A Refined Upper Bound and Inprocessing for the Maximum K-plex Problem

Hua Jiang[*] , **Fusheng Xu** , **Zhifei Zheng** , **Bowen Wang** , **Wei Zhou**
Engineering Research Center of Cyberspace & School of Software, Yunnan University, China
huajiang@ynu.edu.cn,{xfs, zzfreya, morningnews}@mail.ynu.edu.cn, zwei@ynu.edu.cn

Abstract

A k-plex of a graph G is an induced subgraph in which every vertex has at most $k-1$ nonadjacent vertices. The Maximum k-plex Problem (MKP) consists in finding a k-plex of the largest size, which is NP-hard and finds many applications. Existing exact algorithms mainly implement a branch-and-bound approach and improve performance by integrating effective upper bounds and graph reduction rules. In this paper, we propose a refined upper bound, which can derive a tighter upper bound than existing methods, and an inprocessing strategy, which performs graph reduction incrementally. We implement a new BnB algorithm for MKP that employs the two components to reduce the search space. Extensive experiments show that both the refined upper bound and the inprocessing strategy are very efficient in the reduction of search space. The new algorithm outperforms the state-of-the-art algorithms on the tested benchmarks significantly.

1 Introduction

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Two vertices are neighbors if they are adjacent in G. Given an integer k, a subgraph G' of G is a k-plex if every vertex in G' has at most $k-1$ non-neighbors. The decision of existence of a k-plex of a given size in graphs is NP-complete [Balasundaram *et al.*, 2011].

Given a graph G and an integer k, the Maximum k-plex Problem (MKP) consists in finding a k-plex with the largest number of vertices. MKP is a generalization of the Maximum Clique Problem (MCP), the latter is a special case of MKP (k is 1) and is a well-known NP-hard combinatorial optimization problem [Garey and Johnson, 1979].

The k-plex model finds many practical applications, such as social network analysis [Seidman and Foster, 1978] and scalable community detection[Conte *et al.*, 2018]. Due to the flexibility of the k-plex model, it is more suitable for the analysis of the graphs encoding from real-world applications, where noise and defective data are inevitable and the classic clique model is too rigorous to be practical.

[*]Corresponding author

Since MKP is relevant to practical applications, a lot of work has devoted to seeking for efficient algorithms, including exact and heuristic algorithms, for MKP in recent years [Gujjula *et al.*, 2014; Zhou and Hao, 2017; Xiao *et al.*, 2017; Miao and Balasundaram, 2017; Gao *et al.*, 2018; Zhou *et al.*, 2020; Zhou *et al.*, 2021; Jiang *et al.*, 2021].

In the aspect of heuristics, Krishna et al. [2014] proposed a greedy randomized adaptive search and tabu search meta-heuristic for MKP. Zhou and Hao [2017] proposed a frequency-driven multi-neighborhood tabu search for MKP on large networks. Chen et al. [2020] implemented an efficient local search which is based on a double-attributes incremental neighborhood updating and dynamic configuration checking strategies. Pullan presented a local search for MKP [Pullan, 2021]. These heuristics focus on obtaining a suboptimal solution within a reasonable solving time.

Exact search algorithms seek for the optimal solutions by exploring the search space systematically. Balasundaram et al. [2011] designed an integer programming formulation and a branch-and-cut algorithm for MKP. McClosky and Hicks [2012] presented a BnB algorithm with a co-k-plex-coloring upper bound. Xiao et al. [2017] proposed an exact algorithm with time complexity in $O(c^n n^{O(1)})$ for a constant $c < 2$ and each $k \geq 3$. Gao et al. [2018] proposed several reduction rules and a BnB algorithm with a dynamic vertex selection mechanism. Zhou et al. [2021] proposed a BnB algorithm with a second-order reduction and a graph coloring upper bound to reduce the search space. Jiang et al. [2021] gave a partition-based upper bound and designed an efficient BnB algorithm for MKP. Moreover, algorithms for enumerating all maximal k-plexes were studied in [Conte *et al.*, 2017; Zhou *et al.*, 2020] .

We note that exact algorithms for MKP, especially BnB algorithms, have achieved a significant advance in recent years and the advance mainly comes from improvements in two aspects: effective reduction rules and upper bounds [Xiao *et al.*, 2017; Gao *et al.*, 2018; Zhou *et al.*, 2021; Jiang *et al.*, 2021]. The reduction rules reduce graphs by exploiting the structural features of graphs and the upper bounds speed up the search of BnB algorithms by pruning the search space efficiently at every search tree node.

In this paper, we propose a refined upper bound and an efficient inprocessing strategy for MKP. We analyze the defects of existing methods for the partition-based upper bound and then propose a novel notion and a sophisticated algorithm to improve the upper bound. We present an inprocessing strategy that employs a graph reduction procedure to reduce graphs incrementally. Based on the two essential components, we implement a new BnB algorithm for MKP. Extensive experiments were performed to evaluate the performance of the algorithm and the effectiveness of the two components. Results show that both the refined upper bound and the inprocessing strategy are very efficient in the reduction of search space. The new algorithm outperforms the state-of-the-art exact algorithms significantly on the tested benchmarks .

The paper is organized as follows: Section 2 gives the notations and graph definitions used in this paper. Section 3 reviews previous upper bounds and presents a refined upper bound for MKP. Section 4 describes the inprocessing. Section 5 presents a new BnB algorithm. Section 6 reports on the empirical results. Section 7 concludes the paper.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with n vertices and m edges. Two vertices u and v are adjacent or neighbors, if there is an edge between u and v, i.e., $(u, v) \in E$. The set of neighbors of a vertex v is denoted as $N(v)$. The degree of vertex v in G, denoted by $deg(v)$, is the number of neighbors of v and the maximum degree in G is denoted as $deg(G)$. The density of G is computed as $2m/(n(n-1))$.

If V' is a subset of V, the induced subgraph $G[V']$ by V' is defined as $G[V'] = (V', E')$, where $E' = \{(u, v)|u, v \in V'$ and $(u, v) \in E\}$. Given a positive integer k, $G[V']$ is a k-plex if the degree of v in the induced subgraph is greater than or equal to $|V'| - k$ for each $v \in V'$. The size of a k-plex is the number of vertices in it. The size of the maximum k-plex in G is denoted by $\kappa(G)$ in this paper.

An Independent Set (IS) of G is a subset of V in which every two vertices are nonadjacent in G. Graph coloring is to assign a color to every vertex such that every two adjacent vertices receive different colors. In a feasible coloring, the vertices that receive the same color form an IS.

3 A Refined Upper Bound

Bounding strategies are crucial to the performance of BnB algorithms. In this section, we review previous upper bounds for MKP and present a refined upper bound.

Given a graph $G = (V, E)$ and an integer k, let $S = \{v_1, v_2, \ldots, v_q\}$ be a partial solution for MKP in G and let C be the set of candidate vertices that could join in S. In BnB algorithms for MKP, a partial solution S corresponds to a node in the search tree. In this paper, we mainly discuss the upper bound of the maximum k-plex of containing the current partial solution S. Existing BnB algorithms for MKP usually estimate the upper bound by partitioning the candidate set C of the current search tree node into subsets.

The graph coloring upper bound (COL for short), presented in [Zhou *et al.*, 2021], partitions the vertices of C into ISs $I_1, I_2, \ldots, I_p$ and then computes the upper bound as $|S| + \sum_{i=1}^{p} \min\{k, |I_i|\}$, because each IS I_i can contribute at most k vertices to a k-plex. COL derives an upper bound by considering the solution constraints within an IS.

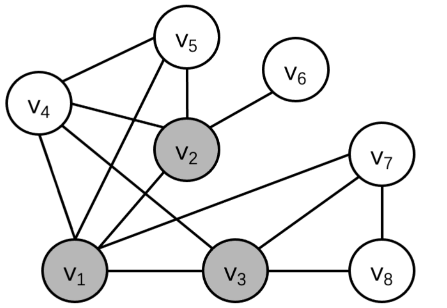


Figure 1: A graph with 8 vertices to illustrate the partitioning-based upper bound

The partition-based upper bound (PUB for short), proposed in [Jiang *et al.*, 2021], partitions the vertices of C into subsets $\Pi = \{\pi_0, \pi_1, \ldots, \pi_q\}$ *w.r.t.* the current partial solution $S = \{v_1, v_2, \ldots, v_q\}$, where each π_i ($i \geq 1$) is a subset of non-neighbors of vertex v_i and π_0 is the set of vertices in C that are adjacent to every vertex $v_i \in S$. Moreover, PUB maintains an array $\Delta = \{\delta_1, \delta_2, \ldots, \delta_q\}$ for S, where each δ_i is the number of vertices in S that are nonadjacent to v_i. PUB is computed as $|S| + |\pi_0| + \sum_{i=1}^{q} \min\{k-1-\delta_i, |\pi_i|\}$, because each $\pi_i \in \Pi$ ($i \geq 1$) can contribute at most $k-1-\delta_i$ vertices to S. PUB derives an upper bound by considering the solution constraints of the current partial solution.

The experimental results show that PUB is more effective for pruning the search space than COL and other upper bounds [Jiang *et al.*, 2021]. However, as the authors point out, the quality of PUB is relevant to the partition Π of C. Since a vertex in C could be a non-neighbor of many vertices in S. The total number of eligible partitions of C could be a power of $|C|$. Different partitions of C derive different computational results of PUB. We use Figure 1, which is originally presented in [Jiang *et al.*, 2021], to illustrate the difference.

Example 1. *Let k be* 3 *and the partial solution S of G in Figure 1 be* $\{v_1, v_2, v_3\}$*. Then, the* Δ *array of S is* $\{0, 1, 1\}$ *and the candidate set C is* $\{v_4, v_5, v_6, v_7, v_8\}$*. The following two partitions* Π_1 *and* Π_2 *are eligible.* Π_1 *is* $\{\pi_0 = \{v_4\}, \pi_1 = \emptyset, \pi_2 = \{v_7, v_8\}, \pi_3 = \{v_5, v_6\}\}$ *and its PUB is computed as:* $|S|+|\pi_0|+\sum_{i=1}^{3} \min\{k-1-\delta_i, |\pi_i|\} = 3+1+0+1+1 = 6$*.* Π_2 *is* $\{\pi_0 = \{v_4\}, \pi_1 = \{v_6, v_8\}, \pi_2 = \{v_7\}, \pi_3 = \{v_5\}\}$ *and its PUB is computed as:* $3 + 1 + 2 + 1 + 1 = 8$*.*

Note that the PUB bound of Π_2 is 2 bigger than that of Π_1 in Example 1. In fact, using the ordering $v_1 > v_2 > v_3$, the sequential construction algorithm proposed in [Jiang *et al.*, 2021] produces Π_2 rather than Π_1. We first analyze the reason why Π_1 computes a better upper bound than Π_2 does and then present a new algorithm that can derive a tighter upper bound than the one presented in [Jiang *et al.*, 2021].

Note that if a set π_i already contains $k - 1 - \delta_i$ vertices, inserting more vertices into it doesn't increase the value of $\min\{k - 1 - \delta_i, \pi_i\}$, but could decrease the value of other π_j because the number of vertices in π_j could be reduced. In the partition Π_2 in Example 1, π_2 and π_3 must contain one vertex (v_7 must be in π_2 and v_5 must be in π_3) and they have the maximum value of $\min\{k - 1 - \delta_i, \pi_i\}$. Therefore, the insertion of v_8 into π_2 and v_6 into π_3 does not increase the computational results of PUB for π_2 and π_3, but can decrease the computational result of PUB for π_1 by 2, because π_1 becomes an empty set after v_8 and v_6 are moved into π_2 and π_3, respectively. Obviously, to obtain a better upper bound, the vertex v_8 and v_6 should be inserted into π_2 and π_3 instead of π_1. However, we don't have any prior knowledge to determine which π a vertex should be inserted into.

Inspired by the above observation, we propose a refined upper bound, which can improve the quality of PUB. Recall that the PUB bound of each π_i is computed as $\min\{k-1-\delta_i, |\pi_i|\}$. We call it the cost for the construction of π_i, denoted by $cost(\pi_i)$, which is formulated as follows.

$$cost(\pi_i) = \min\{k-1-\delta_i, |\pi_i|\} \quad (1)$$

To obtain a tighter upper bound, we take the construction of the partition Π as a process of distributing every vertex u of C into a proper π_i with an objective to minimize the total cost. We propose a notion called *distribution efficiency*, denoted by $dise(\pi_i)$, for each $\pi_i \in \Pi$ ($i > 0$), which is the ratio of the cardinality of π_i to its $cost(\pi_i)$ and is formulated as follows.

$$dise(\pi_i) = |\pi_i|/cost(\pi_i) \quad (2)$$

Especially, if π_i is empty, $dise(\pi_i)$ is defined as 0. We evaluate every $\pi \in \Pi$ *w.r.t.* its distribution efficiency, i.e., we say π_i is better than π_j if $dise(\pi_i)$ is greater than $dise(\pi_j)$.

Now we present a new algorithm to derive a better partition of Π, which is called ***DisePUB*** and is depicted in Algorithm 1. DisePUB constructs Π with two steps. At the first step, it constructs an overlapped partition Π' of C, i.e., a vertex $u \in C$ is distributed into every $\pi_i \in \Pi'$ if u is non-adjacent to v_i of S. The initial upper bound ub is computed as $|S| + |\pi_0|$. At the second step, it picks a π_i in Π' with the largest distribution efficiency $dise(\pi_i)$ iteratively. Each time a π_i is picked, its $cost(\pi_i)$ is accumulated to ub, π_i is removed from Π' and is inserted into Π and all of the vertices in π_i are removed from the remaining $\pi \in \Pi'$. The process repeats till Π' becomes an empty set, i.e., every $\pi \in \Pi$ has been determined. The accumulated ub is an upper bound of the maximum k-plex of containing S.

We illustrate how DisePUB constructs the partition and computes the upper bound with Example 2.

Example 2. *Let k and S be the same as in Example 1. At the first step, DisePUB constructs the Π' as $\{\pi_1 = \{v_6, v_8\}, \pi_2 = \{v_7, v_8\}, \pi_3 = \{v_5, v_6\}\}$, $\pi_0 = \{v_4\}$ and the initial upper bound ub is $3+1=4$. Then, at the second step, DisePUB picks π one by one. At first, $dise(\pi_1), dise(\pi_2)$ and $dise(\pi_3)$ are computed as $1, 2$ and 2, respectively. Without loss of generality, suppose DisePUB picks π_2 at first and then v_7 and v_8 are removed from π_1 and π_3, ub is increased by $cost(\pi_2) = 1$. Next, DisePUB picks π_3 because $dise(\pi_3)$ is 2 and $dise(\pi_1)$ is 1 after removing vertices of π_2 and increases ub by $cost(\pi_3) = 1$. Finally, DisePUB picks π_1 which is an empty set after removing vertices of π_3 from π_1. DisePUB obtains a partition that is the same as Π_1 in Example 1, the upper bound is computed as 6. Note that the same partition can be obtained if π_3 is picked at first.*

DisePUB is quite different from the algorithm described in [Jiang *et al.*, 2021], wherein the algorithm generates every π_i sequentially and the computed upper bound depends on the ordering of vertices of S. DisePUB generates the partition with two steps that is guided by the novel notion of distribution efficiency. The computed upper bound of DisePUB is not related to the ordering of vertices being inserted into Π.

Algorithm 1 DisePUB(S, C, k), a partition algorithm based on distribution efficiency.

```
Input: The partial solution $S = \{v_1, v_2, \ldots, v_q\}$, the candidate set $C$, the $k$ value
Output: An upper bound of maximum $k$-plex containing $S$
1: Let $\Pi = \emptyset, \Pi' = \emptyset, S = \{v_1, v_2, \ldots, v_q\}$;
2: Let $\pi_0 = C \cap N(v_1) \cap \cdots \cap N(v_q)$, $ub = |S| + |\pi_0|$;
3: $\Pi \leftarrow \Pi \cup \{\pi_0\}$;
4: for $i$=1 to $q$ do
5:     create a $\pi_i = C \setminus N(v_i)$;
6:     $\Pi' = \Pi' \cup \{\pi_i\}$;
7: end for
8: repeat
9:     select a $\pi_i \in \Pi'$ with the largest $dise(\pi_i)$;
10:    $\Pi \leftarrow \Pi \cup \{\pi_i\}$, $\Pi' \leftarrow \Pi' \setminus \{\pi_i\}$, $ub \leftarrow ub$+$cost(\pi_i)$;
11:    remove all vertices in $\pi_i$ from each $\pi \in \Pi'$;
12: until $\Pi' = \emptyset$
13: return $ub$;
```

In Algorithm 1, lines 2-7 implement the first step, constructing an overlapped partition Π', which has a time complexity of $O(|S| \times |C|)$. Lines 8-12 implement the second step, picking a π_i with the largest $dise(\pi_i)$ iteratively. Each time a π_i is determined, the $dise(\pi)$ of the remaining $\pi \in \Pi'$ needs to be recomputed. The time complexity of the second step is $O(|S|^2 \times |C|)$. Therefore, the total time complexity of Algorithm 1 is $O(|S|^2 \times |C|)$.

In section 5, we will explain how to adapt Algorithm 1 to reduce the number of branches in a BnB algorithm.

4 Inprocessing for MKP

Previous work has proven that graph reduction is very effective for solving MKP in real-world graphs. In this section, we present an efficient reduction procedure, which is based on a lower bound of $\kappa(G)$ and can be used as an inprocessing strategy in a BnB algorithm to reduce graphs incrementally.

BnB algorithms usually begin their search with an initial solution S_0. We use $lb = |S_0|$ to denote the initial lower bound of $\kappa(G)$. Existing graph reduction rules for MKP are mainly based on the following property.

Property 1. *Given a graph G and a vertex v, the maximum k-plex of containing v is smaller than or equal to $deg(v) + k$.*

According to Property 1, if we are searching for a k-plex S of size greater than lb, then those vertices with degree smaller than or equal to $lb-k$ are redundant and can be removed from G iteratively in the preprocessing stage. [Gao *et al.*, 2018; Zhou *et al.*, 2021; Jiang *et al.*, 2021].

According to the number of common neighbors of two vertices u and v, enhanced reduction rules are proposed in [Zhou *et al.*, 2021; Jiang *et al.*, 2021], which are based on the following property.

Algorithm 2 Reduce(G, k, lb), a reduction algorithm for MKP.

Input: A graph G, k value and the lower bound lb of $\kappa(G)$.
Output: A reduced graph G'

```
1: repeat
2:     remove every vertex $v$ if $deg(v) \leq lb - k$;
3:     remove every edge $(u, v)$ if $cone(u, v) \leq lb - 2k$;
4: until there is no vertex and edge be removed from $G$
5: return the reduced graph $G'$;
```

Property 2. *Let $cone(u, v)$ be the number of common neighbors of vertex u and v in G. The maximum k-plex of containing u and v is smaller than or equal to $cone(u, v) + 2k - \gamma$, where γ is 0 if $(u, v) \in E$; Otherwise, γ is 2.*

Zhou et al. [2021] use Property 2 to remove an edge (u, v) if $cone(u, v) \leq lb - 2k$ in the preprocessing. Jiang et al. [2021] use the property to reduce graphs in the preprocessing and to reduce the number of candidate vertices when the search tree level is less than 3.

We note that the efficiency of reduction rules based on Property 1 and 2 is relevant to the quality of the lower bound lb. The higher the lower bound lb is, the more vertices could be removed from G. However, because of the NP-hardness of MKP, there could be a big gap between the initial lower bound lb and $\kappa(G)$, making the application of reduction rules inadequate if they are just applied in the preprocessing stage.

Based on the above observation, we propose a new approach to perform graph reduction, i.e., performing reduction as an inprocessing. The graph reduction procedure is depicted in Algorithm 2, which performs reduction rules based on Property 1 and 2 iteratively *w.r.t.* a given lower bound lb of $\kappa(G)$. The procedure is designed to be integrated into a BnB algorithm as an inprocessing, which is triggered once the lower bound lb is updated during the search.

The time complexity of removing a vertex (an edge) is $O(deg(G))$ and the computation of common neighbors for edges is $O(|E|^{1.5})$[Latapy, 2008]. Therefore, the total time complexity of inprocessing is $O(|E|^{1.5}+(|V|+|E|) \cdot deg(G))$.

5 A New BnB Algorithm for MKP

We present a new BnB algorithm for MKP, called DiseMKP, which is depicted in Algorithm 3. The algorithm first generates an initial solution S_0 and then calls Algorithm 2 to reduce the input graph G to G'. The initial solution S_0 is generated with the heuristic presented in [Jiang *et al.*, 2021]. Let $v_1 < v_2 < \cdots < v_n$ be the degeneracy ordering. If v_i is the smallest vertex with degree of greater than or equal to $n - i + 1 - k$, then $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ is a k-plex. After reducing the graph, DiseMKP calls a BnB search procedure to search for an optimal solution in the reduced graph G'.

The BnB search procedure is described in Algorithm 4, which implements the BnB search approach proposed in [Li *et al.*, 2017]. Let S be the partial solution and C be the set of candidate vertices. Instead of exploring the whole search space formed by the set C, Algorithm 4 calls a function, called *PartitionDise*, to identify a minimal set B of branching vertices and only branches on vertices in B *w.r.t.* the degeneracy ordering of vertices of B. Each time the search procedure finds a better solution S^*, it calls the reduction procedure (Algorithm 2) to perform graph reduction, which implements an inprocessing. After the reduction, Algorithm 4 backtracks to the root node of the search tree, clears the removed vertices from B and C and then the search proceeds from the first unbranched branching vertex.

Algorithm 3 DiseMKP(G, k), a BnB algorithm for MKP

Input: A graph $G = (V, E)$ and an integer k
Output: a maximum k-plex S^* in G.

```
1: generate an initial solution $S_0$;
2: $G' = (V', E') \leftarrow$ Reduce($G, k, |S_0|$);
3: return BnBSearchMKP($G', \emptyset, V', S_0, k$);
```

The function PartitionDise is described in Algorithm 5, which partitions the candidate set C into B and P subject to $\kappa(G[S \cup P]) \leq |S^*|$. At the beginning, B is set to C. The goal of PartitionDise is to minimize B. Let $\beta = |S^*| - |S|$. We call β the total budget for the minimization of B. Algorithm 5 first constructs an overlapped partition Π and then select a π^* with the highest distribution efficiency $dise(\pi^*)$ and $cost(\pi^*) \leq \beta$. Each time a π^* is determined, the vertices in π^* are removed from B and from the remaining π in Π, and the budget is reduced by $cost(\pi^*)$. The process repeats till the budget β is depleted or no π can be selected. If $\beta > 0$ and B is not empty, Algorithm 5 removes $\min\{\beta, |B|\}$ more vertices from B. Obviously, those vertices that are removed from B form the set P and $\kappa(G[S \cup P]) \leq |S^*|$ holds. Finally, Algorithm 5 returns the reduced B as the branching set.

The time complexity of Algorithm 5 is the same with the time complexity of Algorithm 1. Experimental results of the new algorithm, the effects of the refined upper bound and the inprocessing will be given in the next section.

Algorithm 4 BnBSearchMKP(G, S, C, S^*, k)

Input: A graph G=(V,E), a partial solution S, the candidate set C and the incumbent solution S^* and the k value.
Output: the best solution S^* in G.

```
1: remove every $v \in C$, if $|S \setminus N(v)| \geq k$;
2: if $C$ is empty then return $S$;
3: $B \leftarrow$PartitionDise($C$, $k$, $S$, $S^*$);
4: if $B$ is empty then return $S^*$;
5: Let $u_1 < u_2 < \cdots < u_{|B|}$ be the degeneracy ordering of
   vertices of $B$ and $C' = C \setminus B$.
6: for $i = |B|$ to 1 do
7:     $S' \leftarrow$ BnBSearchMKP($G$, $S \cup \{u_i\}$, $C'$, $S^*$, $k$);
8:     if $|S'| > |S^*|$ then
9:         $S^* \leftarrow S'$;
10:        $G \leftarrow$ Reduce($G, k, |S^*|$);
11:        backtrack to the root node and proceed the search
           from the first unbranched branching vertex;
12:    end if
13:    $C' \leftarrow C' \cup \{u_i\}$;
14: end for
15: return $S^*$;
```

Algorithm 5 PartitionDise(C, k, S, S^*), generates the branching set B with the upper bound DisePUB

Input: The candidate set C, the k value, the partial solution S and the best solution S^* found so far
Output: A branching set B

```
1: Let $S = \{v_1, v_2, \ldots, v_q\}$, $B = C$, $\Pi = \emptyset$;
2: for $i = 1$ to $q$ do
3:     create a $\pi_i = C \setminus N(v_i)$;
4:     $\Pi = \Pi \cup \{\pi_i\}$;
5: end for
6: Let $\beta = |S^*| - |S|$;
7: repeat
8:     $\pi^* \leftarrow \emptyset$, $max\_dise \leftarrow 0$;
9:     for each non-empty $\pi \in \Pi$ do
10:        if $cost(\pi) <= \beta$ and $dise(\pi) > max\_dise$ then
11:            $\pi^* \leftarrow \pi$, $max\_dise \leftarrow dise(\pi)$;
12:        end if
13:    end for
14:    if $\pi^* \neq \emptyset$ then
15:        $B \leftarrow B \setminus \pi^*$, $\Pi \leftarrow \Pi \setminus \{\pi^*\}$, $\beta \leftarrow \beta - cost(\pi^*)$;
16:        $\pi \leftarrow \pi \setminus \pi^*$ for each $\pi \in \Pi$;
17:    end if
18: until $\pi^* = \emptyset$ or $\beta = 0$
19: if $\beta > 0$ and $B \neq \emptyset$ then
20:     remove $\min\{\beta, |B|\}$ vertices from $B$;
21: end if
22: return $B$;
```

6 Experimental Results

We empirically evaluated the new algorithm DiseMKP and its two essential components: the refined upper bound and the inprocessing strategy. DiseMKP was compared with the following three state-of-the-art BnB algorithms:

- BnBK[1]: It is the BnB algorithm proposed in [Gao *et al.*, 2018]. BnBK employs several sophisticated reduction rules and a dynamic vertex selection branching heuristic.
- Maplex[2]: It is the BnB algorithm proposed in [Zhou *et al.*, 2021]. Maplex implements a second-order reduction in the preprocessing and employs the graph coloring upper bound to reduce the search space.
- KpLeX[3]: It is the BnB algorithm proposed in [Jiang *et al.*, 2021]. KpLeX integrates the graph reduction rules in the preprocessing and the partitioning-based upper bound PUB to reduce the number of branches.

DiseMKP[4] and the three compared algorithms were implemented in C++ and compiled using GNU g++ -O3. Experiments were performed on AMD EPYC CPUs 7702 @2.0GHz under Linux with 128GB of memory. We tested a total of 675 graphs from three different datasets to evaluate DiseMKP and the compared algorithms.

Algorithm	$k = 2$	$k = 4$	$k = 6$	$k = 8$
BnBK	516	444	439	419
Maplex	527	457	387	360
KpLeX	556	507	478	446
DiseMKP	**580**	**545**	**517**	**499**

Table 1: The total numbers of instances solved by DiseMKP, KpLeX, Maplex and BnBk at $k = 2, 4, 6$ and 8. The cutoff time is 1800 seconds.

- **DIMACS graphs[5]:** The dataset contains 80 graphs with the number of vertices up to 4000 and densities ranging from 0.03 to 0.99. The dataset has been widely used to evaluate algorithms for MCP and MKP.
- **DIMACS10 graphs[6]:** The dataset contains 84 graphs with the number of vertices up to 2×10^7. The dataset has been widely used to evaluate graph algorithms like MCP and MKP.
- **Real-world graphs from Network Repository[7]:** The dataset contains 511 representative real-world graphs which we select from Network Data Repository [Rossi and Ahmed, 2015], including: biological networks (36), dynamic networks (88), interaction networks (29), labeled networks (104), recommendation networks (12), road networks (15), scientific computing (11), social networks (72), facebook (114) and web networks (30).

We use four different k values: $2, 4, 6$ and 8 to evaluate the algorithms. In general, the difficulty of solving MKP increases dramatically as the k value increases.

6.1 Performance Profiles

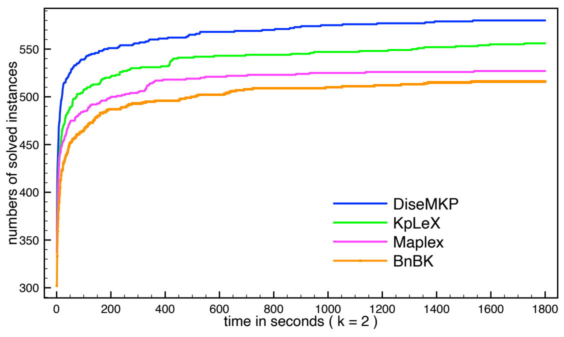
The first experiment we conducted is to evaluate the total performance of DiseMKP and the three compared algorithms. Algorithms were tested over the 675 instances with a cutoff time of 1800s for each instance. The total numbers of instances solved by each algorithm are reported in Table 1.

From Table 1, we can see that our new algorithm DiseMKP can solve more instances than the compared algorithms for every tested k value. The performance superiority of DiseMKP over the three compared algorithms increases steadily as the increase of k value. Especially, DiseMKP can solve 499 instances at $k = 8$, which is 11.9%, 38.6% and 19% more than the number of instances solved by KpLeX, Maplex and BnBK, respectively.

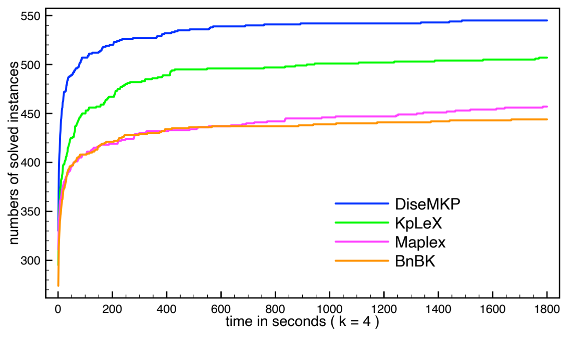
We plotted the cumulative numbers of solved instances of the four algorithms for each k value. The results are detailed in Figure 2. From Figure 2, we can see that the biggest gaps of cumulative numbers of solved instances between DiseMKP and the compared algorithms almost always occur within the first 200s for each tested k, showing that DiseMKP is steadily faster than the compared algorithms. Moreover, we can see that Maplex and BnBK have a comparable performance with KpLeX at $k = 2$, but their performance declines dramatically

[1]https://github.com/JimNenu/codekplex
[2]https://github.com/ini111/Maplex
[3]https://github.com/huajiang-ynu/kplex
[4]Published at https://github.com/huajiang-ynu/ijcai23-kpx

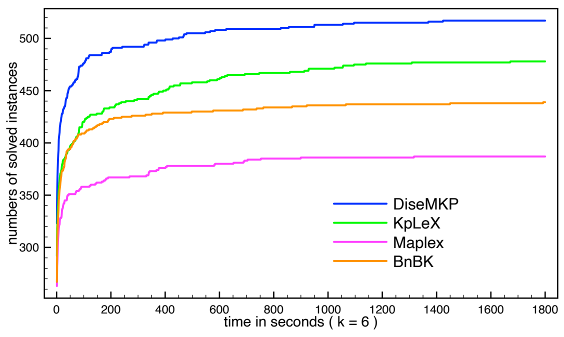
[5]http://archive.dimacs.rutgers.edu/pub/challenge/graph/
[6]https://www.cc.gatech.edu/dimacs10/downloads.shtml
[7]http://networkrepository.com/networks.php



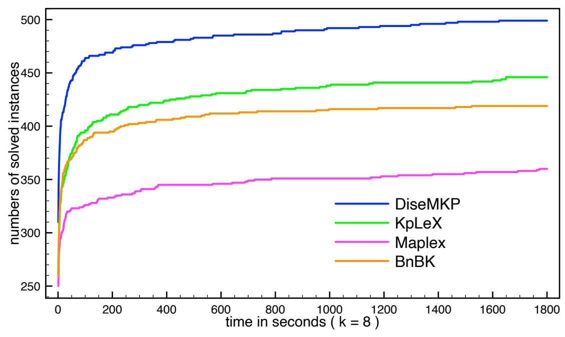
(a) Cumulative numbers of solved instances with k=2



(b) Cumulative numbers of solved instances with k=4



(c) Cumulative numbers of solved instances with k=6



(d) Cumulative numbers of solved instances with k=8

Figure 2: The cumulative numbers of instances solved by DiseMKP, KpLeX, Maplex and BnBk. The cutoff time is 1800 seconds.

when k increases to 4. KpLeX also shows an obvious decline in performance when k increases to 8. Whereas, the performance of DiseMKP declines tardily as k increases. DiseMKP is the most robust among the four tested algorithms.

In general, DiseMKP outperforms the three state-of-the-art algorithms KpLeX, BnBK and Maplex on the tested datasets significantly.

6.2 Effect of the New Upper Bound

To evaluate the effect of the new upper bound DisePUB for pruning the search space, we implemented three variants of DiseMKP, in which the inprocessing procedure (line 10 and 11 in Algorithm 4) is disabled and the *Partition* function is implemented with the three different upper bounds.

- **BnB$_{col}$:** It generates the branching set B with graph coloring bound. Let $lb = |S^*| - |S|$. BnB$_{col}$ constructs IS $I_1, I_2, \ldots, I_p$ one by one and computes an upper bound ub as $\sum_{i=1}^{p} \min\{k, |I_i|\}$ subject to $ub \leq lb$. BnB$_{col}$ takes the candidate vertices that are not inserted into the constructed ISs as the vertices of the set B.
- **BnB$_{pub}$:** It generates the branching set B with the partition function presented in [Jiang *et al.*, 2021], i.e., constructing each π_i of Π one by one $w.r.t.$ the vertex ordering in the partial solution S subject to the computed upper bound of Π is not greater than lb. BnB$_{pub}$ takes vertices that are not in Π as the vertices of the set B.
- **BnB$_{dise}$:** It is DiseMKP but the inprocessing procedure is disabled (line 10 and 11 in Algorithm 4 are removed). BnB$_{dise}$ generates the branching set B with our new upper bound DisePUB that is depicted in Algorithm 5.

We conducted an experiment to compare BnB$_{col}$, BnB$_{pub}$ and BnB$_{dise}$ using 40 representative instances with a cutoff time of 3600s and $k = 4$. We report the search tree size in 10^5 and the running time in seconds of each instance of the three algorithms. Table 2 shows the results of the comparison.

In total, BnB$_{dise}$ solves all of the 40 instances, BnB$_{pub}$ solves 33 instances and BnB$_{col}$ solves only 7 instances within the cutoff time. It is easy to see that the search tree sizes of BnB$_{dise}$ are significantly smaller than that of BnB$_{col}$ and BnB$_{pub}$, especially on hard instances. For example, the search tree sizes of BnB$_{dise}$ for *hamming6-2* and *socfb-UGA50* are 305.9 and 76.4, which are only 3.1% and 2.1% of the search tree sizes of BnB$_{pub}$ for the two instances, respectively. Consequently, BnB$_{dise}$ are almost always several times faster than BnB$_{pub}$ on hard instances in Table 2.

The comparison shows that our new upper bound DisePUB is more efficient in reducing search space than the COL and PUB upper bounds for BnB algorithms for MKP.

6.3 Effect of the Inprocessing Procedure

To investigate the effect of the inprocessing procedure in DiseMKP, we compared DiseMKP with a variant DiseMKP-NOIP, which is DiseMKP but the inprocessing procedure (line 10 and 11 in Algorithm 4) is disabled. We tested DiseMKP and DiseMKP-NOIP over the 675 instances with $k = 2$ and 4 using a cutoff time of 1800s.

Instances	BnB$_{col}$		BnB$_{pub}$		BnB$_{dise}$	
	tree	time	tree	time	tree	time
brock200-2	-	-	1234	221.1	**301.7**	**147.4**
c-fat200-1	416.4	15.12	**0.01**	0.03	**0.01**	**0.02**
c-fat500-1	15362	1091	**0.02**	**0.02**	**0.02**	**0.02**
hamming6-2	-	-	9607	2138	**305.9**	**157.6**
johnson8-4-4	-	-	932.6	137.9	**193.0**	**86.74**
p-hat300-1	-	-	25.58	3.97	**7.08**	**2.99**
p-hat500-1	-	-	440.9	94.69	**118.7**	**70.7**
p-hat700-1	-	-	2964	805.6	**732.5**	**543.9**
MANN-a81	-	-	**0.01**	**32.28**	**0.01**	32.48
san200-0.9-1	-	-	**0.07**	**0.06**	**0.07**	**0.06**
san200-0.9-3	-	-	-	-	**2119**	**2413**
bio-CE-GN	31020	1361	22.7	2.37	**2.52**	**0.88**
bio-DM-CX	-	-	341.3	178.9	**182.6**	**117.2**
ia-wiki-Talk	-	-	98.1	24.56	**17.6**	**11.14**
ia-enron-large	-	-	23.5	4.65	**4.9**	**2.25**
ia-sx-mathoverflow	-	-	4335	2716	**336.6**	**352.0**
fb-CMU-Carnegie49	1405	**45.52**	1247	367.7	**161.1**	**84.67**
sc-pkustk13	-	-	552.1	**1043**	**426.4**	1351
socfb-A-anon	-	-	489.0	101.3	**71.3**	**51.23**
socfb-B-anon	-	-	697.3	335.3	**75.3**	**159.3**
socfb-CMU	1443	**46.96**	1240	362.32	**157.9**	84.15
socfb-FSU53	-	-	-	-	**1083**	**1378**
socfb-Indiana	-	-	-	-	**3954**	**3244**
socfb-MIT	-	-	429.9	127.6	**95.3**	**51.48**
socfb-Wisconsin87	105491	3452	11.6	5.76	**3.4**	**2.96**
socfb-UIllinois	-	-	1394	637.4	**141.9**	**129.6**
socfb-UF	-	-	-	-	**625.0**	**712.7**
socfb-UGA50	-	-	3610	3127	**76.4**	**83.7**
socfb-Stanford3	-	-	-	-	**2228**	**1755**
socfb-Texas84	-	-	-	-	**1475**	**1571**
soc-flixster	-	-	4205	2073	**291.1**	**251.7**
soc-LiveMocha	-	-	1691	640.9	**183.3**	**195.5**
soc-slashdot	-	-	190.4	71.64	**21.8**	**14.37**
soc-wiki-conflict	-	-	-	-	**1226**	**1595**
soc-youtube-growth	-	-	734.5	451.2	**62.9**	**87.68**
tech-WHOIS	-	-	602.9	512.1	**264.9**	**274.6**
tech-as-topology	-	-	2292	1035	**209.6**	**178.5**
web-baidu-baike	-	-	211.5	81.0	**25.1**	**37.67**
web-spam	2893	72.17	0.85	0.17	**0.17**	**0.09**
web-wiki-ch-internal	-	-	2474	2429	**149.4**	**336.4**

Table 2: Comparison of BnB$_{col}$, BnB$_{pub}$ and BnB$_{dise}$ on 40 representative instances. The cutoff time is 3600s. The search tree size is in 10^5 and time is in seconds.

Figure 3 shows the cumulative numbers of instances solved by DiseMKP and DiseMKP-NOIP. The results for $k = 2$ and 4 are plotted in dashed and solid lines, respectively. DiseMKP solves 580 and 545 instances and DiseMKP-NOIP solves 558 and 516 instances for $k = 2$ and 4, respectively. It is easy to see that without the inprocessing procedure, the performance of DiseMKP declines significantly.

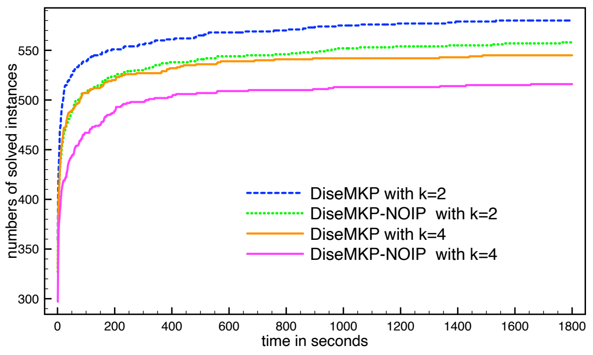


Figure 3: Comparison of DiseMKP with DiseMKP-NOIP using k=2 and 4. The cutoff time is 1800s.

Instance	lb	$\|V\|$	$\|E\|$	Density
sc-pkustk13	-	94893	3260967	0.00072429
	27	94614	3251544	0.00072646
	28	94614	3251544	0.00072646
	29	94614	3251544	0.00072646
	30	94471	3237739	0.00072557
	31	93831	3221467	0.00073181
	32	93813	3220327	0.00073183
	33	91680	3159599	0.00075183
	34	89415	3100882	0.00077571
	35	89415	3100882	0.00077571
	36	27120	644461	0.00175252
rec-epinions	-	755761	13396042	0.00004691
	9	105798	3418701	0.00061086
	10	59471	2126329	0.00120242
	11	36245	1357240	0.00206634
	12	21698	850540	0.00361331
	13	12677	519981	0.00647171
	14	7146	306208	0.01199448
	15	4001	175462	0.02192727
	16	2180	97234	0.04093874

Table 3: Two examples to illustrate the effectiveness of inprocessing in DiseMKP with $k = 4$. $|V|$ is the number of vertices and $|E|$ is the number of edges after graph reduction *w.r.t.* the lower bound lb.

To illustrate the effectiveness of the inprocessing, we selected two representative instances, *sc-pkustk13* and *rec-epinions*, to present their graph reduction processes in inprocessing as the increase of the lower bound lb. With $k = 4$, DiseMKP solves sc-pkustk13 and rec-epinions in 26.9s and 418s, respectively. But DiseMKP-NOIP solves sc-pkustk13 using 402s and cannot solve rec-epinions within 1800s.

The reduction processes are presented in Table 3. The first row of each instance shows the original numbers of vertices and edges and the second row shows the initial lower bound and the initial reduction result of the graph. The last row is the optimal solution. From Table 3, we can see that the two graphs are reduced dramatically as the increase of the lower bound lb, which can speed up the search of DiseMKP in turn.

In general, the inprocessing procedure can exploit graph reduction rules more efficiently and is essential to the performance of the new algorithm DiseMKP.

7 Conclusions

We proposed two efficient strategies for BnB algorithms for MKP, a refined upper bound and an inprocessing procedure. The upper bound is based on a novel notion of distribution efficiency and can compute a tighter bound than existing upper bounds for MKP. The inprocessing procedure implements incremental graph reduction, which can help BnB algorithms to reduce more search space. Based on the two components, we implemented a new BnB algorithm for MKP. Extensive experiments show that the two components are very efficient in the reduction of the search space in BnB algorithms. The new algorithm outperforms the state-of-the-art exact algorithms significantly on the tested benchmarks.

Acknowledgments

This work is funded by the National Natural Science Foundation of China (Grant NO. 62162066, 62162067 and 62162064), and in part supported by the project of Research and Application of Object Detection Based on Artificial Intelligence, by the project of Research on Data Driven Intelligent Logistics System Construction and Key Technologies (Grant NO. 202302AD080006), Science and Technology Department of Yunnan Province. The experiments were performed on Hefei advanced computing center.

References

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

[Chen *et al.*, 2020] Peilin Chen, Hai Wan, Shaowei Cai, Jia Li, and Haicheng Chen. Local search with dynamic-threshold configuration checking and incremental neighborhood updating for maximum k-plex problem. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 2343–2350. AAAI Press, 2020.

[Conte *et al.*, 2017] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. Fast enumeration of large k-plexes. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 115–124. ACM, 2017.

[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 Yike Guo and Faisal Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1272–1281. ACM, 2018.

[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 Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1449–1455. ijcai.org, 2018.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[Gujjula *et al.*, 2014] Krishna Reddy Gujjula, Krishnan Ayalur Seshadrinathan, and Amirhossein Meisami. A hybrid metaheuristic for the maximum k-plex problem. In Sergiy Butenko, Eduardo L. Pasiliao, and Volodymyr Shylo, editors, *Examining Robustness and Vulnerability of Networked Systems*, volume 37 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 83–92. IOS Press, 2014.

[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 Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1689–1696. ijcai.org, 2021.

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

[Li *et al.*, 2017] Chu Min Li, Hua Jiang, and Felip Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & OR*, 84:1–15, 2017.

[McClosky and Hicks, 2012] Benjamin McClosky and Illya V. Hicks. Combinatorial algorithms for the maximum k-plex problem. *J. Comb. Optim.*, 23(1):29–49, 2012.

[Miao and Balasundaram, 2017] Zhuqi Miao and Balabhaskar Balasundaram. Approaches for finding cohesive subgroups in large-scale social networks via maximum *k*-plex detection. *Networks*, 69(4):388–407, 2017.

[Pullan, 2021] Wayne Pullan. Local search for the maximum k-plex problem. *J. Heuristics*, 27(3):303–324, 2021.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 4292–4293. AAAI Press, 2015.

[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.

[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 Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 919–925. AAAI Press, 2017.

[Zhou and Hao, 2017] Yi Zhou and Jin-Kao Hao. Frequency-driven tabu search for the maximum s-plex problem. *Comput. Oper. Res.*, 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 *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 2442–2449. AAAI Press, 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 *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12453–12460. AAAI Press, 2021.