A Fast Maximum k-Plex Algorithm Parameterized by the Degeneracy Gap

Zhengren Wang , Yi Zhou * , Chunyu Luo , Mingyu Xiao
University of Electronic Science and Technology of China
zr-wang@outlook.com, zhou.yi@uestc.edu.cn, Chunyu-Luo@std.uestc.edu.cn, myxiao@uestc.edu.cn

Abstract

Given a graph, the k-plex is a vertex set in which each vertex is not adjacent to at most $k-1$ other vertices in the set. The maximum k-plex problem, which asks for the largest k-plex from a given graph, is an important but computationally challenging problem in applications like graph search and community detection. So far, there is a number of empirical algorithms without sufficient theoretical explanations on the efficiency. We try to bridge this gap by defining a novel parameter of the input instance, $g_k(G)$, the gap between the degeneracy bound and the size of maximum k-plex in the given graph, and presenting an exact algorithm parameterized by $g_k(G)$. In other words, we design an algorithm with running time polynomial in the size of input graph and exponential in $g_k(G)$ where k is a constant. Usually, $g_k(G)$ is small and bounded by $O(\log(|V|))$ in real-world graphs, indicating that the algorithm runs in polynomial time. We also carry out massive experiments and show that the algorithm is competitive with the state-of-the-art solvers. Additionally, for large k values such as 15 and 20, our algorithm has superior performance over existing algorithms.

1 Introduction

A *clique* of a graph is a set of vertices that are pairwise connected. The maximum clique problem (MCP), which is to obtain the largest clique from the given graph, is a fundamental NP-hard problem. Applications of MCP include coding theory, computer vision and multi-agent systems [Wu and Hao, 2015; Tošić and Agha, 2004]. However, for many other applications such as complex network analysis, where dense, not necessarily fully connected structures are of particular interest, the clique model is over-restrictive [Pattillo *et al.*, 2012]. Hence, the *k-plex* is proposed as a relaxed form of clique [Seidman and Foster, 1978]. A k-plex is a vertex set that is nearly a clique but each vertex of the k-plex is allowed to have $k-1$ missing adjacent vertices in this vertex set, k being a positive integer. As a basic problem of the k-plex model, the maximum k-plex problem asks for the largest k-plex from the given graph. Algorithms for this problem are also important tools in the analysis of complex networks [Pattillo *et al.*, 2013], especially in the community detection problem [Conte *et al.*, 2018; Zhou *et al.*, 2020; Zhu *et al.*, 2020].

It is known that maximum k-plex problem is NP-hard and W[1]-hard for any fixed $k \geq 1$ [Lewis and Yannakakis, 1980; Balasundaram *et al.*, 2011]. So currently there is almost no hope of finding a polynomial time algorithm, or an algorithm that is parameterized by a given solution size. Although the maximum k-plex problem is computationally challenging, there exist a number of practical exact algorithms in recent years, for instance, BS [Xiao *et al.*, 2017], BnB [Gao *et al.*, 2018], Maplex [Zhou *et al.*, 2021], KpLeX [Jiang *et al.*, 2021], kPlexS [Chang *et al.*, 2022]. These algorithms are notably fast in large real-world graphs, despite with millions of vertices and edges. But for some small dense graphs like these from the 2nd-DIMACS challenge [1], these algorithms can hardly solve them in reasonable time. It is still unclear that why this happens.

Recently, Wang et al. [2022] made some efforts to answer this question from the perspective of parameterized complexity. They showed that, for any fixed k, their maximal k-plex enumeration algorithm, which also can find the maximum k-plex, is *parameterized by* $d(G)$, the degeneracy of the given graph. That is to say, the maximum k-plex problem can be solved with time polynomial in $|V|$ but exponential in $d(G)$. Because empirically $d(G)$ is much smaller than the vertex number, this result can partially demonstrate the superior performance of their algorithm. However, this is not enough because $d(G)$ can still reach hundreds in many graphs, e.g., the degeneracy of soc-livejournal graph is 213 while the gap $g_k(G)$ is only 1 when $k = 2$.

In this paper, we continue pursuing faster algorithm for the maximum k-plex problem that is parameterized by empirically small parameters. We observed that, although the degeneracy is not always small, the degeneracy bound of maximum k-plex, $d(G)+k$, is tight in many real-world graphs. That means, the gap between $d(G)+k$ and the size of maximum k-plex $\omega_k(G)$, is usually small. Indeed, this gap is usually bounded by $O(\log|V|)$. Therefore, it is natural to ask, *"Is it possible to design an algorithm that is parameterized*

*Corresponding author.

[1] http://dimacs.rutgers.edu/archive/Challenges/

by this gap?"

In this paper, we will answer this question positively. Notice that for MCP, the special maximum k-plex problem with $k = 1$, Walteros and Buchanan [2020] designed an algorithm which is parameterized by the gap between $d(G) + 1$ and the maximum clique number. However, due to the structural complexity of k-plexes, their algorithm cannot be simply extended to the maximum k-plex problem with $k \geq 2$. So in general, our main contributions are twofold.

- We propose an algorithm for the maximum k-plex problem with running time $O(|V|^{O(1)}(k+1)^{g_k(G)})$, where $k \geq 1$ is a fixed value and $g_k(G)$ is the gap between the degeneracy bound $d(G) + k$ and the maximum k-plex size. So when $g_k(G) = O(\log(|V|))$ (which is often the case empirically), the algorithm runs in polynomial time. Our main techniques to achieve this result include degeneracy ordering (and decomposition), subset enumeration and a dedicated algorithm for the complementary *d-Bounded-Degree-Deletion* (d-BDD) problem.
- We implement the algorithm and show that it is empirically competitive with the state-of-the-art algorithms for a wide range of instances. In particular, for the group of real-world graphs, the algorithm performs better than existing algorithms with large k values, e.g., $k = 15$ or 20. This is in consistency with the fact that degeneracy gap increases slowly or even decreases as k increases. We further showed that the average branching factor of our algorithm is much smaller than the theoretical estimation.

Codes and supplementary materials are enclosed at https://github.com/joey001/kplex_degen_gap.

2 Preliminaries

2.1 Basic Notations

Let $G = (V, E)$ be a simple undirected graph with vertex set V and edge set E. The complement graph of G is defined as $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(u, v)|u \neq v \wedge (u, v) \notin E\}$. For any vertex v, we use $N_G(v)$ to denote the set of neighbors of v, i.e., the set of vertices adjacent to v, and $N^2_G(v)$ to denote the *2-hop neighbors* of v, i.e., the set of neighbors of vertices in $N_G(v)$ except v and vertices in $N_G(v)$ themselves. For a vertex set $S \subseteq V$, we use $G[S]$ to denote the *subgraph induced* by S, $G \setminus S$ to denote the subgraph induced by $V \setminus S$, and $\Delta_G(S)$ to denote the set of common adjacent vertices of S in G, i.e., $\cap_{u \in S} N_G(u)$.

Let k be a positive integer, a k-plex P is a vertex set such that for any $v \in P$, $|N_G(v) \cap P| \geq |P| - k$. We denote the size of maximum k-plex in G as $\omega_k(G)$. There are two important properties of k-plex. First, any subset of a k-plex is still a k-plex [Trukhanov *et al.*, 2013]. This property indicates that the graph induced by a k-plex belongs to the class of *hereditary graphs*. Second, if P is a k-plex and $|P| \geq 2k - 1$, then $G[P]$ must be a connected graph and the length of the shortest path between any two distinct vertices in $G[P]$ is bounded by 2, while a k-plex with at most $2k - 2$ vertices is probably unconnected [Xiao *et al.*, 2017; Conte *et al.*, 2018]. In many applications, these trivially small, and probably unconnected k-plexes are not of interest. Thus, in the paper, we only investigate the problem of finding the maximum k-plex of size at least $2k - 1$, as in [Wang *et al.*, 2022; Chang *et al.*, 2022]. We define the decision version of the maximum k-plex problem, namely the k-PLEX problem, as follows.

Problem (k-PLEX)**.** *Given a graph $G = (V, E)$, two positive integers k and p ($p \geq 2k - 1$), is there a k-plex of size p in G?*

2.2 The d-Bounded-Degree-Deletion Problem

The k-PLEX problem is closely related to the d-Bounded-Degree-Deletion (d-BDD) problem. A graph G is called *d-degree-bounded* if the maximum degree of G is at most d.

Problem (d-BDD)**.** *Given a graph $G = (V, E)$ and a non-negative integer d, is there a vertex set D of size t such that $G \setminus D$ is d-degree-bounded?*

In a graph $G = (V, E)$, there is a k-plex of size p if and only if there is a $(k - 1)$-bdd of size $|V| - p$ in the complementary graph $\overline{G}$. In this sense, we say that d-BDD is the complementary problem of k-PLEX. However, the parameterized complexities of the two problems are quite different. It is known that d-BDD is *fixed-parameter tractable* (FPT) with respect to parameter t, i.e., there exists an algorithm running in time $O(|V|^{O(1)}f(t))$ where f is a computable function. In contrast, k-PLEX is W[1]-hard with repsect to parameter p. In the literature, Nishimura et al. [2005] presented a $O((d + t)^{t+3}t + n(d + t))$ algorithm for d-BDD, followed by improvements in [Moser *et al.*, 2012; Xiao, 2016]. For $d \geq 3$, the d-BDD problem can be solved in $O(|V|^{O(1)}(d + 1)^t)$ [Xiao, 2016]. However, these algorithms are not practical and only of theoretical interest at the current stage. In our algorithm, we will adopt a simple and easy-to-implement d-BDD algorithm as a subroutine.

We further emphasize that it is not computationally viable to apply these FPT algorithms of d-BDD with $\overline{G}$ directly. In large real-world graphs, the maximum k-plex size is often very small, while the vertex number is quite large. As a result, the parameter $t = |V| - p$ could be rather large, making the above FPT algorithms of d-BDD inefficient in practice. Moser et al. [2012] have tried to solve the maximum k-plex problem in this way but their algorithm is somehow favorable to dense artificial graphs rather than large real-world graphs.

2.3 The Degeneracy and the Degeneracy Gap

The degeneracy. A graph $G = (V, E)$ is called *d-degenerate* if every subgraph has a vertex of degree at most d. The degeneracy of G, denoted $d(G)$, is the smallest number d such that G is d-degenerate. If G has degeneracy $d(G)$, then it has a *degeneracy ordering* which is a permutation of V, $v_1 \dots v_{|V|}$, such that for each vertex v_i, v_i has the minimum degree in the induced subgraph $G[\{v_i, ..., v_{|V|}\}]$. A degeneracy ordering of $G = (V, E)$ can be computed in time $O(|V| + |E|)$ by repeatedly removing a vertex with the minimum degree until the graph becomes empty [Matula and Beck, 1983]. It is known that $d(G) \leq \sqrt{|V| + 2|E|}$ [Eppstein and Strash, 2011]. However, this bound is pessimistic

in large real-world graphs, where $d(G)$ is much smaller than $|V|$. Given a degeneracy ordering $v_1, ..., v_{|V|}$ of $G = (V, E)$, we use $N_G^+(v_i)$ to denote $N_G(v_i) \cap \{v_{i+1}, ..., v_{|V|}\}$ and $N_G^{2+}(v_i)$ to denote $N_G^2(v_i) \cap \{v_{i+1}, ..., v_{|V|}\}$ in the paper.

The degeneracy gap. It is well-known that the degeneracy bound $d(G) + k$ is an upper bound of $\omega_k(G)$ in a graph G [Zhou and Hao, 2017; Conte *et al.*, 2018]. Based on this, we introduce the parameter *k-plex-degeneracy gap* $g_k(G)$ which is equal to $d(G)+k-\omega_k(G)$ for a given graph G. Because the context is always clear, we simply call $g_k(G)$ the *degeneracy gap* in the remaining.

By our observation, the degeneracy gap is often bounded by $O(\log |V|)$. When $k = 1$, the degeneracy gap of some real-world benchmark graphs like web-NotreDame, web-BerkStan and web-Google is only 1 [Walteros and Buchanan, 2020]. For most real-world graphs, the degeneracy gap changes mildly as k increases.

2.4 Related Works

For the maximum k-plex algorithm, the first worst-case running time guarantee is $O(|V|^{O(1)}\alpha_k^{|V|})$, given by Xiao et al. [2017]. When $k = 1, 2, 3$ and 4, $\alpha_k = 1.618, 1.839, 1.928$ and 1.966 respectively. Wang et al. [2022] reduced the exponent from $|V|$ to $d(G)$. Particularly for $k = 1$, i.e., the maximum clique problem (MCP), the best-known running time is $O(|V|^{O(1)}1.1996^{|V|})$ [Xiao and Nagamochi, 2017]. However, this algorithm is not implemented due to the huge gap between theoretical complexity and real performance. Especially, Walteros and Buchanan [2020] designed a practical efficient algorithm with time complexity $O(|V|^{O(1)}1.28^{g_1(G)})$ for MCP, which inspires us to extend the result for the maximum k-plex problem.

There also exist a number of practical efficient methods without running time guarantees. Gao et al. [2018] proposed BnBd and BnBk with several reduction methods and a dynamic vertex selection mechanism in the pre-processing and branch-&-bound procedure. Zhou et al. [2021] proposed Maplex with a second-order reduction and introduced a coloring-based upper bound for pruning. Jiang et al. [2021] proposed KpLeX with a novel partition-based upper bound and corresponding branch rules to avoid unnecessary computation. Chang et al. [2022] proposed kPlexS with an efficient pre-processing based on k-core and k-truss, and a more comprehensive utilization of the second-order reduction. To date, the last two algorithms are believed to be the most competitive ones in practice.

3 An Algorithm Parameterized by the Degeneracy Gap

We present KPLEX in Alg. 1 for solving the k-PLEX problem. Notably, the algorithm is parameterized by g, where $g = d(G) + k - p$. Let us justify its correctness in the followings.

Because the distance between any two vertices in a k-plex P ($|P| \geq 2k - 1$) is at most 2 in $G[P]$, the following observation holds.

Algorithm 1: Our framework for k-PLEX

```
1  KPLEX(G, k, p)
   Input: An input graph G = (V, E), two positive
          integers k and p ≥ 2k − 1.
   Output: A k-plex of size p or 'NO' if there is no such
           set in G.
2  begin
3      Sort V by degeneracy ordering v_1, ..., v_|V|
4      for v_i from v_1 to v_{|V|−p} where |N_G^+(v_i)| ≥ p − k
        do
5          for S ⊆ N_G^{2+}(v_i) and |S| ≤ k − 1 do
6              G_s = (V_s, E_s) ← G[{v_i} ∪ S ∪ N_G^+(v_i)]
7              d ← k − 1, t ← |V_s| − p
8              D* ← DBDD(Ḡ_s, d, t, N_G^+(v_i), ∅)
9              if D* ≠ 'No' then
10                 return V_s \ D*
11     if |N_G^+(v_{|V|−p+1})| ≥ p − k then
12         return {v_{|V|−p+1}, ..., v_|V|}
13     return 'No'
```

Observation 1. *In a graph $G = (V, E)$, let $P \subseteq V$ be an arbitrary k-plex that $|P| \geq 2k - 1$. Denote v_i as the first vertex in P with respect to the degeneracy ordering of G. Then P can be split into three subsets $\{v_i\}$, $N_G^+(v_i) \cap P$ and $N_G^{2+}(v_i) \cap P$, where $|N_G^{2+}(v_i) \cap P| \leq k - 1$.*

By the above observation, we design the algorithm KPLEX using the following idea. For each v_i, we enumerate all subsets $S \subseteq N_G^{2+}(v_i)$ satisfying $|S| \leq k - 1$. For one vertex v_i and one subset S, we decide if there is a k-plex of size p that includes $\{v_i\} \cup S$ in $G[\{v_i\} \cup S \cup N_G^+(v_i)]$. If so, then we find one k-plex of size p.

To be more specific, in lines 11-12, by the property of degeneracy ordering, for vertices from $v_{|V|-p+1}$ to $v_{|V|}$, KPLEX simply decides if $\{v_{|V|-p+1}, ..., v_{|V|}\}$ is a k-plex. For a vertex v_i from v_1 to $v_{|V|-p}$ and a subset $S \subseteq N_G^{2+}(v_i)$, denote $G[\{v_i\} \cup S \cup N_G^+(v_i)]$ as $G_s = (V_s, E_s)$. In lines 6-10, KPLEX decides if there is a k-plex of size p that includes $\{v_i\} \cup S$ in G_s. However, in line 8, instead of solving this decision problem directly, KPLEX calls a subroutine named DBDD to solve the complementary problem – whether there is a $(k-1)$-bdd of size $|V_s| - p$ from $N_G^+(v_i)$ in graph $\overline{G_s}$. The details of such a subroutine is given in Alg. 2.

In terms of the input of DBDD, D is a set of growing vertices, i.e., a set that must be a part of the solution. So D is empty initially. $C \subseteq V$ is the candidate set, i.e. the target d-bdd set, if exists, must be a subset of C. It is clear that DBDD is a tree search algorithm. At each node (or each invocation), DBDD first reduces the input size or decides the solution directly. When the input cannot be solved or reduced anymore, it branches, i.e. DBDD calls itself multiple times with different inputs that cover all possibilities [Cygan *et al.*, 2015]. For example, if a vertex $u \in C$ is selected as a member of D, then u should be removed from G and C and meanwhile t is reduced by 1 in the recursive call; if a vertex $u \in C$ is

Algorithm 2: The d-BDD algorithm

1 *DBDD*(G, d, t, C, D)
Input: A graph $G = (V, E)$, two non-negative integers d and t, a candidate vertex set C, a growing vertex set D that $D \cap V = \emptyset$.
Output: A d-bdd D of size t or 'No' if there is no such vertex set.
2 **begin**
3 **if** $t < 0$ *or* $\exists u \in V \setminus C$ *that* $|N_G(u) \setminus C| > d$ **then**
4 **return** *'No'*
5 **if** $t = 0$ *and* $\forall u \in V, |N_G(u)| \leq d$ **then**
6 **return** D
7 **if** $\exists u \in C$ *that* $N_G(u) > d + t$ **then**
8 **return** *DBDD*$(G \setminus \{u\}, d, t-1, C \setminus \{u\}, D \cup \{u\})$
9 **if** $\exists u \in C$ *that* $\forall v \in \{u\} \cup N_G(u), N_G(v) \leq d$ **then**
10 **return** *DBDD*$(G, d, t, C \setminus \{u\}, D)$
11 Pick a vertex $u_p \in V$ of maximum degree in G.
12 Suppose $N_G(u_p) \cap C$ as $v_1, ..., v_s$ in arbitrary ordering, where $s \leftarrow |N_G(u_p) \cap C|$.
13 **if** $u_p \in C$ **then**
14 $b \leftarrow d + 1 - |N_G(u_p) \setminus C|$
15 (Br. 1) $D_1 \leftarrow$ DBDD$(G \setminus \{u_p\}, d, t-1, C \setminus \{u_p\}, D \cup \{u_p\})$
16 (Br. $i \in \{2, ..., b\}$) $D_i \leftarrow$ DBDD$(G \setminus \{v_{i-1}\}, d, t-1, C \setminus \{u_p, v_1, ..., v_{i-1}\}, D \cup \{v_{i-1}\})$
17 (Br. $(b+1)$) $D_{b+1} \leftarrow$ DBDD$(G \setminus \{v_b, ..., v_s\}, d, t-1-s+b, C \setminus \{u_p, v_1, ..., v_s\}, D \cup \{v_b, ...v_s\})$
18 **else**
19 $b \leftarrow d - |N_G(u_p) \setminus C|$
20 (Br. $i \in \{1, ..., b\}$) $D_i \leftarrow$ DBDD$(G \setminus \{v_i\}, d, t-1, C \setminus \{v_1, ..., v_i\}, D \cup \{v_i\})$
21 (Br. $(b+1)$) $D_{b+1} \leftarrow$ DBDD$(G \setminus \{v_{b+1}, ..., v_s\}, d, t-s+b, C \setminus \{v_1, ..., v_s\}, D \cup \{v_{b+1}, ...v_s\})$
22 **if** $\exists D_i \neq$ *'No'* **then**
23 **return** D_i
24 **return** *'No'*

excluded from being a member of D, u should be removed from C without changing t. Now, we have Lemma 1 holds.

Lemma 1. *Given $G = (V, E)$, a candidate set $C \subseteq V$ and an integer $t \leq |C|$, DBDD(G, d, t, C, D) correctly finds a d-bdd set $D^* \subseteq C$ such that $|D^*| = t$ or return 'No' if no such set exists.*

Proof. Given the input G, d, t, C and a growing set D for DBDD, we have the following reduction rules.

1. If $t < 0$ or $G \setminus C$ is not d-degree-bounded, then there is no d-bdd set of size t in the current input.
2. If $\exists u \in C$ that $N_G(u) > d + t$, then u must be in any d-bdd set of size t.
3. If $\exists u \in C$ that $\forall v \in \{u\} \cup N_G(u), N_G(v) \leq d$, then u must be excluded from some d-bdd set of size t.

The first rule holds straightforwardly and the second and third are from [Moser *et al.*, 2012]. These reduction rules are implemented in lines 3-10 in Alg. 2. When the above reductions cannot be applied any more, the current input is in a state that G is not d-degree-bounded and $t > 0$, so there must exist a vertex $u_p \in V$ that $|N_G(u_p)| > d$. By definition, if there is a solution in the current input, then either u_p is in the solution or at least $|N_G(u_p)| - d$ vertices from $N_G(u_p)$ are in the solution. Assume $D^* \subseteq C$ is a solution with $|D^*| = t$ in the current input. For illustration purpose, denote $N_G(u_p) \cap C = \{v_1, ..., v_s\}$ in arbitrary ordering, s being the size of $N_G(u_p) \cap C$. It is easy to check that the following cases are disjoint and complete.

- $u_p \in C$. There are $b+1$ possibilities where $b = d+1-|N_G(u_p) \setminus C|$.
  1. The first possibility, $u_p \in D^*$.
  2. The i-th possibility where $i \in \{2, ..., b\}$, $u_p, v_1..., v_{i-2} \notin D^*$ and $v_{i-1} \in D^*$. (In case $i = 2$, $u_p \notin D^*$ and $v_1 \in D^*$.)
  3. The b+1-th possibility $u_p, v_1, ..., v_{b-1} \notin D^*$ and $v_b, ..., v_s \in D^*$.
- $u_p \notin C$. There are $b+1$ possibilities where $b = d - |N_G(u_p) \setminus C|$.
  1. The i-th possibility where $i \in \{1, ..., b\}$, $v_1, ..., v_{i-1} \notin D^*$ and $v_i \in D^*$. (In case $i = 1$, $v_1 \in D^*$.)
  2. The b+1-th possibility, $v_1, ..., v_b \notin D^*$ and $v_{b+1}, ..., v_s \in D^*$.

Since DBDD exactly covers all the above cases (lines 11-23), we conclude that DBDD is a complete algorithm. □

3.1 Running Time Analysis

The running time of DBDD. As we mentioned, DBDD is a tree search algorithm. We can measure its running time by the number of tree nodes multiplying the time taken at each node itself. Suppose the input of the our algorithm is DBDD$(G = (V, E), d, t, C, D)$. We can safely assume that the time taken at each tree node is $O(|V|^{O(1)})$. (In our implementation, the time is bounded by $O(|V|^2)$). By the branching rule, the parameter t decreases at least 1 at each sub-node. Denote $L(t)$ as the number of leaf nodes in the subtree. We have

$$L(t) \leq \underbrace{L(t-1) + ... + L(t-1)}_{b \text{ times}} + L(t - N_G(u_p) + d),$$

where $b = d + 1 - |N_G(u_p) \setminus C|$ when $u_p \in C$ and $b = d - |N_G(u_p) \setminus C|$ when $u_p \notin C$, and $L(1) = 1$. In the worst case, $u_p \in C$, $N_G(u_p) \setminus C = \emptyset$ and $N_G(u_p) = d + 1$, that is to say, $b = d + 1$ and $N_G(u_p) - d = 1$. So, we obtain $L(t) \leq (d+2)^t$ and the number of all tree nodes is $O((d+2)^t)$. Combining the running time of each node, we conclude the running time $T_{DBDD}(G, d, t, C, D)$ is bounded by $O(|V|^2(d+2)^t)$.

The running time of KPLEX. Given $KPLEX(G = (V, E), k, p)$, the running time of KPLEX $T_{KPLEX}(G, k, p)$ can be calculated by

$$T_{KPLEX}(G,k,p) = T_{deg}(G) + \sum_{i=1}^{|V|-p} \sum_{\substack{S \subseteq N_G^{2+}(v_i) \\ |S| \le k-1}} \Big(T_{graph}(G_s) + T_{DBDD}(\overline{G_s}, k-1, t, N_G^+(v_i), \emptyset) \Big),$$

where $T_{deg}(G)$ is the time of degeneracy ordering, $T_{graph}(G_s)$ is the time of building G_s given a vertex v_i and a subset S, and $T_{DBDD}(\overline{G_s}, k-1, t, N_G^+(v_i), \emptyset)$ is the time of running DBDD$(\overline{G_s}, k-1, t, N_G^+(v_i), \emptyset)$. Note that $G_s = (V_s, E_s)$ is the induced subgraph $G[\{v_i\} \cup S \cup N_G^+(v_i)]$ and $t = |V_s| - p$.

It is known that $T_{deg}(G) \le O(|V| + |E|)$ by [Batagelj and Zaversnik, 2003]. Define $g = d(G) + k - p$. Because $|N_G^+(v_i)| \le d(G)$ and $|S| + 1 \le k$, we have $|V_s| \le d(G) + |S| + 1 \le d(G) + k$ and $t \le d(G) + k - p = g$. Therefore $T_{graph}(G_s) \le (d(G) + k)^2$ and $T_{DBDD}(\overline{G_s}, k - 1, t, N_G^+(v_i), \emptyset) \le (d(G) + k)^2(k + 1)^g$. (The first inequality is obtained by using adjacency matrix to build a graph and the second inequality is obtained by the above analysis of DBDD). Therefore,

$$\begin{aligned} T_{KPLEX}(G,k,p) &= O\Bigg(|V| + |E| + \sum_{i=1}^{|V|-p} \sum_{\substack{S \subseteq N_G^{2+}(v_i) \\ |S| \le k-1}} \Big((d(G)+k)^2 + (d(G)+k)^2 (k+1)^g \Big) \Bigg) \\ &= O\Bigg(|V| + |E| + \sum_{i=1}^{|V|-p} |V|^{k-1} (d(G)+k)^2 (k+1)^g \Bigg) \\ &= O(|V| + |E| + |V|^k (d(G)+k)^2 (k+1)^g). \end{aligned}$$

In summary, we obtained the running time for KPLEX as follows.

Theorem 1. *Given a graph $G = (V, E)$, a fixed integer $k \ge 1$ and an integer p that $p \ge 2k - 1$, KPLEX(G, k, p) solves the k-PLEX problem in time $O(|V|^{O(1)}(k + 1)^g)$, where $g = d(G) + k - p$.*

It is easy to use KPLEX for solving the maximum k-plex problem. For each integer p from $d(G) + k$ to $2k - 1$, we call KPLEX(G, k, p) to decide if there exists a k-plex of size p in G. If so, then we stop and conclude that $\omega_k(G)$ is equal to p. If no for any p value, then we conclude that $\omega_k(G) < 2k - 1$, i.e. the maximum k-plex is trivially small. To sum up, we run KPLEX at most $|V|$ times, and at each time, $g \le d(G) + k - \omega_k(G)$. Thus, we have the time complexity for the maximum k-plex problem as follows.

Theorem 2. *Given a graph $G = (V, E)$ and a fixed integer $k \ge 1$, the maximum k-plex problem can be solved in time $O(|V|^{O(1)}(k+1)^{g_k(G)})$, where $g_k(G) = d(G) + k - \omega_k(G)$.*

Remarks We observed that in real-world graphs, $g_k(G)$ is often small. If we assume $g_k(G)$ is bounded by $O(\log |V|))$ (which is often the case), we can solve the maximum k-plex problem in polynomial time. Moreover, we observe that the branching factor of our tree search, $k + 1$, is very pessimistic in practice. For example, for the consph graph with $k = 20$, the average branching factor can be as small as 1.84 in the experiments.

4 An Efficient Implementation

In this section, we introduce some important techniques for an efficient implementation of KPLEX.

4.1 Graph Reduction

In KPLEX, we need to build G_s for each v_i and $S \subseteq N_G^{2+}(v_i)$. Here, we introduce some reduction rules to reduce the size of G_s or even identify that the computation of G_s is unnecessary.

Reduction 1 (First- and Second-order Reduction [Zhou *et al.*, 2021])**.** *Given a graph $G = (V, E)$ and a vertex $u \in V$, if $|N_G(u)| < p - k$, then u is not in any k-plex of size p. Furthermore, for any two distinct vertices u and v of G,*

1. *if $(u, v) \in E$ and $|\Delta_G(\{u, v\})| < p - 2k$, then u and v are not in any k-plex of size p at the same time.*
2. *if $(u, v) \notin E$ and $|\Delta_G(\{u, v\})| < p - 2k + 2$, then u and v are not in any k-plex of size p at the same time.*

The first-order reduction is a default configuration in almost all algorithms. The second-order reduction recently appeared in [Zhou *et al.*, 2021; Jiang *et al.*, 2021; Chang *et al.*, 2022]. We generalize them to the following rule, namely the *higher-order reduction* rule.

Reduction 2 (Higher-order Reduction)**.** *Given a graph $G = (V, E)$ and an arbitrary vertex set $P \subseteq V$, denoting $n = |P|$ and $\lambda = |E(G[P])|$, if $|\Delta_G(P)| < p - nk + n(n - 1) - 2\lambda$, then P is not a subset of any k-plex of size p.*

The higher-order reduction can help avoid unnecessary computation in KPLEX. For each vertex v_i and subset S, let $P = \{v_i\} \cup S$, $n = |S| + 1$ and $\lambda = |E(G_s[\{v_i\} \cup S])|$. With the high-order reduction, we can save the invocation of DBDD when $|\Delta_{G_s}(\{v_i\} \cup S)| < p - nk + n(n-1) - 2\lambda$. If the invocation to DBDD is unavoidable, the size of G_s can still be reduced by this rule. We leave the details of this reduction algorithm in the appendix.

4.2 Branch-&-Bound

We implement KPLEX in the branch-&-bound [Jiang *et al.*, 2021; Chang *et al.*, 2022] framework for solving the maximum k-plex algorithm. (This is somehow different to our theoretical analysis that maximum k-plex can be solved by multiple invocations of KPLEX). The branch-&-bound maintains a lower bound p and branches to either prove that p is the optimal size or update p when a better solution is found.

Initial lower bound. The initial lower bound is computed by an extension of the degeneracy ordering. As a degeneracy ordering can be computed by repeatedly removing a vertex with minimum degree. So, at some time, the set of remaining vertices becomes a k-plex. We take this remaining set as a lower bound solution.

Bounding procedure. Bounding is used to early certify that some tree nodes are unfruitful. That is, given an input instance DBDD(G, d, t, C, D), we estimate a lower bound of the size of the d-bdd before branching. If this bound is larger than t, we can drop the current tree node without further branching. To find such a tight lower bound, we use the most recent bounding technique in [Jiang *et al.*, 2021].

5 Experiments

In this section, we evaluate our algorithm empirically. Our algorithm is written in C++11 and compiled by G++ version 9.3.0 with -Ofast flag. All experiments are conducted on a machine with an Intel(R) Xeon(R) Gold 6130 CPU @ 2.1GHz and a Ubuntu 22.04 operating system. Hyper-threading and turbo techniques are disabled for steady clock frequency.

We mainly compare our algorithm with two recent algorithms, KpLeX [Jiang *et al.*, 2021] [2] and kPlexS [Chang *et al.*, 2022] [3]. To the best of our knowledge, KpLeX and KPlexS are state-of-the-art algorithms and dominate earlier algorithms in experiments. Note that, we adapt KpLeX such that it searches for a maximum k-plex of size at least $2k - 1$.

As far as we know, the existing algorithms are only tested with k values at most 7. However, the performance with even larger k values should also be an important metric for the maximum k-plex algorithms. Therefore, we carry out experiments with $k = 2, 5, 10, 15, 20$ and time limit 1800 seconds.

5.1 Overall Performance

Real-World Graphs. We evaluate the algorithms with two sets of real-world graphs.

- **Network-Repo Graphs.** This dataset contains 139 real-world graphs with up to 5.87×10^7 vertices from the Network Data Repository [4], including social networks, biological networks, collaboration networks and so on.
- **10th-DIMACS Graphs.** This dataset contains 84 graphs with up to 5.09×10^7 vertices [5], most of them are real-world graphs.

These graphs have also been used in the literature [Gao *et al.*, 2018; Zhou *et al.*, 2021; Jiang *et al.*, 2021; Chang *et al.*, 2022]. Note that many graphs can be solved by kPlexS and our algorithm in 10 seconds for all these k values while some graphs cannot be solved by any solver for any k in the time limit. For convenience of comparison, these extremely easy or hard graphs are removed so that only 27 in Network-Repo, and 16 in 10th-DIMACS graphs are left for presentation. In Fig. 1, we present the number of solved instances within different time frames for $k = 2, 10, 15, 20$ for these real-world graphs and the following artificial graphs as well. The $k = 5$ case is left in the appendix for space reason.

As shown in Fig. 1, in general, our algorithm is competitive with other algorithms for all k values in real-world graphs. Specifically, when $k = 2$, our algorithm is on par with kPlexS, but gradually outperforms it when k becomes larger. When k is 15 and 20, the superiority of our algorithm is evident. For example, our algorithm solves 25 Network-Repo graphs when $k = 15$ in 30 seconds, while kPlexS solves 17 and KpLeX only solves 1. For 10th-DIMACS graphs, our algorithm remarkably solves 15 over all the 16 graphs in 300 seconds for any k.

Artificial Dense Graphs. We also test the algorithms in the traditional clique graphs, i.e., the graphs from 2nd-DIMACS challenge [6].

- **2nd-DIMACS Graphs** This dataset contains 80 graphs with up to 4.00×10^3 vertices. Because a large body of the set are artificial dense graphs, this set is often hard to be solved [Jiang *et al.*, 2021].

As shown in Fig. 1, the situation is quite different from real-world graphs. When $k = 2$, KpLeX performs better than both kPlexS and our algorithm. And when $k = 5$, these algorithms compete each other. However, when k becomes 15 and 20, our algorithm clearly outperforms others.

We can roughly conclude that our algorithm is scalable to large real-world graphs and large k values, as well as kPlexS, while KpLeX is suitable for artificial dense graphs with relatively small k values.

5.2 Analysis on Key Components

Analysis on Degeneracy Gap. We investigate the degeneracy gap parameter $g_k(G) = d(G) + k - \omega_k(G)$ in this part.

In Table 1, we present information on six representative graphs, covering all the aforementioned datasets. We notice that $d(G)$, the degeneracy of the given graph, is often above 50 and can be as large as 213 in soc-livejournal. In contrast, $g_k(G)$ is much smaller than 50, e.g. $g_k(G) = 1$ when $k = 2$ in soc-livejournal. A scatter graph in the appendix demonstrates that, $g_k(G)$ is often much smaller than $d(G)$ in most real-world graphs. It is also interesting to notice that $g_k(G)$ might even decrease as k increases, as it happens in the graph consph when k increases from 10 to 15. Our statistic result reveals that $g_k(G)$ is likely to be bounded by $O(\log(|V|))$ in many real-world graphs.

On the flip side, the degeneracy gaps on artificial dense graphs are relatively large. As shown in Table 1, in C125.9, $g_k(G) = 62, 44$ and 21 when $k = 2, 5, 10$, respectively. As a result, our algorithm cannot solve it in reasonable time. When $k = 15$ and 20, $g_k(G)$ of C125.9 is 5 and 0, respectively, and our algorithm can easily solve it in 66 and 0 seconds, respectively. In general, there is a positive correlation between the practical efficiency of our algorithm and the degeneracy gap. Nevertheless, we emphasize that the gap parameter only partially explains the practical performance of our algorithm.

Analysis on Branching Factor. As the branching factor of our DBDD subroutine is $k+1$ in the worst-case, our algorithm has exponential complexity with base factor $k + 1$. However, experiments show that this estimation is pessimistic. In Table 1, the average branching factor γ is also presented. In general, the average branching factor increases slowly as k

[2] https://github.com/huajiang-ynu/kplex

[3] https://lijunchang.github.io/Maximum-kPlex

[4] http://lcs.ios.ac.cn/~caisw/Resource/realworld%20graphs.tar.gz

[5] https://networkrepository.com/dimacs10.php

[6] https://networkrepository.com/dimacs.php

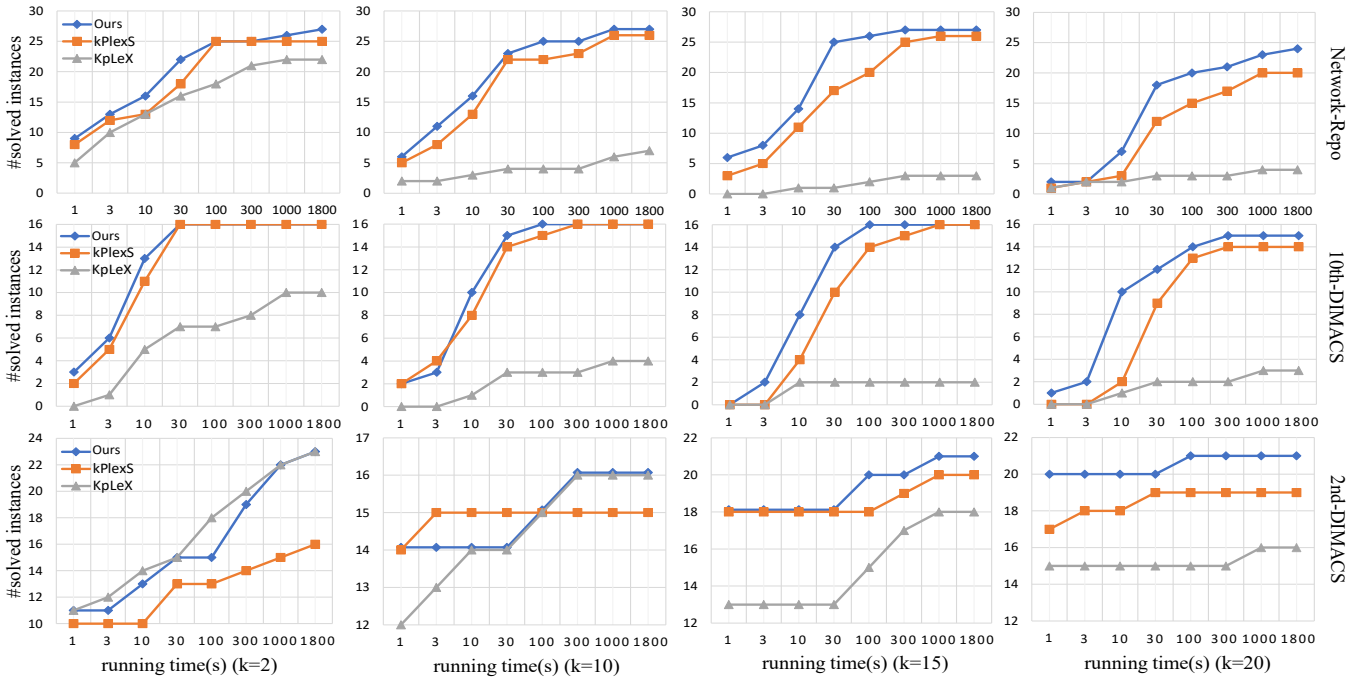

Figure 1: Number of solved instances for Network-Repo, 10th-DIMACS and 2nd-DIMACS graphs, with $k = 2, 10, 15, 20$ and time limit 1800 seconds.

increases. Indeed, it is always much smaller than the $k + 1$. We observe that when $k = 2$, the branching factor of consph, tech-as-skitter and soc-livejournal is 1, meaning that an optimal solution can be found without branching. Even for C125.9, the branching factor 6.29 is still much smaller than the worst-case value 16, given $k = 15$. We notice that the reduction rules could possibly influence the average branching factor. For instance, when $k = 20$, the branching factor on soc-slashdot graph is 4.12 with reduction rules while it becomes 7.64 without them.

6 Conclusion

In this paper, we gave rigorous explanation on why exponential-time maximum k-plex algorithm is efficient in practice. Specifically, we presented an algorithm that is parameterized by degeneracy gap $g_k(G)$, a parameter which is the difference between the degeneracy bound of graph G and the size of maximum k-plex in G. We demonstrated that in many real-world graphs, the degeneracy gap is very small, bounded by $O(\log(|V|))$. With experiments, we further showed that the algorithm is efficient with real-world graphs which have small degeneracy gaps. Remarkably, the algorithm outperforms the state-of-the-art algorithms when k is large. Our work not only investigates the complexity of maximum k-plex problem from a new viewpoint, but also provides novel insights for other graph models, like k-bundle [Zhou *et al.*, 2022; Hu *et al.*, 2023], k-defective clique [Chen *et al.*, 2021; Gao *et al.*, 2022] and so on.

Graph	k	$\omega_k(G)$	$g_k(G)$	γ_b	γ	T_b	T	T_o
soc-slashdot $\|V\| = 70068$ $\|E\| = 358647$ $d(G) = 53$	2	31	24	1.28	1.34	0.27	0.73	1.59
	5	40	18	1.43	1.88	42.18	0.19	1.21
	10	51	12	2.31	1.39	335.61	0.06	0.13
	15	59	9	OOT	2.66	OOT	1.02	9.47
	20	68	5	7.64	4.12	4.76	0.27	0.11
consph $\|V\| = 79679$ $\|E\| = 2963573$ $d(G) = 41$	2	24	19	1.00	1.00	1.77	1.71	1.75
	5	26	20	1.76	1.89	82.84	25.14	56.14
	10	33	18	2.92	1.25	101.05	9.83	19.27
	15	42	14	OOT	1.45	OOT	16.77	437.79
	20	45	16	OOT	1.84	OOT	268.64	OOT
sc-pwtk $\|V\| = 217891$ $\|E\| = 5653221$ $d(G) = 35$	2	24	13	2.96	2.00	2.27	2.17	2.31
	5	26	14	2.19	1.99	4.34	3.50	4.38
	10	33	12	1.79	2.00	15.57	10.51	10.85
	15	38	12	OOT	1.64	OOT	18.61	116.60
	20	46	9	2.13	1.97	1151.39	6.61	122.78
tech-as-skitter $\|V\| = 1694616$ $\|E\| = 11094209$ $d(G) = 111$	2	69	44	1.00	1.00	1.02	1.03	1.42
	5	75	41	1.78	2.31	0.87	0.87	0.90
	10	84	37	OOT	2.95	OOT	1.04	4.35
	15	95	31	OOT	1.94	OOT	0.91	1.38
	20	104	27	OOT	3.42	OOT	1.14	1.82
soc-livejournal $\|V\| = 4033137$ $\|E\| = 27933062$ $d(G) = 213$	2	214	1	1.00	1.00	1.53	1.45	3.63
	5	214	4	1.00	1.00	2.17	1.86	2.35
	10	217	6	5.29	5.29	2.21	2.87	3.52
	15	221	7	1.00	1.00	3.47	3.46	3.46
	20	222	11	9.57	1.89	3.62	3.67	3.60
C125.9 $\|V\| = 125$ $\|E\| = 6963$ $d(G) = 102$	2	42	62	OOT	OOT	OOT	OOT	OOT
	5	63	44	OOT	OOT	OOT	OOT	OOT
	10	91	21	OOT	OOT	OOT	OOT	OOT
	15	112	5	OOT	6.29	OOT	66.22	158.44
	20	122	0	1.00	1.00	0.00	0.00	0.00

Table 1: Properties of six representative graphs and their running behavior. γ is the average branching factor, T is the running time, γ_b and T_b represent the branching factor and running time respectively that without second- and higher-oder reduction, and T_o is the shorter running time between KpLeX and kPlexS.

Acknowledgments

This work was supported by CCF-Huawei Huyanglin Fund - Theoretical Computer Science Special Project, China Post-

doctoral Science Foundation under grant 2022M722815, Natural Science Foundation of Sichuan Province of China under grants 2023NSFSC1415 and 2023NSFSC0059, and National Natural Science Foundation of China under grant 61972070.

References

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

[Batagelj and Zaversnik, 2003] Vladimir Batagelj and Matjaz Zaversnik. An o(m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[Chang *et al.*, 2022] Lijun Chang, Mouyi Xu, and Darren Strash. Efficient maximum k-plex computation over large sparse graphs. *Proceedings of the VLDB Endowment*, 16(2):127–139, 2022.

[Chen *et al.*, 2021] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. Computing maximum k-defective cliques in massive graphs. *Computers & Operations Research*, 127:105131, 2021.

[Conte *et al.*, 2018] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. 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*, pages 1272–1281. ACM, 2018.

[Cygan *et al.*, 2015] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

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

[Gao *et al.*, 2018] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. An exact algorithm for maximum k-plexes in massive graphs. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 1449–1455, 2018.

[Gao *et al.*, 2022] Jian Gao, Zhenghang Xu, Ruizhi Li, and Minghao Yin. An exact algorithm with new upper bounds for the maximum k-defective clique problem in massive sparse graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10174–10183, 2022.

[Hu *et al.*, 2023] Shan Hu, Yi Zhou, Mingyu Xiao, Zhang-Hua Fu, and Zhipeng Lü. Listing maximal k-relaxed-vertex connected components from large graphs. *Information Sciences*, 620:67–83, 2023.

[Jiang *et al.*, 2021] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. A new upper bound based on vertex partitioning for the maximum k-plex problem. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 1689–1696, 2021.

[Lewis and Yannakakis, 1980] John M Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

[Matula and Beck, 1983] David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983.

[Moser *et al.*, 2012] Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *Journal of Combinatorial Optimization*, 24(3):347–373, 2012.

[Nishimura *et al.*, 2005] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. *Discrete Applied Mathematics*, 152(1-3):229–245, 2005.

[Pattillo *et al.*, 2012] Jeffrey Pattillo, Nataly Youssef, Sergiy Butenko, et al. Clique relaxation models in social network analysis. *Springer Optimization and Its Applications*, pages 143–162, 2012.

[Pattillo *et al.*, 2013] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.

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

[Tošić and Agha, 2004] Predrag T Tošić and Gul A Agha. Maximal clique based distributed coalition formation for task allocation in large-scale multi-agent systems. In *International Workshop on Massively Multiagent Systems*, pages 104–120. Springer, 2004.

[Trukhanov *et al.*, 2013] Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications*, 56(1):113–130, 2013. Number: 1.

[Walteros and Buchanan, 2020] Jose L Walteros and Austin Buchanan. Why is maximum clique often easy in practice? *Operations Research*, 68(6):1866–1895, 2020.

[Wang *et al.*, 2022] Zhengren Wang, Yi Zhou, Mingyu Xiao, and Bakhadyr Khoussainov. Listing maximal k-plexes in large real-world graphs. In *Proceedings of the ACM Web Conference 2022*, pages 1517–1527, 2022.

[Wu and Hao, 2015] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.

[Xiao and Nagamochi, 2017] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017.

[Xiao *et al.*, 2017] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. A fast algorithm to compute maximum k-plexes in social network analysis. In *Proceedings of the AAAI conference on Artificial Intelligence*, pages 919–925, 2017.

[Xiao, 2016] Mingyu Xiao. A parameterized algorithm for bounded-degree vertex deletion. In Thang N. Dinh and My T. Thai, editors, *Computing and Combinatorics - 22nd International Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, 2016, Proceedings*, volume 9797 of *Lecture Notes in Computer Science*, pages 79–91. Springer, 2016.

[Zhou and Hao, 2017] Yi Zhou and Jin-Kao Hao. Frequency-driven tabu search for the maximum s-plex problem. *Computers & Operations Research*, 86:65–78, 2017.

[Zhou *et al.*, 2020] Yi Zhou, Jingwei Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. Enumerating maximal k-plexes with worst-case time guarantee. In *Proceedings of the AAAI conference on Artificial Intelligence*, pages 2442–2449, 2020.

[Zhou *et al.*, 2021] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. Improving maximum k-plex solver via second-order reduction and graph color bounding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12453–12460, 2021.

[Zhou *et al.*, 2022] Yi Zhou, Weibo Lin, Jin-Kao Hao, Mingyu Xiao, and Yan Jin. An effective branch-and-bound algorithm for the maximum s-bundle problem. *European Journal of Operational Research*, 297(1):27–39, 2022.

[Zhu *et al.*, 2020] Jinrong Zhu, Bilian Chen, and Yifeng Zeng. Community detection based on modularity and k-plexes. *Information Sciences*, 513:127–142, 2020.