \\ &\quad + T(m - d - 1, n - d + p) + c_1|V|^2\} \end{aligned} \tag{4}$$

We sketch how to compute the closed-form upper-bound of $T(m, n)$ which satisfies (in)equalities (1)-(4). First, one can notice that the second item of $T(m, n)$, n, can be dropped from (1)-(4) without loss of generality. Then, we observe that the largest $T(m)$ is upper-bounded by (1) and (4), i.e.

$$T(m) \leq \begin{cases} c_2 & \text{if } m = 0, \\ 2c_1|V|^2 + \sum_{i=0}^{p} T(m - 1 - i) & \\ +T(m - d - 1) & \text{otherwise.} \end{cases} \tag{5}$$

Combining with the fact that $p \leq d - 1$ and $p < k$, it is not hard to verify that when $p = k - 1$ and $d = k$, we can reach the maximum of $T(m)$ via Inequality (5). We leave the detailed procedure of solving $T(m)$ in complementary materials by directly showing that $T(m) \in \mathcal{O}(\alpha_k \gamma_k^m - \beta_k)$ where γ_k is the largest real root of $x^{k+2} - 2x^{k+1} + 1 = 0$, $\beta_k = \frac{2c_2|V|^2}{k}$ and $\alpha_k = max_{i=1}^{k+1}\{\frac{(2^{i-1}k+1)\beta_k + 2^{i-1}c_1}{\gamma_k^i}\}$. As *FaPlexen* calls *FaPlexen-Rec* with $m = |V|$ and k is a constant, we finally have the following result.

Theorem 1. *Given an undirected graph $G = (V, E)$, a positive integer $k \geq 1$,* FaPlexen *enumerates all maximal k-plexes in time $\mathcal{O}(|V|^2\gamma_k^{|V|})$ where γ_k is the largest real root of $x^{k+2} - 2x^{k+1} + 1 = 0$. For example, when $k = 1, 2, 3, 4$ and 5, $\gamma_k = 1.618, 1.839, 1.928, 1.966$ and 1.984, respectively.*

To the best of our knowledge, it is the first algorithm with worst-running time guaranteed to be better than the trivial $2^{|V|}$ bound for enumerating maximal k-plexes when $k > 2$.

Enumeration all maximal k-plexes of prescribed size

We now extend *FaPlexen* to solve Problem 3, i.e., listing all maximal k-plexes with size larger than q, which is equivalent to find communities as defined by Problem 2. A natural idea is to call *FaPlexen* and screen out maximal k-plexes of size smaller than q. However, this approach scales poorly to real-life massive graphs by Theorem 1. Hence, we mitigate this issue via *CommuPlex* as shown in Alg. 3

Algorithm 3: Community detection via k-plex enumeration

```
1 CommuPlex(G, k, q)
2 begin
3     Sort V by degeneracy order as {v_1, ..., v_n}
4     for i ← 1, ..., n and c(v_i) ≥ q − k do
5         Cand ← {v_{i+1}, ..., v_n} ∩ (N_G(v_i) ∪ N_G^2(v_i))
6         Excl ← {v_1, ..., v_{i−1}} ∩ (N_G(v_i) ∪ N_G^2(v_i))
7         Prune vertices from Cand and Excl which
            satisfies Rule 1.
          /* Notice that FaPlexen-Rec
             below does not report k-plex
             smaller than q.                */
8         FaPlexen-Rec(G, k, {v_i}, Cand, Excl)
```

For a graph $G = (V, E)$, the *degeneracy order* is a permutation of vertices $\{v_1, ..., v_n\}$ such that every vertices v_i has the minimum degree in subgraph induced by $\{v_i, ..., v_n\}$. The degree of v_i in $G[\{v_i, ..., v_n\}]$ is called *core number of* v_i, i.e., $c(v_i)$ and the largest core number among all vertices is *degeneracy of* G.

Generally, *CommuPlex* sorts the vertices by degeneracy order, then for each v_i in degeneracy order, *CommuPlex* searches k-plexes with v_i in the subgraph induced by $\{v_i, v_{i+1}, ..., v_n\}$.

Specifically, in line 4, we omit vertices which have core number smaller than $q - k$. Clearly, this does not affect

the correctness of our algorithm by the definition of k-plex. In lines 5 and 6, we restrict $Cand$ and $Excl$ in $N_G(v_i) \cup N_G^2(v_i)$ since the k-plex we are searching is bounded by diameter 2. In line 7, we further prune vertices from $Cand$ and $Excl$ which satisfies the following rule.

Prune Rule 1. *Given a graph $G = (V, E)$, if $v_i \in P$, $G[P]$ is a k-plex and $|P| \geq q$ ($q \geq 2k - 2$), then any other vertex u which satisfies either of the following conditions is not in the P.*

- $u \in N_G(v)$ *and* $|N_G(u) \cap N_G(v_i)| < q - 2k + 2$,
- $u \in N_G^2(v)$ *and* $|N_G(u) \cap N_G(v_i)| < q - 2k$.

The correctness of the above rules can be verified without too much efforts thus we omit the proof.

Given massive sparse real-life graphs $G = (V, E)$, we now analyze the worst-case time complexity of *CommuPlex*. First, the degeneracy order can be efficiently computed in time $\mathcal{O}(|E|)$ (Batagelj and Zaversnik 2003). So the crux is to estimate the worst-case running time of *FaPlexen-Rec* in line 8. Let D denotes the maximum degree of G, Δ denotes the degeneracy of G. By the property of degeneracy order, it is clear $|Cand| \leq D\Delta$ and $|Excl| \leq \Delta^2$ in lines 5 and 6. A further study of Prune Rule 1 discloses that the sizes of $Cand$ and $Excl$ after line 7 are bounded by $\mathcal{O}(\frac{D\Delta}{q-2k+2})$ and $\mathcal{O}(\frac{\Delta^2}{q-2k+2})$, respectively. Now, we can assume that *update* procedure of *FaPlexen-Rec* runs in time $\mathcal{O}(\frac{\Delta^3 D}{(q-2k+2)^2})$ by Lemma 1. Following the procedure of analyzing *FaPlexen-Rec*, we end up by the conclusion that the worst-case time complexity of *CommuPlex* is $\mathcal{O}(|V|\frac{\Delta^3 D}{(q-2k+2)^2}\gamma_k^{\Delta D})$. Let k and q be two constant value, the following statement holds.

Theorem 2. *Given a graph $G = (V, E)$ with maximum degree D and degeneracy Δ,* CommuPlex *finds all maximal k-plexes of size at least q ($q \geq 2k - 2$) (Problem 2 and 3) in time $\mathcal{O}(|V|\Delta^3 D\gamma_k^{\Delta D})$ where γ_k is the largest real root of $x^{k+2} - 2x^{k+1} + 1 = 0$.*

Experiments

In this section, we carry out experiments to evaluate the proposed algorithms.

Experiments setup The codes are written in C++ and compiled by g++ with optimization option '-O3'. All the experiments are conducted on a computer with a CentOS operating system an Intel 3106 CPU (1.7GHz, 8 cores) with 8G memory.

We appreciate the authors of *LP* (Berlowitz, Cohen, and Kimelfeld 2015), *GP* (Wang et al. 2017) and *D2K* (Conte et al. 2018) for publishing their codes. As far as we know, the three solvers are among the most recent and competitive algorithms for enumerating maximal k-plexes. We also revised the codes of *GP* since it misses maximal k-plexes in certain cases. These algorithms are compiled with their makefiles and executed in single-thread mode. Since *GP* and *LP* are algorithms of enumerating maximal k-plexes, we compare *FaPlexen* with *LP* and *GP*. *D2K* is dedicated to find constrained maximal k-plexes defined in Problem 2 in large graphs, therefore, we compare *CommuPlex* with *D2K* in massive real-life graphs. We set the cut off time for each algorithm as 1 day (86400 seconds) for each tested instance.

Table 1: The running time of enumerating maximal k-plexes in random graphs.

Graph	k	#k-plexes	The running time (s)		
			FaPlexen	*GP*	*LP*
$G(100, 0.1)$	2	4449	**0.03**	0.22	53
	3	216556	**0.93**	8.12	15042.94
	4	4061449	**15.95**	229.73	inf
	5	77342779	**367.69**	2388.74	inf
$G(100, 0.2)$	2	13381	**0.06**	1.447	186.08
	3	452145	**2.25**	14.91	41927.63
	4	9241090	**51.07**	153.6	486.61
	5	167704159	**999.73**	2140.98	inf
$G(100, 0.3)$	2	43422	**0.17**	5.478	1442.25
	3	1223082	**6.81**	45.75	inf
	4	30010268	**148.43**	669.15	inf
	5	608724252	**3621.65**	18366.47	inf
$G(100, 0.4)$	2	128151	**0.69**	14.8	9799.18
	3	5054590	**34.25**	194.27	inf
	4	151668073	**1173.20**	7605.75	inf
	5	3681305426	**30328.02**	inf	inf
$G(100, 0.5)$	2	698211	**4.46**	38.47	28426.02
	3	35550150	**312.94**	3248.46	inf
	4	1439934196	**1173.20**	7605.75	inf
$G(100, 0.6)$	2	3537594	**30.57**	381.42	inf
	3	271465956	**3636.09**	36520.96	inf
$G(100, 0.7)$	2	35557201	**456.17**	5799.15	inf
$G(100, 0.8)$	2	1130355448	**32397.67**	inf	inf
$G(500, 0.1)$	2	687111	**8.08**	31.41	82893.97
	3	91709174	**1138.54**	1940.73	inf
$G(500, 0.2)$	2	7587578	**73.15**	381.54	inf
	3	867004746	**9536.80**	49376.75	inf
$G(500, 0.3)$	2	69246850	**830.10**	6561.74	inf
$G(500, 0.4)$	2	1168708597	**21105.59**	inf	inf
CA-GrQc (5242, 28980)	2	13718439	**2788.55**	inf	inf
celegans (453, 2025)	2	104518	**2.09**	726.03	5310.93
	3	16053622	**254.94**	inf	inf
	4	1734552825	**26447.71**	inf	inf
ia-infect-hyper (113, 2196)	2	175887	**1.47**	1754.12	inf
	3	6523528	**61.68**	inf	inf
	4	180196030	**1881.7**	inf	inf
	5	3845997332	**45522.32**	inf	inf
web-edu (3031, 6474)	2	4585512	**408.94**	inf	inf

Enumerating maximal k-plexes We compare *GP, LP* with *FaPlexen* for random and real graphs. In a random graph $G(n, prob)$, there are n vertices and an edge exists between any two vertices with an unified probability $prob \in [0, 1]$. We show the results of random graphs with n ranging in $\{100, 500\}$ and $prob$ varying in $\{0.1, ..., 0.9\}$ in Table 1. In this table, we also demonstrate the results of 5 real graphs which are used in (Wang et al. 2017). We set $k = 2, 3, 4$ and 5. In order to save space, we exclude the very easy instances that are solved by all algorithms in less than 1 seconds and the difficult ones that both algorithms fail to solve within 1 day. Column "#k-plexes" demonstrates the total number of maximal k-plexes in the graph, the label "inf" indicates that the algorithm does not finish in 1 day or the program runs out-of-memory. Clearly, *FaPlexen* is substantially better than other algorithms for almost all

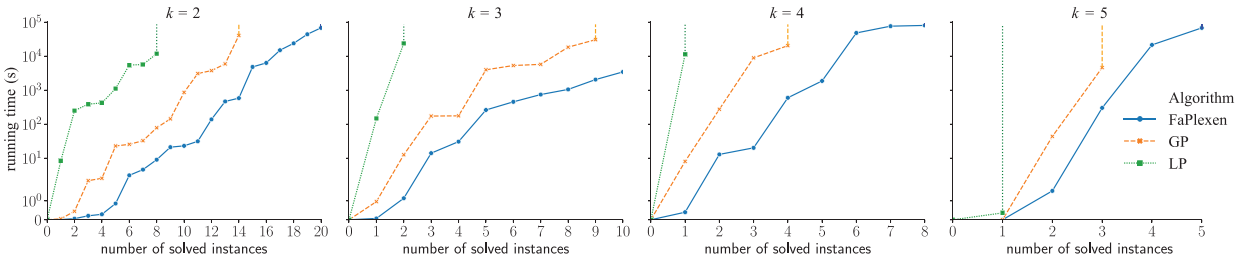


Figure 2: The number of 2^{nd} DIMACS graphs solved by *FaPlexen, GP* and *LP*

Table 2: The running time for community detection.

Graph ($\|V\|, \|E\|$)	k	q	#k-plexes	The running time (s)		Graph ($\|V\|, \|E\|$)	k	q	#k-plexes	The running time (s)	
				CommuPlex	*D2K*					*CommuPlex*	*D2K*
Amazon0505 (410236, 3356824)	2	12	376	3.43	**1.43**	Wiki-Vote (8298, 100761)	2	12	2919931	**121.09**	262.06
		20	0	**0.31**	1.27			20	52	**4.60**	24.71
		30	0	**0.34**	1.32			30	0	1.52	**0.07**
	3	12	6347	17.06	**1.60**		3	12	458153396	**17187.20**	44178.64
		20	0	**0.30**	1.36			20	156727	**331.76**	4365.72
		30	0	**0.35**	1.30			30	0	**1.45**	341.17
	4	12	105649	44.38	**8.40**		4	12	9773156	inf	**11682.45**
		20	0	**0.56**	1.22			20	46729532	**84180.32**	inf
		30	0	**0.32**	1.41			30	0	5.97	**0.20**
Email-EuAll (265214, 420045)	2	12	412779	**9.12**	24.87	soc-pokec -relationships (1632803, 30622564)	2	12	7679906	5949.84	**437.61**
		20	0	2.08	**1.25**			20	94184	1629.28	**46.50**
		30	0	1.05	**0.28**			30	3	543.99	**9.92**
	3	12	32639016	**858.41**	1981.38		3	12	520888893	**17759.73**	33085.17
		20	2637	**10.05**	98.62			20	5911456	1909.81	**1360.47**
		30	0	1.12	**0.263**			30	5	851.52	**14.50**
	4	20	1707177	**833.36**	6008.21		4	20	318035938	**37716.04**	inf
		30	0	1.26	**0.224**			30	4515	1125.52	**225.57**
Slashdot090221 (82144, 500480)	2	12	27208777	**683.57**	743.43	soc-Epinions1 (75879, 508837)	2	12	49823056	**1412.64**	2018.643
		20	11411028	**351.91**	459.81			20	3322167	**165.00**	476.024
		30	453	**14.06**	41.65			30	0	**11.38**	28.131
	3	12	2807943240	79641.40	**76759.21**		3	20	548634119	**28538.47**	75171.243
		20	1303148522	46292.76	**42227.49**			30	16066	**222.50**	5071.633
		30	1679468	**429.96**	5117.93		4	30	13172906	**53793.02**	inf
	4	30	502699966	**77217.32**	inf	jazz (198, 2742)	3	12	93969	**1.23**	1.88
caida (26475, 53381)	3	12	281251	**7.45**	29.54		4	12	2745953	**49.63**	77.33
	4	12	15939883	**448.64**	1788.25	CA-GrQc (5241, 14484)	4	12	128932	**1.00**	1.13

the instances. A speed-up of 2-10 orders of magnitude is often observed. In random graphs, for each k, the speed-up of *FaPlexen* even grows with density when the number of vertices is fixed.

The 2^{nd} DIMACS graphs are well-known difficult benchmarks for clique problems. We then compare the running times of *FaPlexen* with *GP* and *LP* on all 80 2^{nd} DIMACS graphs with k=2, 3, 4 and 5. In Fig. 2, we show the number of solved graphs against the elapsed time. Clearly, *FaPlexen* solves more instances than *GP* and *LP* for all ks. *LP* can only solves 1 to 2 graphs when $k = 3$ and 4, possibly caused by the fact that it does not optimize the worst-case running time. As k grows, the problem becomes harder since all the algorithms can solves fewer instances in the same time frame. We publish the detailed computational results of these graphs along with the codes.

Enumerating size-constrained k-plexes (Community detection) We compare *CommuPlex* with *D2K* for networks in Stanford Large Network Dateset Collection (SNAP) (Leskovec and Krevl 2014) and Laboratory for Web Algorithmics (LAW) [1]. In Table 2, we show the benchmarks which have been used in (Conte et al. 2018) with $k = 2, 3, 4$ and $q = 12, 20, 30$. In order to save space, we exclude the very easy instances that are solved by both algorithms in less than 1 seconds and the difficult ones that both algorithms fail to solve within 1 day.

The results are shown in Table 2 by the same format as Table 1. For the computation of 4-plexes with $q = 30$ for graphs Slashdot090221 and soc-Epinions1, *CommuPlex* was as able to found the all communities while *D2K* not. For graphs like soc-Epinions1, jazz and CA-GrQc, our algorithm outperforms *D2K* for all ks and qs, while for the other graphs, *CommuPlex* competes with *D2K* for different settings of k and q. Generally, the results shows the superiority of *CommuPlex* over *D2K* for more than half of the instances.

Conclusion

We proposed *FaPlexen*, a fast algorithm for enumerating maximal k-plexes. By taking advantages of the framework

[1] http://law.di.unimi.it/

of Bron-Kerbosch algorithm, we developed a new pivot selection and a branching strategy for *FaPlexen*. Based on *FaPlexen* and degeneracy ordering, we further obtained a scalable algorithm which enumerates all maximal k-plexes of a size larger than a given bound, namely, *CommuPlex*. Extensive theoretical and empirical evaluations show that the proposed algorithms can compete with the state-of-the-art algorithms in terms of worst-case time complexity and computation experiments.

Our work not only provides new insights to the fundamental problem of enumerating all maximal k-plexes but also paves the road of the utilization k-plex enumeration algorithm in future graph mining tasks.

Acknowledgements

The work is supported by Natural Science Foundation of China (61802049, 61602196, 61972070) and UESTC (ZYGX2018KYQD210).

References

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

Batagelj, V., and Zaversnik, M. 2003. An o(m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*.

Bentert, M.; Himmel, A.-S.; Molter, H.; Marik, M.; Niedermeier, R.; and Saitenmacher, R. 2018. Listing all maximal k-plexes in temporal graphs. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 41–46. IEEE.

Berlowitz, D.; Cohen, S.; and Kimelfeld, B. 2015. Efficient enumeration of maximal k-plexes. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 431–444. ACM.

Bron, C., and Kerbosch, J. 1973. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM* 16(9):575–577.

Chang, L.; Yu, J. X.; and Qin, L. 2013. Fast maximal cliques enumeration in sparse graphs. *Algorithmica* 66(1):173–186.

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

Cheng, J.; Zhu, L.; Ke, Y.; and Chu, S. 2012. Fast algorithms for maximal clique enumeration with limited memory. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 1240–1248. ACM.

Conte, A.; Grossi, R.; Marino, A.; and Versari, L. 2016. Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, 148:1–148:15.

Conte, A.; Firmani, D.; Mordente, C.; Patrignani, M.; and Torlone, R. 2017. Fast enumeration of large k-plexes. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 115–124. ACM.

Conte, A.; De Matteis, T.; De Sensi, D.; Grossi, R.; Marino, A.; and Versari, L. 2018. D2k: Scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1272–1281. ACM.

Eppstein, D., and Strash, D. 2011. Listing all maximal cliques in large sparse real-world graphs. In *International Symposium on Experimental Algorithms*, 364–375. Springer.

Gao, J.; Chen, J.; Yin, M.; Chen, R.; and Wang, Y. 2018. An exact algorithm for maximum k-plexes in massive graphs. In *IJCAI*, 1449–1455.

Leskovec, J., and Krevl, A. 2014. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

Mukherjee, A. P.; Xu, P.; and Tirthapura, S. 2016. Enumeration of maximal cliques from an uncertain graph. *IEEE Transactions on Knowledge and Data Engineering* 29(3):543–555.

Pajouh, F. M.; Balasundaram, B.; and Hicks, I. V. 2016. On the 2-club polytope of graphs. *Operations Research* 64(6):1466–1481.

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

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

Tomita, E.; Tanaka, A.; and Takahashi, H. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science* 363(1):28–42.

Veremyev, A.; Prokopyev, O. A.; Butenko, S.; and Pasiliao, E. L. 2016. Exact mip-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications* 64(1):177–214.

Wang, Z.; Chen, Q.; Hou, B.; Suo, B.; Li, Z.; Pan, W.; and Ives, Z. G. 2017. Parallelizing maximal clique and k-plex enumeration over graph data. *Journal of Parallel and Distributed Computing* 106:79–91.

Wu, B., and Pei, X. 2007. A parallel algorithm for enumerating all the maximal k-plexes. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 476–483. Springer.

Xiao, M., and Nagamochi, H. 2017. Exact algorithms for maximum independent set. *Information and Computation* 255:126–146.

Xiao, M.; Lin, W.; Dai, Y.; and Zeng, Y. 2017. A fast algorithm to compute maximum k-plexes in social network analysis. In *Thirty-First AAAI Conference on Artificial Intelligence*, 919–925.