A New Upper Bound Based on Vertex Partitioning for the Maximum k-plex Problem

Hua Jiang[1] , **Dongming Zhu**[1] , **Zhichao Xie**[1] , **Shaowen Yao**[1] and **Zhang-Hua Fu**[2,3*]
[1]Engineering Research Center of Cyberspace & School of Software, Yunnan University, China
[2]Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen, China
[3]The Chinese University of Hong Kong, Shenzhen, China
huajiang@ynu.edu.cn, {zmd140610,xzc}@mail.ynu.edu.cn, yaosw@ynu.edu.cn,
fuzhanghua@cuhk.edu.cn

Abstract

Given an undirected graph, the Maximum k-plex Problem (MKP) is to find a largest induced subgraph in which each vertex has at most $k-1$ non-adjacent vertices. The problem arises in social network analysis and has found applications in many important areas employing graph-based data mining. Existing exact algorithms usually implement a branch-and-bound approach that requires a tight upper bound to reduce the search space. In this paper, we propose a new upper bound for MKP, which is a partitioning of the candidate vertex set with respect to the constructing partial solution. We implement a new branch-and-bound algorithm that employs the new upper bound to reduce the number of branches. Experimental results show that the upper bound is very effective in reducing the search space. The new algorithm outperforms the state-of-the-art algorithms significantly on real-world massive graphs, DIMACS graphs and random graphs.

1 Introduction

In an undirected graph $G = (V, E)$, a k-plex is a subset S of V satisfying that each vertex in the induced subgraph by S, denoted by $G[S]$, has a degree greater than or equal to $|S|-k$, i.e., every vertex in S can have at most $k-1$ non-neighbors in $G[S]$. When k is 1, $G[S]$ is a complete graph and S is called a *clique* of G.

k-plex is a clique relaxation model, which was originally introduced for social network studies [Seidman and Foster, 1978]. A k-plex defines a cohesive subgroup by restricting the number of missing connections of every member in the subgroup. Compared to the restrictive clique model, k-plex is more suitable for the analysis of the massive graphs encoding from real-world problems, because real-world cohesive subgraphs do not need to meet the rigorous constraint of cliques and could be missing a few connections. Due to the relevance to practical applications, the research attention on k-plex sustainably grows in recent years [Balasundaram *et al.*, 2011; Xiao *et al.*, 2017; Miao and Balasundaram, 2017; Gschwind *et al.*, 2018; Conte *et al.*, 2018; Gao *et al.*, 2018; Zhou *et al.*, 2020; Zhou *et al.*, 2021].

Searching for a largest clique in a given graph G is known as the Maximum Clique Problem (MCP), which is NP-Hard [Garey and Johnson, 1979] and is a prominent research area in the past thirty years. The Maximum k-plex Problem (MKP for short) is to find a k-plex with the maximum number of vertices in G. Obviously, MCP is the maximum 1-plex problem, which is a special case of MKP. Therefore, the difficulty of solving MKP is not lower than solving MCP. In practice, MKP is more intractable than MCP, because the restriction of adjacency of every two vertices in a clique is relaxed in a k-plex when $k > 1$.

Compared to MCP, fewer algorithms were proposed for MKP. Existing algorithms for MKP fall into two categories, heuristic and exact. The heuristic algorithms usually employ local search approaches and dedicate to searching for a suboptimal solution within an acceptable time [Zhou and Hao, 2017; Chen *et al.*, 2020]. The exact algorithms usually implement approaches, such as branch-and-bound/cut/search, to systemically explore the search space to seek the optimality [McClosky and Hicks, 2012; Xiao *et al.*, 2017; Gao *et al.*, 2018; Zhou *et al.*, 2021].

We note that although there are several efficient exact algorithms for MCP, the performance of existing exact algorithms for MKP is still inadequate. One reason lies in the lack of good pruning strategies, namely, effective upper bounds. The upper bounds that are effective for MCP, such as upper bounds based on vertex coloring [Tomita *et al.*, 2010] and on MaxSAT reasoning [Li and Quan, 2010; Li *et al.*, 2017; Jiang *et al.*, 2017], are hard to be applied to MKP efficiently, because of the relaxation of the adjacency constraint of every two vertices in solutions. McClosky et al [2012] proposed two upper bounds based on a notion of co-k-plex coloring for MKP. But, their empirical results showed that the two bounds were difficult to be made efficient in practice.

In this paper, we propose a novel and efficient upper bound for MKP. The bound is based on a partitioning of the candidate vertex set with respect to the current growing partial solution. Using the partition, we can derive an upper bound of the maximum k-plex that can be extended from the current partial solution. We implement a branch-and-bound (BnB) algorithm and integrate the upper bound into the algorithm to reduce the number of branches at each search tree node.

*Corresponding author

Extensive experiments were conducted to evaluate the performance of the new algorithm and the new upper bound. The results show that the new upper bound can significantly reduce the search space. Thanks to the new bound, the proposed BnB algorithm outperforms the state-of-the-art algorithms on real-world massive graphs, DIMACS graphs and random graphs.

The paper is organized as follows: Section 2 gives some basic graph definitions and properties for k-plex. Section 3 reviews previous BnB algorithms for MKP. Section 4 presents the new upper bound. Section 5 describes a new BnB algorithm for MKP. Section 6 reports on the empirical results. Section 7 gives the conclusions.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph, where V is a set of n vertices, E is a set of m edges. The density of G is defined as $2m/(n(n-1))$. The complement graph of G is defined as $\bar{G} = \{V, \bar{E}\}$, where $\bar{E} = \{(u, v)|(u, v) \notin E\}$. Two vertices u and v of V are adjacent or neighbors, if $(u, v) \in E$. The set of neighbors of a vertex v in G is denoted by $N(v) = \{u|(u, v) \in E\}$. The cardinality of $N(v)$ is the degree of v, denoted by $deg_G(v)$ or simply $deg(v)$ when the context is clear. A subset I of V is an independent set if every two vertices in I are nonadjacent. We use $G[S]$ to denote the subgraph of G induced by the subset S of V. The following definitions and properties are related to k-plex.

Definition 1. *Given a positive integer k, a subset S of V is a* k-plex, *if $deg_{G[S]}(v) \geq |S| - k$ for each $v \in S$.*

Note that any subset S' of a k-plex S is also a k-plex, which indicates the *hereditary* property of k-plexes. The maximum k-plex in G is a k-plex with the largest cardinality and the cardinality is denoted by $\omega_k(G)$ in this paper.

Definition 2. *Given a positive integer k, a subset S of V is a* co-k-plex, *if $deg_{G[S]}(v) \leq k - 1$ for each $v \in S$.*

It is easy to see that a co-k-plex in G is a k-plex in $\bar{G}$. The following two properties define an upper bound of $\omega_k(G)$ and an upper bound of a k-plex S containing a given vertex v.

Property 1. *Let $\varepsilon(G)$ be the maximum degree of vertices of G, then $\omega_k(G) \leq \varepsilon(G) + k$.*

Property 2. *If S is a k-plex containing vertex v in G, then $|S| \leq deg_G(v) + k$.*

Let $\Theta_G(u, v)$ denote the set of common neighbors of two vertices u and v in G, i.e., $\Theta_G(u, v) = N(u) \cap N(v)$. The following Property 3 gives an upper bound of a k-plex S containing both u and v.

Property 3. *If S is a k-plex containing vertex u and v in G, then it holds that $|S| \leq |\Theta_G(u, v)| + 2k - \gamma$, where γ is 0 if $(u, v) \in E$; Otherwise, γ is 2.*

The following two properties describe relations of a k-plex with a co-k-plex and an independent set.

Property 4. *If G is a co-k-plex, then $\omega_k(G) \leq 2k - 2 + (k \ mod \ 2)$.*

Property 5. *Let I be an independent set of vertices in G, then a k-plex of G can contain at most* $\min\{|I|, k\}$ *vertices of I.*

Property 1 and 2 are straightforward. Property 3 and 5 can be derived from the definition of a k-plex. Property 4 gives an upper bound of $\omega_k(G)$, when G is a co-k-plex [Balasundaram *et al.*, 2011].

3 Related Works

In this section, we review representative upper bounds and exact algorithms for MKP.

The first exact algorithm we reviewed is a branch-and-cut algorithm [Balasundaram *et al.*, 2011]. The algorithm encodes MKP as an integer programming problem and combines a peeling procedure to recursively remove the vertices that cannot belong to a k-plex of size greater than the incumbent. The removing rule is based on Property 3, i.e., when considering a k-plex containing vertex u, then the upper bound of a k-plex simultaneously containing another v is computed with Property 3. If the bound is not greater than the incumbent, v can be removed from the candidate set.

McClosky et al [2012] introduced two upper bounds for MKP based on *co-k-plex coloring*. Let co-k-plex $C_1, C_2, \ldots, C_p$ partition the vertex set of G, then $\sum_{i=1}^{p} \min\{2k - 2 + (k \ mod \ 2), |C_i|, \varepsilon(G[C_i]) + k\}$ is an upper bound of $\omega_k(G)$. The bound is derived from Property 1 and 4. The authors proposed two heuristics to compute the partitioning and implemented two BnB algorithms based on the two upper bounds. However, their empirical results showed that the two algorithms did not benefit from the upper bounds, showing that the co-k-plex coloring heuristics might not produce tight upper bounds for MKP.

Algorithms for complex networks analysis, including algorithms for MKP over massive graphs, have received a lot of attention in recent years. Xiao et al [2017] proposed a branch-and-search algorithm to solve MKP for massive graphs. The authors investigated several structural properties that could be used to prune the search branches. The experimental results showed that, with well-designed reduction and pruning rules, massive graphs with tens of thousands of vertices could be solved within reasonable times. Additionally, the authors proved that the trivial exponential bound of 2^n of MKP for $k \geq 3$ can be broken.

Gao et al [2018] proposed several new graph reduction methods for MKP in massive graphs. Those methods are used to reduce the search space in the preprocessing and the search phases. Combining a dynamic branching vertex selection heuristic, the authors implemented a BnB algorithm that can solve graphs with millions of vertices in a very short time. Compared to the sophisticated reduction rules, the bounding strategy in the algorithm is simple; Use the number of remaining vertex whose degree exceeds the lower bound minus k as the bound, which is the direct application of Property 2.

Zhou et al [2021] proposed a BnB algorithm with a second-order preprocessing and graph color bounding for MKP. The algorithm partitions the candidate vertices into independent sets and computes an upper bound based on Property 5. Their experimental results showed that the graph coloring bound could be effective to reduce search space when the graphs are very sparse.

We note that although existing exact algorithms for MKP mainly implement branch-and-bound/cut/search approaches, the obtained performance improvement are mostly from the reduction and/or branching strategies. The bounding strategies used in those algorithms are still simple and weak.

4 A New Upper Bound

In this section, we propose a novel upper bound for k-plex, which is tight and computational efficient and can be used to reduce the search space for BnB MKP algorithms .

Given a graph $G = (V, E)$, let $S = \{v_1, v_2, \ldots, v_q\}$ be a growing k-plex in G and let P be the candidate set of vertices that could extend S. We define a partition $\Pi = \{\pi_0, \pi_1, \ldots, \pi_q\}$ of P *w.r.t.* the growing solution S. Π satisfies the following three conditions:

(1) $\bigcup_{i=0}^{q} \pi_i = P$ and $\pi_i \cap \pi_j = \emptyset$, for $i \neq j$ $(0 \leq i, j \leq q)$.

(2) π_0 is the set of vertices in P that are adjacent to every vertex in S, i.e., $\pi_0 = P \cap N(v_1) \cap N(v_2) \cap \cdots \cap N(v_q)$.

(3) Each π_i, $1 \leq i \leq q$, is a subset of vertices in P that are nonadjacent to the vertex v_i in S, i.e., $\pi_i \subseteq P \setminus N(v_i)$.

We define an array $\Delta = (\delta_1, \delta_2, \ldots, \delta_q)$ for the growing solution S. Each element δ_i is the number of non-neighbors of v_i in S, i.e., $\delta_i = |S \setminus N(v_i)| - 1$. With the partition Π of P and the Δ array of S, we propose the following lemma to compute an upper bound of the maximum k-plex that can be extended from S.

Lemma 1. *With a partition Π of P and the Δ array of S, the upper bound of the maximum k-plex containing S in G can be computed as $|S| + |\pi_0| + \sum_{i=1}^{q} \min\{k - 1 - \delta_i, |\pi_i|\}$.*

Proof. $|S| + |\pi_0|$ is an obvious upper bound of k-plex in $G[S \cup \pi_0]$. Since S is a k-plex and π_i is a set of non-neighbors of v_i in P, the maximum number of vertices in π_i that can be inserted into S is $k - 1 - \delta_i$, or $|\pi_i|$ if $|\pi_i| < k - 1 - \delta_i$. Then, the total number of vertices that can be inserted into S in $P \setminus \pi_0$ is no more than $\sum_{i=1}^{q} \min\{k - 1 - \delta_i, |\pi_i|\}$.Thus, $|S| + |\pi_0| + \sum_{i=1}^{q} \min\{k - 1 - \delta_i, |\pi_i|\}$ is an upper bound of the maximum k-plex containing S in G . □

Given a graph G, a growing k-plex S in G and a partition of the candidate set P, Lemma 1 defines a formula to compute an upper bound of the maximum k-plex that can be extended from S. We call the new upper bound *PUB*, a Partitioning-based Upper Bound for k-plex.

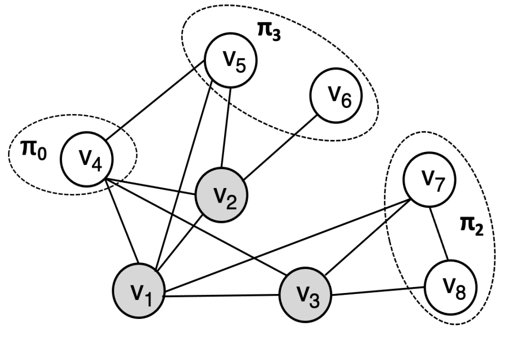


Figure 1: An example of the partitioning-based upper bound

Algorithm 1 KpLeX(G,k), an exact algorithm for MKP

```
Input: A graph $G = (V, E)$, the value $k$
Output: a maximum $k$-plex $S^*$ in $G$.
1: $(S_0, G' = (V', E')) \leftarrow$ Pre-Procedure($G$,$k$);
2: Let $S = \emptyset$, $S^* = S_0$, $P = V'$;
3: return BnBSearch($G'$, $S$, $P$, $S^*$, $k$);
```

Figure 1 illustrates PUB with a simple graph for 3-plex, where $S = \{v_1, v_2, v_3\}$ is the growing 3-plex and $P = \{v_4, v_5, v_6, v_7, v_8\}$ is the candidate set. The Δ array of S is computed as $(\delta_1 = 0, \delta_2 = 1, \delta_3 = 1)$, $\Pi = \{\pi_0 = \{v_4\}, \pi_1 = \emptyset, \pi_2 = \{v_7, v_8\}, \pi_3 = \{v_5, v_6\}\}$ is a partitioning of P *w.r.t.* S. According to Lemma 1, the upper bound of a 3-plex containing S in the graph is computed as: $|S| + |\pi_0| + \sum_{i=1}^{3} \min\{k-1-\delta_i, |\pi_i|\} = 3+1+0+1+1 = 6$.

To compute PUB, we need to compute a partitioning Π for P. Note that π_0 is always fixed for a given S and P, but there exist many possibilities to compute $\pi_1, \ldots, \pi_q$, because a vertex $v \in P$ could be nonadjacent to many vertices in S. A straightforward partitioning of P can be stated as follows: Iteratively construct a set π_i for each $v_i \in S$ with all the vertices in P that are not adjacent to v_i. Each time a π_i is constructed, the vertices of π_i are removed from P. The vertices that do not belong to any π_i $(i \neq 0)$ form the set π_0. The time complexity of the partition procedure is $O(|S| \times |P|)$.

Note that the new upper bound PUB is quite different from the previous upper bounds based on the co-k-plex partitioning and the graph coloring. For PUB, the partitioning of the candidate set P is based on the growing solution S. The relationship between S and P is exploited in the partition. However, for the co-k-plex and the graph coloring bound, the partitioning of P depends only on the value k and the subgraph $G[P]$. Therefore, it is reasonable to expect that the new bound PUB can derive a tighter upper bound than the previous two upper bounds.

5 A New BnB Algorithm with the New Upper Bound to Reduce the Number of Branches

We present a new algorithm for MKP, called KpLeX, which is depicted in Algorithm 1. KpLeX first calls a preprocessing to compute an initial solution S_0 and to reduce the input graph G to G', then calls a BnB search procedure to search for an optimal solution in the reduced graph G', which employs the new upper bound to reduce the search space. We first describe the preprocessing procedure, then the BnB search procedure.

5.1 The Preprocessing Procedure

The preprocessing procedure in Algorithm 1 performs two tasks: compute an initial solution S_0 and reduce the graph G.

To compute an initial solution S_0 for a given graph $G = (V, E)$ and k, we adopt the efficient approach for MCP proposed in [Jiang *et al.*, 2016] and extend it for MKP. The procedure sorts the vertices of V in the *degeneracy ordering* $v_1 < v_2 < \cdots < v_n$ in such a way that v_i is the vertex with minimal degree $deg^*(v_i)$ in $G[\{v_i, \ldots, v_n\}]$. According to the definition of k-plex, when there is a vertex v_i satisfying $deg^*(v_i) \geq n - i + 1 - k$, the remaining vertices $v_i, v_{i+1}, \ldots, v_n$ forms a k-plex. So, S_0 is $\{v_i, v_{i+1}, \ldots, v_n\}$.

Next, the procedure reduces the graph with the initial solution S_0. Let $lb = |S_0|$, for searching for $\omega_k(G)$, the vertex v with a degree $deg_G(v)$ satisfying $deg_G(v) + k < lb + 1$ can be removed from G safely, because $deg_G(v) + k$ is an upper bound of k-plex containing v. In fact, let v_p be the first vertex with $deg^*(v_p) \geq lb + 1 - k$ (starting with 1), then all the vertices smaller than v_p in the ordering can be removed from G. This rule has been widely used in [Xiao *et al.*, 2017; Gao *et al.*, 2018; Zhou *et al.*, 2021] for preprocessing.

To further reduce the input graph, we exploit Property 3 to find out more vertices that can be removed. Formally, we define an *'unsupported'* notion for two adjacent vertices w.r.t. a given lower bound lb of k-plex.

Definition 3. *Given a lower bound lb, two adjacent vertices u, v are* 'unsupported' *each other w.r.t lb, if they cannot occur simultaneously in a k-plex of size greater than or equal to lb.*

We use $usp^{lb}_G(v)$ to denote the set of neighbors of v that are *'unsupported'* to v w.r.t. the lower bound lb. The following two properties are natural extensions of Property 3 and 2.

Property 6. *Two adjacent vertices u and v are 'unsupported' each other w.r.t. a lower bound lb, if $\Theta_G(u, v) < lb - 2k$.*

Property 7. *If there exists a k-plex of size lb containing vertex v in G, then $|usp^{lb}_G(v)| \leq deg_G(v) + k - lb$.*

Since $deg_G(v) - |usp^{lb}_G(v)|$ is the largest number of neighbors of v that can occur in a k-plex S of size lb and containing v, then $deg_G(v) - |usp^{lb}_G(v)| + k$ is an upper bound of $|S|$. So, we have $deg_G(v) - |usp^{lb}_G(v)| + k \geq lb$ and then Property 7 can be derived.

After the procedure obtaining S_0, the next target of the algorithm is to find a k-plex of size greater than $|S_0|$. Let $lb' = |S_0| + 1$, we can compute $usp^{lb'}_G(v)$ for each v with Property 6, and then remove the vertex v having $|usp^{lb'}_G(v)| > deg_G(v) + k - lb'$ according to Property 7. Note that the cardinality of $usp^{lb'}_G(\cdot)$ of remaining vertices could be reduced after some vertices removed from G, then the reduction rule can be applied iteratively till no more vertex can be removed.

With a mark array, the computation of $\Theta_G(u, v)$ can be done in $O(|V|)$. So the time complexity of the second reduction phase is $O(|V| \cdot |E| \cdot r)$, where r is the number of rounds of the reduction step carrying out.

5.2 The Branch-and-Bound Search Procedure

Algorithm 2 depicts the BnB search procedure called in Algorithm 1 line 3, which explores the search space formed by the candidate set P to search for a maximum k-plex of size greater than the incumbent S^*.

Before exploring search space formed by the candidate set P, the algorithm first calls function $Partition(S, P, k, |S^*|)$ to partition P into B and $P \setminus B$ in such a way that S could not be extended to a solution of size greater than $|S^*|$ with vertices in $P \setminus B$, i.e, $\omega_k(G[S \cup (P \setminus B)]) \leq |S^*|$. Consequently, B is the set of branching vertices. The algorithm branches on every vertices of B in the degeneracy vertex ordering of G.

For each branching vertex $u_i \in B$ ($i = |B|$ to 1), let Q be a copy of the current candidate set $P' = P \setminus B$, Algorithm 2 employs two pruning steps to reduce Q. First, the algorithm excludes those vertices whose number of non-neighbors in $S \cup \{u_i\}$ reaches k. Second, if the cardinality of S is smaller than 3, the algorithm then exploits Property 3 to further exclude vertices in Q. We call the second step *Pro3-based* pruning. This step is restricted to the search tree nodes of level smaller than three in our implementation, because its time complexity is $O(|V|^2)$, which is not economical enough to be applied at every search tree node, especially for dense graphs. After reducing Q, Algorithm 2 searches for a maximum k-plex in the search space formed by Q recursively.

Algorithm 2 BnBSearch(G, S, P, S^*,k)

Input: A graph G=(V,E), a growing k-plex S, the candidate set P and the incumbent solution S^* and k .
Output: the best solution S^* in G.

```
1: if $P$ is empty then
2:     return $S^*$;
3: end if
4: $B \leftarrow$Partition($S$, $P$, $k$, $|S^*|$);
5: if $B$ is empty then
6:     return $S^*$;
7: end if
8: Let $B = \{u_1, u_2, \ldots, u_{|B|}\}$ and $P' = P \setminus B$, $u_1 < u_2 < \cdots < u_{|B|}$ w.r.t. the degeneracy vertex ordering of $G$.
9: for $i = |B|$ to 1 do
10:     Let $Q = P'$, remove vertices in $Q$ whose number of non-neighbors in $S \cup \{u_i\}$ is equal to $k$.
11:     if $|S| < 3$ then
12:         compute $\Theta_{G[Q \cup S]}(u_i, v)$ for each $v \in Q$.
13:         remove $v$ from $Q$ if $\Theta_{G[Q \cup S]}(u_i, v) + 2k - \gamma \leq |S^*|$, $\gamma$ is 0 if $(u_i, v) \in E$; Otherwise, $\gamma$ is 2.
14:     end if
15:     $S' \leftarrow$ BnBSearch($G$, $S \cup \{u_i\}$, $Q$, $S^*$,$k$);
16:     if $|S'| > |S^*|$ then
17:         $S^* \leftarrow S'$;
18:     end if
19:     $P' \leftarrow P' \cup \{u_i\}$;
20: end for
21: return $S^*$;
```

Algorithm 3 describes the function *Partition*, which employs the new upper bound PUB to partition P. At first, the branching set B is set with P. Then, the algorithm constructs a set π_i for each $v_i \in S$ one by one, where π_i is the set of non-neighbors of v_i in B. The algorithm maintains an upper bound $ub = \sum_{j=1}^{i} \min\{k - 1 - \delta_j, |\pi_j|\}$, which is the largest number of vertices that can be inserted into S in $\bigcup_{j=1}^{i} \pi_j$. If $|S| + ub \leq lb$, then the vertices in π_i are removed from B and the algorithm begins to construct the next π_{i+1}; Otherwise, the algorithm returns the set B. If all the π_i ($i \neq 0$) have been constructed and $|S| + ub$ is still smaller than lb, then the algorithm removes $lb - |S| - ub$ more vertices from B before returning B, if B is not empty.

In Algorithm 3, $v_1 < v_2 < \cdots < v_q$ is the ordering in which they are inserted into S. We use this ordering to construct the partition since we are not yet aware that any other

Algorithm 3 Partition(S,P,k, lb), algorithm to partition P

Input: The growing k-plex $S = \{v_1, v_2, \dots, v_q\}$, the candidate set P , the k value and the lower bound lb
Output: A branching set B

```
1: Let $B = P$, $ub = 0$ and $(\delta_1, \dots, \delta_q)$ be $\Delta$ array of $S$;
2: for $i$=1 to $q$ do
3:     create a $\pi_i = \emptyset$;
4:     for each $u \in B$ do
5:         if $u$ is not adjacent to $v_i$ then
6:             $\pi_i \leftarrow \pi_i \cup \{u\}$;
7:         end if
8:     end for
9:     $ub \leftarrow ub + \min\{k - 1 - \delta_i, |\pi_i|\}$;
10:    if $|S| + ub \leq lb$ then
11:        $B \leftarrow B \setminus \pi_i$;
12:    else
13:        return $B$;
14:    end if
15: end for
16: if $|S| + ub < lb$ and $B \neq \emptyset$ then
17:     remove $lb - |S| - ub$ vertices from $B$;
18: end if
19: return $B$;
```

specialized ordering could result in a better partitioning. We use a global array to maintain each δ_i value incrementally. The time complexity of Algorithm 3 is $O(|S| \times |P|)$.

Note that PUB depends crucially on the growing solution S. When the cardinality of S is small, the number of π_i that can be constructed in Algorithm 3 is small, then the pruning ability of Algorithm 3 is limited. That's the rationality we employ Pro3-based pruning step only at the search tree nodes of level smaller than three. As the growing of S, the pruning ability of PUB increases. So, Algorithm 2 switches off the costly Pro3-based pruning step when $|S|$ is greater than two.

6 Experimental Results

We conducted experiments to evaluate our algorithm KpLeX and the bound PUB. KpLeX was compared with three state-of-the-art exact algorithms, BS[1] [Xiao *et al.*, 2017], BnBK[2] [Gao *et al.*, 2018] and Maplex[3] [Zhou *et al.*, 2021].

KpLeX[4] was implemented in C++ and compiled using GNU g++ -O3. All the experiments were performed on Intel Xeon CPUs E5-2680 v4@2.40GHz under Linux with 128GB of memory. The experiments were conducted on the following four datasets using a cutoff time of 3600 seconds for each tested instance and five different k values, 2 to 6.

2nd DIMACS graphs. 80 graphs containing up to 4000 vertices with densities ranging from 0.03 to 0.99.[5] The dataset are widely used to evaluate MCP and MKP.

Real-world massive graphs. 139 real-world sparse graphs from the Network Data Repository [Rossi and Ahmed, 2015]. The dataset[6] were used to evaluate BS, BnBK [Gao *et al.*, 2018] and Maplex [Zhou *et al.*, 2021].

10th DIMACS graphs. 82 graphs containing up to 2×10^7 vertices[7]. The dataset were introduced in 10th DIMACS challenge and also widely used to evaluate MCP and MKP.

Random generated graphs. 120 random graphs of fixed vertex number of 1000 generated with six different densities d: 0.05, 0.10, 0.15, 0.20, 0.25 and 0.30. We generate 20 graphs for each density d in such a way that two vertices are adjacent with probability d.

Since the original BS implementation cannot deal with massive graphs, we integrate a graph reduction procedure used in KpLeX into BS when testing massive graphs.

6.1 Comparison of Total Performance

We first present the total numbers of solved instances of the four algorithms over the four datasets in Figure 2. The x axis is the k value (2 to 6) and y axis is the number of solved instances. We can see that KpLeX shows superiority over the other three algorithms for every tested dataset at almost every k value. For example, KpLeX can solve 27 the 2nd DIMACS instances at $k = 2$, which is 1.2, 1.5 and 1.8 times of the number of instances that Maplex(22), BnBK (18) and BS (15) can solve, respectively. KpLeX performs much better than BS and BnBK on random graphs. It can solve all the 120 instances at k=2 and 3 and 80 instances at k=4, but BnBk (BS) can solve only 42 (40), 20 (0) and 20 (0) at the three points. Maplex has a good performance at k=2, but it declines dramatically with the increasing of k value.

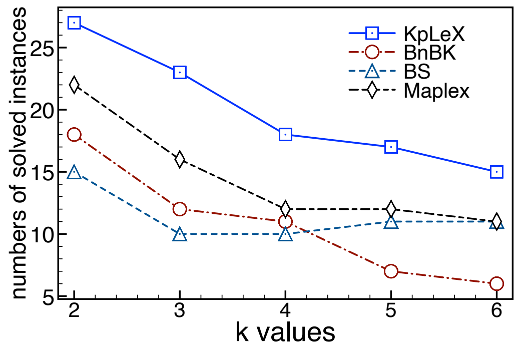
In general, our new algorithm KpLeX outperforms BS, BnBK and Maplex significantly over the four tested datasets. Especially, KpLeX shows a better robustness than the three compared algorithms as the k value increases.

6.2 Comparison of Graph Reduction

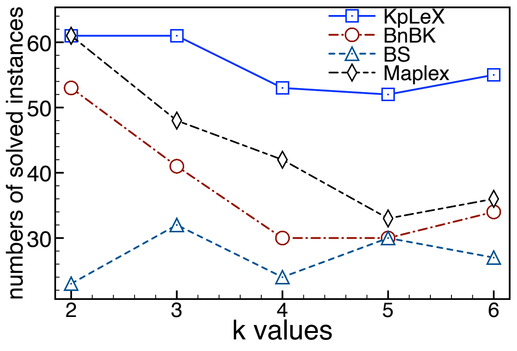
Before investigating the influence of the new upper bound, we compare the effect of the graph reduction of KpLeX with BnBK (According to [Zhou *et al.*, 2021], the effect of the graph reduction of BnBK and Maplex is comparable). Note that existing graph reduction methods can only work for massive sparse graphs. We make the comparison only on real-world dataset. We select 23 graphs with more than 5×10^5 of vertices from the 139 graphs and compare the number of vertices being reduced in the preprocessing phases of the two algorithms using k=2. Table 1 shows the comparison result.

It is easy to see that, except for the graph *sc-ldoor*, almost 99% vertices of tested graphs can be removed by the two algorithms in their preprocessing phases and the qualities of the initial solutions S_0 and the rates of vertex reduction rt, computed as $(|V| - |V'|)/|V|$, of the two algorithms are nearly equal. The comparison shows that the performance differences between KpLeX and BnBK (Maplex) mainly come from the searching components of the three algorithms, rather than from the graph reduction in preprocessing.

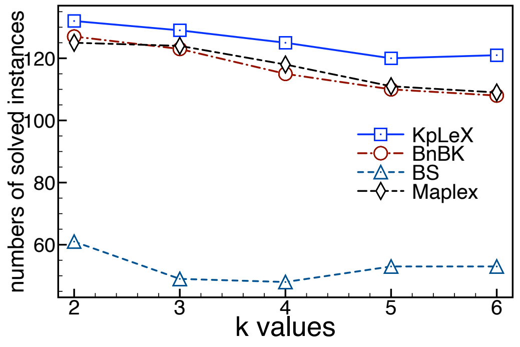
[1]https://github.com/Lweb/KPLEX
[2]https://github.com/JimNenu/codekplex
[3]https://github.com/ini111/Maplex
[4]Published at https://github.com/huajiang-ynu/kplex
[5]http://cs.hbg.psu.edu/txn131/clique.html
[6]http://lcs.ios.ac.cn/ caisw/Resource/realworld%20graphs.tar.gz
[7]http://networkrepository.com/dimacs10.php



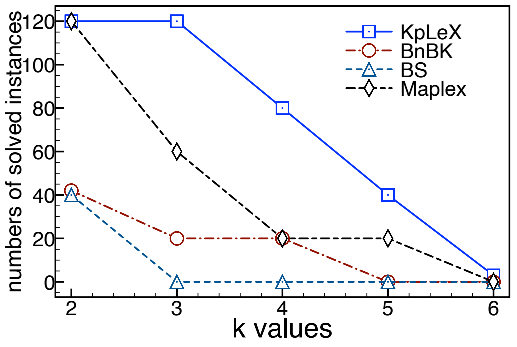
(a) 2nd DIMACS Graphs



(b) 10th DIMACS Graphs



(c) Real-world Graphs



(d) Random Graphs

Figure 2: The total numbers of solved instances of KpLeX, BnBk, BS and Maplex using $k = 2$ to 6. The cutoff time is 3600 seconds.

Instance	$\|V\|$	BnBK			KpLeX		
		$\|S_0\|$	$\|V'\|$	rt	$\|S_0\|$	$\|V'\|$	rt
ca-coauthors-dblp	540486	337	0	1.000	337	0	1.000
ca-hollywood-2009	1069126	2209	0	1.000	2209	0	1.000
inf-roadNet-CA	1957027	5	0	1.000	5	0	1.000
inf-roadNet-PA	1087562	5	0	1.000	5	0	1.000
inf-road-usa	23947347	5	0	1.000	5	0	1.000
rt-retweet-crawl	1112702	14	0	1.000	12	51	0.999
scc_retweet-crawl	1131801	21	0	1.000	21	0	1.000
sc-ldoor	952203	21	882715	0.073	21	882399	0.073
soc-delicious	536108	18	181	0.999	18	183	0.999
soc-digg	770799	22	11885	0.985	22	11836	0.985
socfb-A-anon	3097165	24	2239	0.999	24	2155	0.999
socfb-B-anon	2937612	15	37564	0.987	15	35907	0.988
socfb-uci-uni	58790782	9	0	1.000	7	805	0.999
soc-flickr	513969	64	2251	0.996	64	2255	0.996
soc-flixster	2523386	36	282	0.999	36	283	0.999
soc-FourSquare	639014	32	9974	0.984	31	11326	0.982
soc-lastfm	1191805	16	1254	0.999	16	1250	0.999
soc-livejournal	4033137	214	0	1.000	214	0	1.000
soc-pokec	1632803	21	838	0.999	18	2359	0.999
soc-youtube-snap	1134890	16	1034	0.999	16	1004	0.999
tech-as-skitter	1694616	62	422	0.999	62	423	0.999
web-it-2004	509338	432	0	1.000	432	0	1.000
web-wikipedia2009	1864433	9	5824	0.997	9	5724	0.997

Table 1: The comparison of graph reduction of KpLeX and BnBK with k=2. $|V|$ is the number of vertices of original graphs, S_0 denotes the initial solution the two algorithms found in their preprocessing phases. V' is the vertex set of reduced graph and rt denotes the rate of vertex being reduced, computed as $(|V| - |V'|)/|V|$.

6.3 The Effect of the Upper Bound

To investigate the efficiency of the new upper bound for reducing search space in KpLeX, we conducted another experiment and compared KpLeX with the following two variants.

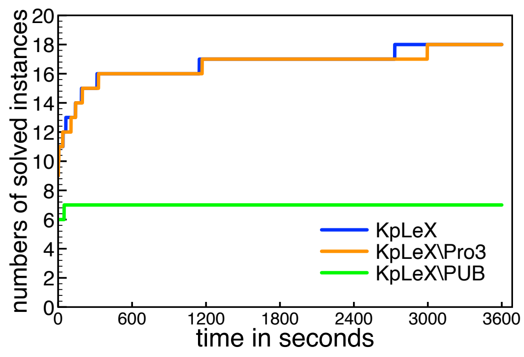
KpLeX\Pro3. It is KpLeX but the Pro3-based reduction step (line 11 to 14 in Algorithm 2) is disabled.

KpLeX\PUB. It is KpLeX but uses a simple method, instead of using PUB, to generate the branching set B, i.e., B is P but the last $lb - |S|$ vertices, *w.r.t.* the degeneracy ordering, are removed in Algorithm 3.

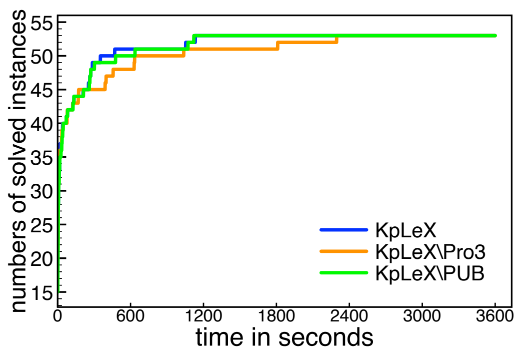
The comparison of KpLeX with the two variants can help us evaluate the effectiveness of the PUB bound and the Pro3-based reduction in reducing search space in KpLeX. We use k=4, the median of $[2, 6]$, to evaluate KpLeX and the two variants. Figure 3 shows the cumulative numbers of solved instances over the four datasets within a cutoff time 3600s.

From Figure 3, we can see that the variant KpLeX\Pro3 performs nearly the same with KpLeX, only showing a slight decline of performance on the 10th DIMACS and real-world graphs, which shows that the Pro3-based reduction step can enhance the performance of KpLeX but limitedly. Whilst, the performance of KpLeX\PUB declines dramatically on the three datasets out of the four, showing that the PUB bound is crucial to the performance of KpLeX.

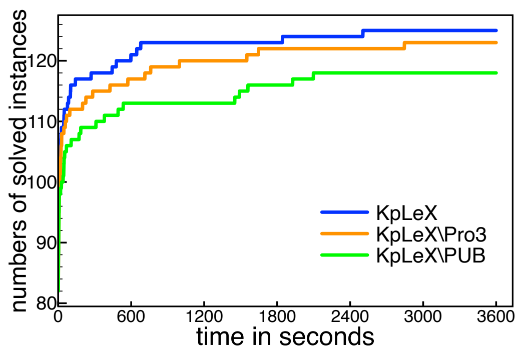
To make a deep insight, we select 4 representative instances from the four tested datasets and analyse the search tree sizes and the running times of them at k=2, 4 and 6. The



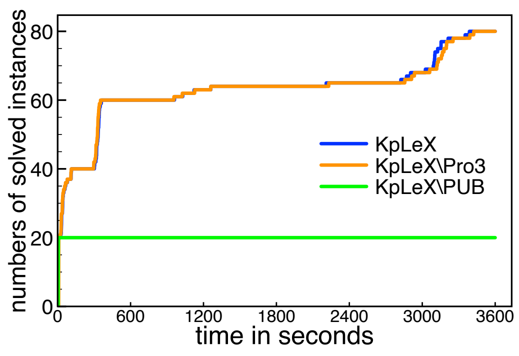
(a) 2nd DIMACS Graphs



(b) 10th DIMACS Graphs



(c) Real-world Graphs



(d) Random Graphs

Figure 3: Cumulative numbers of solved instances of KpLeX and two variants at $k = 4$.

cutoff times for KpLeX\PUB and KpLeX\Pro3 are 10000 seconds.

Table 2 shows the result of the comparison. The results show that the search tree sizes of KpLeX\PUB increase dramatically for the four instances, as well as the running times. KpLeX\PUB cannot solve three of them when k=6 using 10000s. Nevertheless, the search tree sizes of KpLeX\Pro3 increase less dramatically and only one instance *consph* cannot be solved within 10000s at k=6.

In general, the new upper bound PUB is very effective to reduce the number of branches and is crucial to the performance of the new algorithm KpLeX. And we also note that the Pro3-based reduction step could be effective for solving the real-world sparse graphs.

Instance	k	KpLeX		KpLeX\PUB		KpLeX\Pro3	
		$tree$	$time$	$tree$	$time$	$tree$	$time$
p_hat300-1	2	**0.69**	0.40	17.38	1.08	0.70	**0.20**
	4	**25.4**	5.77	99926	7621	25.58	**5.05**
	6	2250	528.1	-	-	**2219**	**487.1**
consph	2	**0.93**	303.8	2.66	**266.8**	176.8	287.1
	4	**121.7**	**351.3**	4985	659.2	3315	5557
	6	**299.2**	**400.4**	2392	536.4	-	-
socfb-A-anon	2	**16.03**	38.15	139.4	**32.82**	24.74	36.37
	4	**372.9**	**141.8**	27937	2102	489.4	178.9
	6	**5761**	**1277**	-	-	6482	1647
1000_0.05_20	2	**0.02**	**0.08**	0.03	0.10	0.23	0.15
	4	**11.32**	**1.10**	122.2	4.52	44.9	8.08
	6	**14102**	**1806**	-	-	18758	3449

Table 2: The comparison of KpLeX with two variants. $tree$ is the search tree size in 10^5 and $time$ is the running time in seconds.

7 Conclusions

We proposed a novel upper bound for MKP. The bound partitions the candidate set P w.r.t. the growing solution S. The advantage of the bound is that the close relationship between the growing solution S and the candidate set P is exploited. We implemented a new BnB algorithm for MKP that incorporates the upper bound to reduce the number of branches. The reported experiments show that the new upper bound is very effective in reducing the search space, and that the algorithm KpLeX outperforms relevant exact algorithms on DIMACS graphs, real-world massive graphs and random graphs.

Acknowledgments

This work is supported in part by NSFC under grant 61863036, Key Area Research Program of Yunnan Province under grant 202001BB050076, Education Department Funding of Yunnan Province under grant 2021J0006, the Shenzhen Science and Technology Innovation Commission under grant JCYJ20180508162601910, the National Key R&D Program of China under grant 2020YFB1313300, the Funding from the Shenzhen Institute of Artificial Intelligence and Robotics for Society under grant 2019-INT003 and by the MatriCS platform of University of Picardie Jules Verne.

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.*, 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.

[Gschwind *et al.*, 2018] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. Maximum weight relaxed cliques and russian doll search revisited. *Discret. Appl. Math.*, 234:131–138, 2018.

[Jiang *et al.*, 2016] Hua Jiang, Chu Min Li, and Felip Manyà. Combining efficient preprocessing and incremental MaxSAT reasoning for MaxClique in large graphs. In *Proceedings of the 22nd European Conference on Artificial Intelligence, ECAI, The Hague, The Netherlands*, pages 939–947, 2016.

[Jiang *et al.*, 2017] Hua Jiang, Chu Min Li, and Felip Manyà. An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, California, USA*, pages 830–838, 2017.

[Li and Quan, 2010] C.M. Li and Z. Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, Usa, July*, pages 128–133, 2010.

[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.

[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.

[Tomita *et al.*, 2010] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *Proceedings of the 4th International Workshop on Algorithms and Computation, WALCOM, Dhaka, Bangladesh*, pages 191–203, 2010.

[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.