Improving Maximum k-Plex Solver via Second-Order Reduction and Graph Color Bounding

Yi Zhou[1], **Shan Hu** [1], **Mingyu Xiao** [1], **Zhang-Hua Fu** [2,3*]

[1]University of Electronic Science and Technology of China,
[2]Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen, China,
[3]Institute of Robotics and Intelligent Manufacturing, The Chinese University of Hong Kong, Shenzhen, China
zhou.yi@uestc.edu.cn, hu.shan@std.uestc.edu.cn, myxiao@gmail.com, fuzhanghua@cuhk.edu.cn

Abstract

In a graph, a k-plex is a vertex set in which every vertex is not adjacent to at most k vertices of this set. The maximum k-plex problem, which asks for the largest k-plex from the given graph, is a key primitive in a variety of real-world applications like community detection and so on. In the paper, we develop an exact algorithm, Maplex, for solving this problem in real world graphs practically. Based on the existing first-order and the novel second-order reduction rules, we design a powerful preprocessing method which efficiently removes redundant vertices and edges for Maplex. Also, the graph color heuristic is widely used for overestimating the maximum clique of a graph. For the first time, we generalize this technique for bounding the size of maximum k-plex in Maplex. Experiments are carried out to compare our algorithm with other state-of-the-art solvers on a wide range of publicly available graphs. Maplex outperforms all other algorithms on large real world graphs and is competitive with existing solvers on artificial dense graphs. Finally, we shed light on the effectiveness of each key component of Maplex.

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 algorithm 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, Youssef, and Butenko 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 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 (MPLEX) asks for the largest k-plex from the given graph. Algorithms for the MPLEX are also important tools in the analysis of complex networks (Pattillo, Youssef, and Butenko 2013), especially in the community detection problem (Conte et al. 2018; Zhou et al. 2020; Zhu, Chen, and Zeng 2020).

It is clear that MPLEX is equal to MCP when $k = 1$ and thus an NP-hard problem. Indeed, for any $k > 1$, MPLEX is still NP-hard (Lewis and Yannakakis 1980; Balasundaram, Butenko, and Hicks 2011). So far, it is known that MPLEX can be solved in time $O(\gamma^n)$ where n is the number of vertices in the given graph and γ is a value related to k but always slightly smaller than 2 (Xiao et al. 2017). Unless P=NP, there cannot be any polynomial time algorithm that approximates the maximum k-plex within a factor better than $O(n^\epsilon)$, for any $\epsilon > 0$ (Lund and Yannakakis 1993).

Despite the fact that MPLEX is theoretically challenging, there exists a considerable number of algorithms for solving MPLEX practically. In the literature, we mainly distinguish two types of algorithms for MPLEX, the heuristic algorithms and exact algorithms. The heuristic algorithms are able to quickly provide a lower bound, but cannot guarantee the optimality of their solutions. Representative heuristic approaches for MPLEX mainly use stochastic local search (Zhou and Hao 2017; Chen et al. 2020) or the GRASP method (Miao and Balasundaram 2017). Exact algorithms, in contrary, ensure the optimality of their solution. Existing exact algorithms for MPLEX include integer programming methods (Balasundaram, Butenko, and Hicks 2011), branch-and-bound algorithms (McClosky and Hicks 2012; Moser, Niedermeier, and Sorge 2012; Xiao et al. 2017; Gao et al. 2018; Wu et al. 2019) and Russian Doll Search (Trukhanov et al. 2013; Shirokikh 2013; Gschwind, Irnich, and Podlinski 2018).

In the paper, we investigate MPLEX by developing a new practical solver, *Maplex*. Maplex is an exact algorithm which is time-efficient and scalable for the ubiquitous large real world graphs. Compared with the existing exact solvers, Maplex has two notable features.

First, due to the sheer sizes of real world graphs, it is computationally expensive to directly manipulate them in memory. Hence, existing algorithms like (Trukhanov et al. 2013; Zhou and Hao 2017; Gschwind, Irnich, and Podlinski 2018) used *Peel* to reduce the graph in preprocessing. Peel is a linear-time algorithm which reduces low-degree vertices without missing the optimality. However, in many cases, the graphs after using Peel are still too large to fit the memory, e.g., the facebook network socfb-A-anon has around 390144

*Corresponding author.

vertices after Peel when $k = 2$. In this paper, we introduce a stronger preprocessing procedure based on the new second-order reduction rule for Maplex. Our preprocessing can remove not only low-degree vertices but also unnecessary edges in an alternative manner. For graph socfb-A-anon with $k = 2$, the number of vertices can degrades to only 2155 by using this new approach. It is worth mentioning that Gao *et al.* (2018) introduced another strong inference algorithm for preprocessing the input graph. We show that our preprocessing reduces the input graph into roughly the same level of scale as their method, but runs 4x to 10x faster.

In order to obtain the optimal solution after preprocessing, we use a branch-and-bound algorithm to search in the remaining graph. For the first time, we apply the *graph color* heuristic for upper-bounding in the branch-and-bound. The Graph Color Problem (GCP) for a given graph G asks for the minimum number of colors necessary to color the vertices of G such that no two adjacent vertices share the same color. It is known that the number of colors of G is an upper bound of the size of maximum 1-plex of G and the graph color heuristics are used in some successful MCP algorithms like (Li and Quan 2010; Tomita and Seki 2003). Inspired by this, we build connections between GCP and MPLEX for any $k \geq 1$ and borrow the successful graph color heuristic in MCP algorithms for bounding the maximum k-plex. We empirically demonstrate that the branch-and-bound with graph color based bounding strategy runs suitably well.

The remainder of the paper is organized as follows. We introduce some necessary notations and backgrounds in the next section. Then, we present the general framework of Maplex. Afterwards, we elaborate the key techniques of Maplex including the reduction rules and the bounding methods. We also show some implementation details. Lastly, we empirically investigate the behaviour of Maplex and compare it with the best known algorithms on different types of benchmark graphs.

Preliminary

Let $G = (V, E)$ be a given graph with vertex set V and edge set E. For a vertex $v \in V$, $N_G(v)$ denotes the set of adjacent vertices of v in G. For two vertices u, v of G, $\Delta_G(u, v)$ denotes the set of common adjacent vertices of v and u in G, i.e., $\Delta_G(u, v) = N_G(u) \cap N_G(v)$.

A vertex set $C \subseteq V$ is a clique if the vertices of C are pairwise adjacent in G. A *triangle* is a clique of size 3. In contrary, a vertex set $I \subseteq V$ is an *independent set* if any two vertices of I are not adjacent in G. A vertex set $P \subseteq V$ is a k-plex if for any vertex $v \in P$, $|N_G(v) \cap P| \geq |P| - k$, k being a positive integer. So, a 1-plex of G is also a clique of G.

As mentioned, the maximum k-plex problem (MPLEX) asks for a largest k-plex in the given graph. Note that a k-plex is *maximal* in G if it is not a subset of any other k-plex in G.

An important property of k-plex is the *hereditary property*, which says that if a vertex set P is a k-plex, then any subset of P is also a k-plex. The hereditary property implies that a k-plex P is maximal if no other vertex can be added to P to form a larger k-plex.

General Framework

We show the framework of Maplex in Alg. 1. In general, Maplex consists of three parts, a fast heuristic search, a preprocessing component and a branch-and-bound search.

Heuristic Search As shown in line 3 of Alg. 1, heuristic algorithm HeuristicSolution(G, k) provides a lower bound, lb, for the input graph G and value k. Note that the solution set, i.e., the best-known k-plex, is recorded whenever lb updates though we do not explicitly point it out in the algorithm. In (Trukhanov et al. 2013; Gschwind, Irnich, and Podlinski 2018), the maximum clique size of the input graph is simply used as the lower bound solution. In (Gao et al. 2018), the heuristic algorithm is extended from the MCP algorithm in (Jiang, Li, and Manya 2017). In Maplex, we use a problem-specific heuristic which finds the longest suffix of the so-called *degeneracy ordering* that is a k-plex. This heuristic runs in time $O(m + lb^2 n)$ where n and m represent the number of vertices and edges, respectively. Detailed procedure is described in the supplementary material.

Preprocessing After obtaining a solution of size lb, we are only interested in finding out the solution of size larger than lb, if it exists. Hence, we preprocess the input graphs by removing the vertices and edges that do not belong to the solutions, as shown in line 4 of Alg. 1. The step is critical for future exact search because the cost of visiting a smaller graph is much cheaper than visiting the massive graph. Sometimes, the preprocessing directly reduces the input graph into an empty graph, indicating that lb is the already the optimal size. For the rest of the paper, let us call the graph after preprocessing a *kernel graph*.

Branch-and-bound In line 5 of Alg. 1, we use branch-and-bound to find the best solution in the kernel graph. The branch-and-bound search is essentially a depth-first tree search algorithm which recursively calls subroutine BranchBound(G, k, P, C) to solve the following subproblem.

Given a graph $G = (V, E)$, a growing k-plex $P \subseteq V$ and a candidate set $C \subseteq V$ (P and C are disjoint), find the largest k-plex which is a superset of P from $G[P \cup C]$.

BranchBound(G, k, P, C) first updates lb if the growing k-plex is larger than the current lb value, as in lines 8-9 of Alg. 1. In line 10, BranchBound(G, k, P, C) estimates an upper bound of the current subproblem. If the upper bound is not larger than lb, the search of the current subproblem is discarded. Otherwise, BranchBound(G, k, P, C) considers every possibility of moving a vertex $u \in C$ to P. In line 14, BranchBound(G, k, P, C) removes *unfruitful candidate vertices* which have no possibility of being a member of a solution that is better than lb.

In the following, we introduce new preprocessing and bounding techniques for Maplex.

Stronger Reduction Rules for Preprocessing

In this section, we study the key reduction rules that are used for the preprocessing. Notably, we propose a new *second-order reduction* rule which substantially improves the preprocessing.

Algorithm 1: The algorithmic framework of Maplex

```
Maplex(G, k)
begin
    lb ← HeuristicSolution(G, k)
    G′ ← Preporcessing(G, k, lb)
    BranchBound(G′, k, ∅, V(G′))        ▷ V(G′) is the vertex set of G′
BranchBound(G, k, P, C)
begin
    if |P| > lb then
        lb ← |P|
    if UpperBound(G, k, P, C) ≤ lb then
        return
    while C is not empty do
        Pick and remove a branching vertex u from C
        C′ ← Reducing unfruitful vertices from C
        BranchBound(G, k, P ∪ {u}, C′)
```

First-order reduction

Proposition 1 (First-order reduction)**.** *Given a graph $G = (V, E)$, a vertex $v \in V$. If $|N_G(v)| \le lb - k$, then v is not in a k-plex larger than lb.*

A widely used preprocessing procedure named **Peel** is based on the first-order reduction rule. Peel recursively removes vertices with degree at most $lb - k$ until there is no such vertex in the remaining graph. Peel is effective because most vertices in real world graphs have low degrees by power-law distribution. Meanwhile, Peel can be implemented in linear time $O(m)$ where m represents the number of edges of the original graph. Due to the simplicity and effectiveness, Peel is used in solving MPLEX in (Trukhanov et al. 2013; Gschwind, Irnich, and Podlinski 2018; Zhou and Hao 2017; Chen et al. 2020).

Second-order reduction

Proposition 2 (Second-order reduction)**.** *Given a graph $G = (V, E)$, two vertices $u, v \in V$. If $(u, v) \in E$ and $|\Delta_G(u, v)| \le lb - 2k$, then u, v are not in a k-plex larger than lb at the same time. If $(u, v) \notin E$ and $|\Delta_G(u, v)| \le lb - 2k + 2$, then u, v are not in a k-plex larger than lb at the same time.*

Proof. Assume that $(u, v) \in E$ satisfies $|\Delta_G(u, v)| \le lb - 2k$ but both u, v belong to a k-plex S where $|S| > lb$. By the definition of k-plex, there are at most $k - 1$ vertices that are not adjacent to u (or v) in $S \setminus \{u, v\}$. Hence, there are at most $2k - 2$ vertices in S that do not belong to $\Delta_G(u, v)$, i.e., $|S| - 2 - |\Delta_G(u, v) \cap S| \le 2k - 2$, indicating that $|\Delta_G(u, v) \cap S| \ge |S| - 2k > lb - 2k$, which contradicts the assumption that $|\Delta_G(u, v)| \le lb - 2k$. Likewise, we can also obtain a contradiction for the second case where $(u, v) \notin E$ and $|\Delta_G(u, v)| \le lb - 2k + 2$. $\square$

Algorithm 2: The preprocessing in Maplex

```
1  Preprocess(G = (V, E), k, lb) begin
2      Let q be an empty queue
3      Push vertices v ∈ V that |N_G(v)| ≤ lb − k into q
4      while true do
5          while q is not empty do
6              Pop a vertex v from q
7              Remove v and its incident edges from G
8              for u ∈ N_G(v) and N_G(u) ≤ lb − k do
9                  Push u into q
10         Listing triangles in G by compact-forward
           algorithm in (Latapy 2008), counting
           |Δ_G(u, v)| for each edge (u, v) ∈ E
11         for (u, v) ∈ E(G) do
12             if |Δ_G(u, v)| ≤ lb − 2k then
13                 Remove (u, v) from G
14                 if |N_G(u)| ≤ lb − k then
15                     Push u into q
16                 if |N_G(v)| ≤ lb − k then
17                     Push v into q
18         if q is empty then
19             Break
20     return G
```

An enhanced preprocessing With both first- and second-order reduction rules, we enhance Peel by jointly removing extra vertices and edges.

The idea is straightforward. First, we remove vertices of degree at most $lb - k$, the same as in Peel. When there are no such vertex, we identify edges (u, v) such that $|\Delta_G(u, v)| \le lb - 2k$ and remove them. Whenever an edge (u, v) has been removed from G, $|N_G(u)|$ and $|N_G(v)|$ are also decreased by one, making v or u possibly removable again. Hence, the preprocessing procedure alternatively removes vertices by the first-order reduction rule and edges by the second-order reduction rule until there is no reducible vertex and edge.

Alg. 2 implements this idea. Because $|\Delta_G(u, v)|$ is equal to the number of triangles that involve edge (u, v), we compute $|\Delta_G(u, v)|$ for all edges by listing all the triangles in the graph, as shown in line 10. The problem of listing all the triangles without repetition is called *Triangle Listing Problem*, which is well-studied. We choose the optimized *compact-forward* triangle listing algorithm proposed in (Latapy 2008). The time complexity of triangle listing is $O(m^{1.5})$, so the whole running time of our Preprocessing is $O(lm^{1.5})$ where l is the number of out-most loop. Normally $l \ll n$.

Other higher-order reduction rules It is possible to jointly consider even more vertices at the same time. For example, three vertices in a triangle are not in a k-plex larger than lb simultaneously if their common adjacent vertices are no more than $lb - 3k$. However, higher-order reduction rules are not easy to be used in an efficient manner. Another preprocessing technique by Gao *et al*. (2018) used the sophisticate *subgraph reduction rule*. We are not aware of the worst-case running time of their approach, but as shown in the experiments, our method removes approximately similar number as their preprocessing but runs much faster.

The Bounding Techniques

The bounding techniques overestimate an upper bound of the optimal solution in $G[P \cup C]$ and prune the search if the upper bound is not larger than the lower bound. We study a new graph color based bounding technique for MPLEX.

Graph color bound A *coloring* of a graph G is a partition of the vertex set such that each vertex set in the partition is an independent set in G. The number of independent sets in a coloring is the upper bound of the maximum clique size of G. We generalize this bound to MPLEX.

Proposition 3. *Given a graph $G = (V, E)$, if V can be partitioned into c disjoint independent sets $I_1, ..., I_c$, then $\sum_{i=1}^{c} \min\{|I_i|, k\}$ is the upper bound of the size of maximum k-plex in G, a.k.a. color-bound.*

We leave the proofs of Prop. 3 and all the remaining propositions in the supplementary material.

The problem of finding the minimum color-bound is NP-hard (it is equal to GCP when $k = 1$). In our algorithm, we borrow the fast constructive heuristic coloring procedure from the celebrated MCP algorithm in (Tomita and Seki 2003). The heuristic asks for an initial order of the vertices in the graph. As P must be a subset of the solution, we only need to color the vertices of C. So, let us assume the given order of C is $v_1, ..., v_q$ where q is the size of C. In the first round, a first independent set $I_1 = \{v_1\}$ is initialized. In the jth round where j starts from 2, the heuristic puts v_j into the first independent set such that v_j is not adjacent to any vertex in the set. If such an independent set does not exist, a new independent set is opened and v is inserted in it. When all vertices of C are partitioned, the color-bound plus $|P|$ is the upper bound for the subproblem of BranchBound(G, k, P, C).

Lookahead by color-bound The partition of C after graph color heuristic also provides information for the next subproblem in which a branching vertex $u \in C$ is moved to P.

Proposition 4. *Given a subproblem with a growing k-plex P, a candidate set C, assume $\mathcal{I} = \{I_1, ..., I_c\}$ is a coloring of C. For any vertex $u \in C$, the size of k-plex S that $u \in S$ and $P \subset S$ is bounded by $ub_u = \sum_{i=1, u \notin I_i}^{c} \min(|I_i \cap N_G(u)|, k) + (k - |P \setminus N_G(u)|) + |P|$.*

Prop. 4 suggests a way of evaluating the upper bound for the next subproblem where a branching vertex u is taken from C to P. If $ub_u \leq lb$, the next subproblem has no promising solutions. Given a color partition of C, the computation of ub_u can be finished in $O(|C|)$ time.

Other bounding techniques We remark on other popular bounding techniques. The *core number* of G, denoted by $c(G)$, is the largest k such that every vertex of the graph is in the maximal subgraph whose minimum degree is at least k. Clearly, $c(G) + k$ is an upper bound of the maximum k-plex in G. However, our preprocessing ensures that the core number of a kernel graph is at least $lb - k$.

The linear program (LP) of MPLEX is studied in (Balasundaram, Butenko, and Hicks 2011). The *LP* relaxation often leads to tight upper bound but the computation cost of the LP formulation is too heavy. In the MCP algorithm in (Li and Quan 2010), SAT reasoning is used to tighten the graph color bound for MCP. However, it is not known how to efficiently extend this technique to MPLEX with $k > 2$.

Algorithm 3: The branch-and-bound algorithm in Maplex

```
1  BranchBound(G, k, P, C)
2  begin
3      Partition vertices of C into I_1, ..., I_c by greedy
       coloring heuristic (Tomita and Seki 2003). Note that
       C is an ordered list.
4      ub ← Σ_{j=1}^{c} min(|I_j|, k) + |P|      ▷ Computing color-bound
5      if ub ≤ lb then
6          return
7      Re-sort vertices in C by their color numbers in 𝓘 in
       an increasing order, vertices of the same color
       number preserve their original relative order
8      while C is not empty do
9          Remove the last vertex u from C
10         ub_u = Σ_{i=1,u∉I_i}^{c} min(|I_i ∩ N_G(u)|, k) + (k −
           |P \ N_G(u)|) + |P|                    ▷ lookahead
11         if ub_u ≤ lb then
12             Continue
13         C' ← Reducing unfruitful vertices from C,
           keeping the order of C unchanged
14         BranchBound(G, k, P ∪ {u}, C')
```

Implementation Details

Finally, we give a more detailed description of our branch-and-bound in Alg. 3. There are a number of data structures and implementation details that are important for an efficient implementation of the branch-and-bound.

Vertex ordering and branching heuristic In the input of Alg. 3, the candidate set C is an ordered list as asked by the graph color heuristic in line 3. When BranchBound is called by Maplex initially, the vertices of C are ordered by non-increasing degrees so that the graph color heuristic uses this order to partition the vertices of C at the root node. For a color partition $I_1, \cdots, I_c$ and a vertex $v \in C$, let us call the index of the independent set I_i where v belongs the *color number* of v. When a new partition of C is produced by the heuristic, the vertices of C are reordered so that vertices of smaller color number precede these of larger color number, while vertices of the same color number preserve their original relative order. The last vertex of C, which is a vertex of maximum color number in C, is always selected as branching vertex.

Bitset encoding San Segundo *et al.* (2011) introduce the bitset data structure to encoding the vertex set. For example, the intersecting between $N_G(v)$ and an independent set I_i, i.e., $N_G(v) \cap I_i$, is frequently used in graph color heuristic. By encoding the $N_G(v)$ and I_i as bit sets, the operation becomes a simple "and" operation between the two bitsets.

Reducing unfruitful vertices The second-order reduction can be also used to reduce unfruitful candidate vertices in branch-and-bound. In BranchBound(G, k, P, C), after a branching vertex $u \in C$ moves to P (as in line 14 of Alg. 1), we screen out the unfruitful vertices $v \in C$ if $(u, v) \in E$ and $|\Delta_G(u, v) \cap (P \cup C)| \leq lb - 2k$, or $(u, v) \notin E$ but $|\Delta_G(u, v) \cap (P \cup C)| \leq lb - 2k + 2$.

Experiments

In this section, we carry out experiments to evaluate the proposed algorithm. The codes are written in C++ and compiled by g++ with optimization option '-O3' [1]. All the experiments are conducted on a cluster running CentOS operating system in intel-E5-2695 (2.1GHz, 36 cores) with 8G memory.

For the purpose of comparison, we use three recent baseline algorithms, *BS* (Xiao et al. 2017), *BnBk*(Gao et al. 2018) and *RDS* (Gschwind, Irnich, and Podlinski 2018). These algorithms are more efficient than other older solvers like IPBC (Balasundaram, Butenko, and Hicks 2011), OsterPlex (McClosky and Hicks 2012) and GuidedBranching (Moser, Niedermeier, and Sorge 2012) as reported in the literature. The codes of BS and BnBk are provided by their authors and RDS is a fine-tuned open source version in https://github.com/zhelih/rds-serial.

We mainly test k values in range 2 to 5, the same as in the existing literature. We set the cut off time as 1800s (half an hour) for each algorithm and each instance. As we will show lastly, the running time of solving an instance with k larger than 5 is often prohibitively long.

Real World Graphs

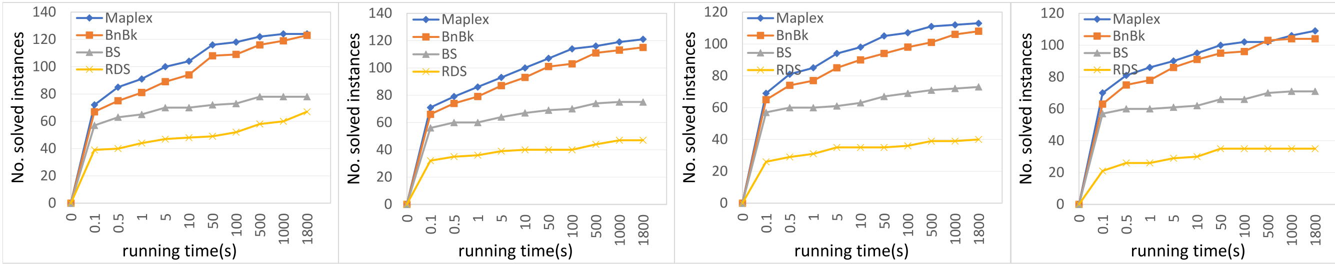
Graphs from Network Repository We first test all the undirected and simple real world graphs in the Network Repository (Rossi and Ahmed 2015) which are also used in (Gao et al. 2018). This set includes 139 biological networks, collaboration networks, Facebook networks, infrastructure networks and so on.

In Fig. 2, we show the number of solved instances within different time frames for $k = 2, 3, 4$ and 5. In terms of the number of solvable instances in different time frames, we see a clear dominance among the four algorithms, i.e., Maplex>BnBk>BS>RDS. Indeed, setting a time limit of

graph	k	peel-reduct			subgraph-reduct		second-reduct			
		#vtx	#edges	time	#vtx	time	#vtx	#edges	time	speedup
sc-nasasrb (54870, 1311227)	2	53675	1293082	0.05	**12960**	3.04	12972	322518	0.96	3.1x
	3	53945	1298844	0.06	**21575**	7.72	21586	470123	1.96	3.9x
	4	54012	1300215	0.06	51075	10.08	**51069**	1193313	1.47	6.8x
	5	54012	1300215	0.06	51154	8.67	**51153**	1205309	1.40	6.1x
sc-pkustk13 (94893, 3260967)	2	92533	3194949	0.11	27558	113.48	**27120**	644421	3.40	33.3x
	3	92539	3195147	0.12	27789	143.68	**27120**	644448	2.50	57.4x
	4	92708	3200612	0.15	27615	138.00	**27120**	644448	3.61	38.2x
	5	92730	3201296	0.15	94795	61.05	**60849**	1982268	45.75	1.3x
soc-flixster (2523386, 7918801)	2	21774	856061	0.70	282	14.31	**275**	12479	2.94	4.8x
	3	15651	631892	0.71	**252**	9.99	263	11618	2.40	4.1x
	4	7545	312522	0.68	**237**	4.42	239	10187	1.48	2.9x
	5	844	38648	0.54	237	3.48	**226**	9407	0.65	5.3x
soc-FourSquare (639014, 3214986)	2	23248	860065	0.23	N\A	N\A	**12743**	481491	122.41	N\A
	3	18709	724599	0.19	**100**	1048.56	9692	374193	92.11	11.3x
	4	17257	678326	0.20	**84**	889.88	9612	371635	80.77	11.0x
	5	17257	678326	0.22	**84**	863.22	10645	410671	76.10	11.3x
soc-LiveMocha (104103, 2193083)	2	66980	2059265	0.15	4210	99.71	**4064**	182930	16.23	6.1x
	3	63437	2031014	0.19	6031	130.53	**4057**	182887	15.82	8.2x
	4	60099	2001067	0.19	4182	90.17	**4036**	182688	19.12	4.7x
	5	57177	1971921	0.18	5884	112.14	**4005**	182378	18.41	6.0x
soc-pokec (1632803, 22301964)	2	605699	15871320	2.74	**838**	174.10	2359	50231	40.03	4.3x
	3	510227	14174602	1.70	**298**	214.70	1194	25050	33.29	6.4x
	4	480453	13586210	1.55	**298**	124.54	1192	25010	33.12	3.7x
	5	422488	12355265	1.40	**298**	123.04	825	17403	28.84	4.2x
socfb-A-anon (3097165, 23667394)	2	390144	14512286	2.94	2239	234.83	**2155**	64767	50.16	4.6x
	3	357908	13760467	1.85	1657	216.17	**1578**	46887	48.21	4.4x
	4	327525	12991403	1.84	1182	197.86	**1101**	32196	42.58	4.6x
	5	298641	12203325	1.71	819	186.89	**777**	21635	39.81	4.6x
socfb-B-anon (2937612, 20959854)	2	469342	15355191	2.85	37564	440.91	**25847**	645480	78.17	5.6x
	3	450382	15072871	1.80	26887	513.00	**25555**	641827	72.27	7.1x
	4	432416	14787481	2.85	36219	597.15	**24916**	633217	79.60	7.5x
	5	399864	14222285	2.89	34767	593.33	**17124**	452253	70.70	8.3x
socfb-Duke14 (9885, 506437)	2	7013	465047	0.02	685	14.46	**630**	28590	2.11	6.8x
	3	6745	455562	0.03	553	12.89	**396**	20459	2.11	6.1x
	4	6745	455562	0.03	553	12.98	**487**	23704	2.07	6.2x
	5	6536	447455	0.04	399	11.87	**330**	18225	2.01	5.9x
socfb-Indiana (29732, 1305757)	2	19088	1104303	0.07	**2316**	31.04	2317	84021	3.95	7.8x
	3	17038	1021599	0.07	1601	27.85	**1541**	54605	3.55	7.8x
	4	16238	986201	0.07	**1031**	22.28	1224	43309	3.18	7.0x
	5	15503	952191	0.08	**976**	24.37	985	35164	3.26	7.4x
socfb-uci-uni (58790782, 92208195)	2	165396	1179070	15.49	N\A	N\A	**24**	138	17.43	N\A
	3	67366	543298	15.51	N\A	N\A	**0**	0	16.29	N\A
	4	67366	543298	15.22	N\A	N\A	**0**	0	15.99	N\A
	5	26159	234484	14.73	N\A	N\A	**0**	0	15.02	N\A
socfb-UCSB37 (14917, 482215)	2	1050	57497	0.02	148	0.86	**77**	2818	0.13	6.5x
	3	874	47721	0.01	76	0.64	76	2754	0.07	9.0x
	4	874	47721	0.01	76	0.56	76	2754	0.07	8.0x
	5	814	44014	0.01	0	0.49	0	0	0.07	7.0x
tech-as-skitter (1694616, 11094209)	2	5608	389663	0.87	**422**	27.53	423	29468	2.53	10.8x
	3	5041	357594	0.80	404	26.82	404	27713	2.34	11.4x
	4	4196	303506	0.85	403	25.00	**334**	21022	2.11	11.8x
	5	3740	272755	0.85	334	21.47	**204**	9598	1.92	11.1x

Figure 1: The experimental results of three preprocessing techniques. Due to space limit, we only list 13 "hard graphs" which are graphs that cannot be solved by Maplex within 20s when $k = 2$. #vtx and #edge represent the number of vertices and edges after using the preprocessing, respectively. *time* is the running time of using the preprocessing technique. 0.00 means that the time is less than 0.005. N\A indicates that the result cannot be obtained due to memory failure or time out. *speedup* is the speedup of second-reduct over subgraph-reduct.

[1] The code and supplementary documents can be downloaded from https://github.com/ini111/Maplex.git



(a) $k = 2$. (b) $k = 3$. (c) $k = 4$. (d) $k = 5$.

Figure 2: Experimental results of real world graphs.

1800s for each instance, Maplex solves a total of 17 more instances that BnBk.

The influence of preprocessing We compare different preprocessing techniques in Fig. 1, including Peel (denoted by peel-reduct), the preprocessing in Maplex (denoted by second-reduct) and the subgraph-reduction used in BnBk (denoted by subgraph-reduct). The initial lower bounds of peel-reduct and second-reduct are obtained via our heuristic while peel-reduct has its own heuristic for obtaining a lower bound. From the figure, peel-reduct is the fastest but the second-reduct and subgraph-reduct can remove much more vertices and edges than peel-reduct. For large graphs like soc-pokec and socfb-A-anon, the numbers of vertices after using second-reduct are reduced by two orders of magnitude, comparing with that only using peel-reduct. The subgraph-reduct and second-reduct remove roughly equal number of vertices in these graphs. However, the second-reduct is more time-efficient than subgraph-reduct. Generally, a speed up of 4x-10x is often observed in the figure.

The influence of color-bound We continue to study the influences brought by our bounding technique. We set up a tailored version of Maplex, namely Maplex-NoCol, which removes the color-bound as well as the lookahead components. We then compare Maplex, Maplex-NoCol and the baseline algorithms in Fig. 3. In order to eliminate the impact of different preprocessing strategies, we use the kernel graphs which are provided by our preprocessing as the input of these algorithms.

Comparing between Maplex with Maplex-NoCol, we see that color-bound and lookahead techniques can reduce the branching nodes by up to an order of magnitude and thus, improve the time-efficiency. Comparing with the other baseline algorithms, Maplex is still the best-performing algorithm generally. Indeed, the baseline algorithms require even more computational time than Maplex when using the original graph as input.

Erdös collaboration graphs A Erdös graph ERDOS-x-y represents the collaborate networks of authors who have Erdös numbers[2] at most y as of year x. We test 6 such graphs that are also used in (Xiao et al. 2017), i.e., ERDOS-97-1, ERDOS-97-2, ERDOS-98-1, ERDOS-98-2, ERDOS-99-

[2]The Erdös number of an author is the length of the shortest path between Paul Erdös and the author in the collaboration networks

graph	k	opt	Maplex		Maplex-NoCol		BnBk	BS	RDS
			nodes	time	nodes	time			
sc-nasasrb	2	24	114134	**31.38**	415682	39.27	1041.54	204.67	N\A
	3	24	567036	**153.13**	2485953	207.67	N\A	558.78	N\A
	4	24	1676545	**1781.37**	7995967	N\A	N\A	N\A	N\A
sc-pkustk13	2	36	505380	144.66	1136684	326.77	279.54	**60.19**	N\A
	3	36	11113284	762.31	28308270	1338.09	N\A	**168.43**	N\A
	4	36	N\A	N\A	N\A	N\A	N\A	**1517.46**	N\A
soc-flixster	2	38	51893366	289.48	259887884	693.63	644.25	120.17	**4.07**
	3	42	N\A	N\A	N\A	N\A	N\A	N\A	**464.16**
soc-FourSquare	2	35	9118949	**1158.08**	N\A	N\A	N\A	N\A	N\A
	3	39	8452295	**87.69**	16477192	97.59	591.75	N\A	N\A
	4	42	44859662	**151.64**	69717963	168.36	169.47	N\A	N\A
	5	44	7279628	**31.05**	11031373	31.67	65.57	N\A	N\A
soc-LiveMocha	2	19	2211000	**10.03**	8538232	21.56	110.07	N\A	230.58
	3	22	231628480	**822.13**	496640471	1041.74	1557.97	N\A	N\A
soc-pokec	2	31	229	**0.26**	1549	0.28	3.01	5.55	19.58
	3	32	11145	**0.10**	27377	0.12	2.23	36.18	272.39
	4	32	1510477	**4.92**	3436847	7.66	6.82	485.93	N\A
	5	34	7399842	27.09	15851749	33.75	**25.18**	1612.22	N\A
socfb-A-anon	2	28	222243	**3.08**	7799968	28.41	26.65	267.68	24.21
	3	32	758323	**8.22**	9580924	33.51	76.81	N\A	1776.45
	4	35	1032676	**8.37**	8210105	25.63	264.98	N\A	N\A
	5	37	1962993	**12.26**	15228233	42.22	251.80	N\A	N\A
socfb-B-anon	2	27	185317	**3.13**	11423410	31.93	62.72	N\A	N\A
	3	30	3906023	**40.58**	80012762	259.72	84.49	N\A	N\A
	4	33	14744144	**90.46**	233789118	534.07	142.73	N\A	N\A
	5	35	136913172	708.45	N\A	N\A	**354.37**	N\A	N\A
socfb-Duke14	2	38	31094914	174.87	88884311	292.12	897.85	643.76	**33.13**
	3	43	N\A	N\A	N\A	N\A	N\A	N\A	**1704.69**
socfb-Indiana	2	51	2943995	69.70	13596023	110.39	284.06	31.54	**30.79**
	3	55	65689191	**646.09**	106278245	671.05	N\A	857.00	N\A
socfb-uci-uni	2	9	0	0.00	0	0.00	0.00	0.00	0.00
	3	10	0	0.00	0	0.00	0.00	0.00	0.00
	4	11	0	0.00	0	0.00	0.00	0.00	0.00
	5	13	0	0.00	0	0.00	0.00	0.00	0.00
socfb-UCSB37	2	59	15705	0.10	17928	0.10	53.27	0.10	**0.03**
	3	63	12174	0.08	13862	0.07	1.03	**0.04**	0.06
	4	66	1181788	5.64	1354989	5.14	0.18	**0.03**	0.06
	5	68	0	0.00	0	0.00	0.00	0.00	0.00
tech-as-skitter	2	69	4446674	**39.72**	17593205	72.18	N\A	375.31	107.10
	3	71	212149171	1376.42	394548503	1563.76	**1316.18**	N\A	N\A
	4	74	N\A	N\A	N\A	N\A	**1202.27**	N\A	N\A
	5	75	N\A	N\A	N\A	N\A	**223.57**	N\A	N\A

Figure 3: The experimental results of Maplex, Maplex-NoCol, BnBk, BS and RDS. The input graphs for each algorithm are pruned by our preprocessing method. *opt* indicates the optimal value. *nodes* refers to the number of recursive calls made by branch-and-bound algorithm. The instance that neither of these algorithms obtains the optimal solution is omitted.

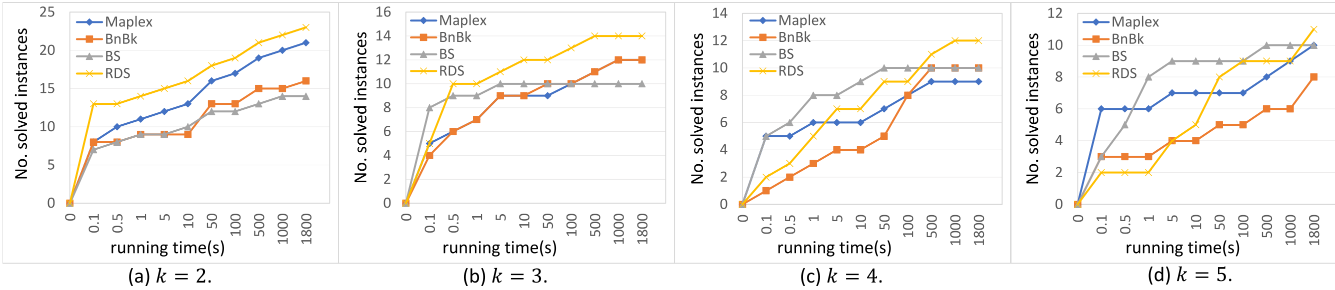


Figure 4: Experimental results of clique graphs for $k = 2, 3, 4$ and 5.

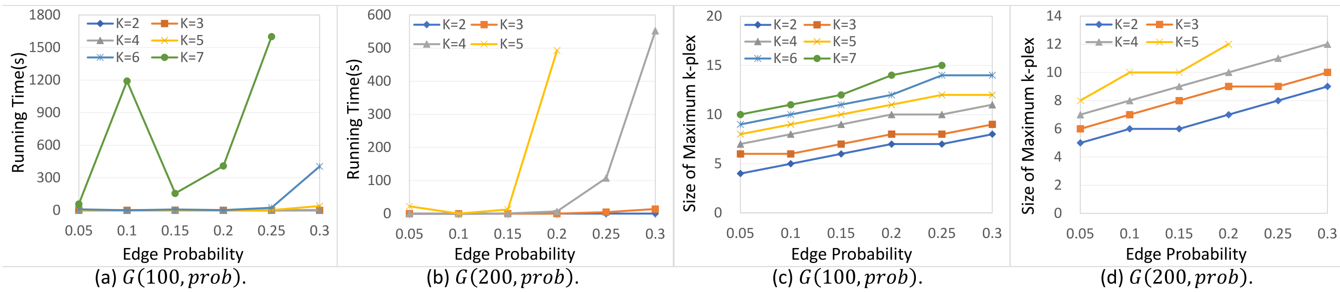


Figure 5: Experimental results for random graphs with edge probability ranging from 0.05 to 0.3 and $n = 100$ or 200.

1 and ERDOS-99-2 for $k = 2, ..., 5$. The numbers of vertices and edges range from 472 to 6100 and 1314 to 9939, respectively. As a result, Maplex and BnBk solve each of these test cases in less than 0.005s, BS spends a bit more time than Maplex and BnBk, but still less than 0.1s. However, RDS cannot solve all the graphs when $k = 3, 4$ and 5 in 1800s.

SNAP and partition graphs We also test the 43 real world graphs in (Trukhanov et al. 2013; Gschwind, Irnich, and Podlinski 2018) which are extracted from SNAP benchmark set (Leskovec and Krevl 2014) and 10th DIMACS challenge. However, we notice that Maplex still outperforms the others. Interested readers can refer to the complete report of our experiments.

Artificial Graphs

Clique graph We present experimental results for the *clique graph* from the Second DIMACS Implementation Challenge[3]. The graphs in this set are extremely dense. Almost all graphs of this set have a diameter only 2 (except the "c-fat" group). Indeed, all preprocessing methods fail to prune one vertex of these graphs. In Fig. 4, we show the number of solved instances against different time frames. There is no dominant algorithm among all k values. RDS outperforms others for $k = 2$ and 3. It is conjectured that the strategies of RDS are naturally suitable for solving dense clique graphs and small ks. BS performs well for $k = 5$ which may result from its multiple branching strategies. Maplex is still competitive but it is believed that Maplex is most suitable for dealing with sparse large real world graphs.

[3]http://networkrepository.com/dimacs.php

Random graphs A random graph $G(n, prob)$ consists of n vertices and random edges which are generated with probability $prob \in [0, 1]$. Concretely, for each pair of vertices in $G(n, prob)$, there is an edge between them with an unified probability $prob$. We generate random graphs to test the behaviour of Maplex with respect to different edge probability $prob$ and ks.

In Fig. 5, we show the changes of running times and sizes of optimal solutions as the edge probability increases from 0.05 to 0.3 and $n = 100$ or 200. As n changes from 100 to 200, the size of maximum k-plex changes mildly. Lastly, as we mentioned, the algorithm cannot find the solution even for these small graphs with 200 vertices when k becomes larger than 5.

Conclusion

In the paper, we proposed new strategies for solving MPLEX efficiently in real world graphs. We applied new reduction rules to enhance the preprocess procedure which enjoys lower computational cost but powerfully shrinks the input graph into a very smaller kernel graph. For the first time, we used the graph color heuristic to obtain a tight upper bound for the exact branch-and-bound.

In the experiments, the final algorithm, Maplex, outperforms the state-of-the-art solvers like BnBk, BS and RDS on real-world graphs and keeps competitive on dense artificial graphs.

It is believed that the work not only provides new insights to the fundamental MPLEX problem but also paves the road of utilizing the k-plex model in real world graph mining tasks.

Acknowledgments

The work is supported by Natural Science Foundation of China (No. 61802049 and No. 61972070), Shenzhen Science and Technology Innovation Commission (No. JCYJ20180508162601910), Shenzhen Institute of Artificial Intelligence and Robotics for Society (No. 2019-INT003) and Sub Project of Independent Scientific Research Project (No. ZZKY-ZX-03-02-04).

References

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

Chen, P.; Wan, H.; Cai, S.; Li, J.; and Chen, H. 2020. Local Search with Dynamic-Threshold Configuration Checking and Incremental Neighborhood Updating for Maximum k-plex Problem. In *AAAI*, 2343–2350.

Conte, A.; De Matteis, T.; De Sensi, D.; Grossi, R.; Marino, A.; and Versari, L. 2018. D2K: Scalable Community Detection in Massive Networks via Small-Diameter k-Plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1272–1281. ACM.

Gao, J.; Chen, J.; Yin, M.; Chen, R.; and Wang, Y. 2018. An Exact Algorithm for Maximum k-Plexes in Massive Graphs. In *IJCAI*, 1449–1455.

Gschwind, T.; Irnich, S.; and Podlinski, I. 2018. Maximum weight relaxed cliques and Russian doll search revisited. *Discrete Applied Mathematics* 234: 131–138.

Jiang, H.; Li, C.-M.; and Manya, F. 2017. An Exact Algorithm for the Maximum Weight Clique Problem in Large Graphs. In *AAAI*, 830–838.

Latapy, M. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science* 407(1-3): 458–473.

Leskovec, J.; and Krevl, A. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

Lewis, J. M.; and Yannakakis, M. 1980. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences* 20(2): 219–230.

Li, C. M.; and Quan, Z. 2010. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI*, volume 10, 128–133.

Lund, C.; and Yannakakis, M. 1993. The approximation of maximum subgraph problems. In *International Colloquium on Automata, Languages, and Programming*, 40–51. Springer.

McClosky, B.; and Hicks, I. V. 2012. Combinatorial algorithms for the maximum k-plex problem. *Journal of Combinatorial Optimization* 23(1): 29–49.

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

Moser, H.; Niedermeier, R.; and Sorge, M. 2012. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *Journal of Combinatorial Optimization* 24(3): 347–373.

Pattillo, J.; Youssef, N.; and Butenko, S. 2012. Clique relaxation models in social network analysis. In *Handbook of Optimization in Complex Networks*, 143–162. Springer.

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

Rossi, R. A.; and Ahmed, N. K. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. URL http://networkrepository.com.

San Segundo, P.; Rodríguez-Losada, D.; and Jiménez, A. 2011. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research* 38(2): 571–581.

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

Shirokikh, O. A. 2013. *Degree-based Clique Relaxations: Theoretical Bounds, Computational Issues, and Applications*. Ph.D. thesis, University of Florida.

Tomita, E.; and Seki, T. 2003. An efficient branch-and-bound algorithm for finding a maximum clique. In *International Conference on Discrete Mathematics and Theoretical Computer Science*, 278–289. Springer.

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

Trukhanov, S.; Balasubramaniam, C.; Balasundaram, B.; and Butenko, S. 2013. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications* 56(1): 113–130.

Wu, K.; Gao, J.; Chen, R.; and Cui, X. 2019. Vertex Selection Heuristics in Branch-and-Bound Algorithms for the Maximum k-Plex Problem. *International Journal on Artificial Intelligence Tools* 28(05): 1950015.

Wu, Q.; and Hao, J.-K. 2015. A review on algorithms for maximum clique problems. *European Journal of Operational Research* 242(3): 693–709.

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

Zhou, Y.; and Hao, J.-K. 2017. Frequency-driven tabu search for the maximum s-plex problem. *Computers & Operations Research* 86: 65–78.

Zhou, Y.; Xu, J.; Guo, Z.; Xiao, M.; and Jin, Y. 2020. Enumerating Maximal k-Plexes with Worst-Case Time Guarantee. In *AAAI*, 2442–2449.

Zhu, J.; Chen, B.; and Zeng, Y. 2020. Community detection based on modularity and k-plexes. *Information Sciences* 513: 127–142.