Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

A Fast Algorithm to Compute Maximum k-Plexes in Social Network Analysis

Mingyu Xiao,[†] Weibo Lin,[†] Yuanshun Dai,[†] Yifeng Zeng[‡]
[†] School of Computer Science and Engineering,
University of Electronic Science and Technology of China, China
[‡] School of Computing, Teesside University, UK

Abstract

A clique model is one of the most important techniques on the cohesive subgraph detection; however, its applications are rather limited due to restrictive conditions of the model. Hence much research resorts to k-plex - a graph in which any vertex is adjacent to all but at most k vertices - which is a relaxation model of the clique. In this paper, we study the maximum k-plex problem and propose a fast algorithm to compute maximum k-plexes by exploiting structural properties of the problem. In an n-vertex graph, the algorithm computes optimal solutions in $c^n n^{O(1)}$ time for a constant $c < 2$ depending only on k. To the best of our knowledge, this is the first algorithm that breaks the trivial theoretical bound of 2^n for each $k \geq 3$. We also provide experimental results over multiple real-world social network instances in support.

Introduction

In computational social networks, finding a large cohesive subgraph is an extensively studied topic with a large number of applications. Clique is one of the earliest and most commonly used models in the field of cohesive subgraphs detection. A clique is a graph with an edge between any pair of vertices, which can be regarded as the most cohesive graph. The MAXIMUM CLIQUE problem, to find a clique of maximum size in a graph, is a fundamental problem in graph algorithms not only having great applications in social networks but also finding applications in ad hoc wireless networks (Chen, Liestman, and Liu 2004), data mining (Washio and Motoda 2003), biochemistry and genomics (Butenko and Wilhelm 2006), and many others.

Due to its overly restrictive (Alba 1973) and modeling disadvantages (Freeman 1992), the clique has been challenged by many practical problems. Alternative approaches were suggested that essentially relaxed the definition of cliques. Researchers have relaxed a variety of clique properties including familiarity, reachability, and robustness (Balasundaram, Butenko, and Hicks 2011). In graph theoretic terms, these properties correspond to vertex degree, path length, and connectivity respectively. This paper focuses on a relaxation model of clique by relaxing its familiarty restriction known as a k-plex (Seidman and Foster 1978). A simple undirected graph with n vertices is a k-plex if the degree of each vertex of the graph is at least $n - k$. When $k = 1$, a 1-plex is a clique. In the MAXIMUM k-PLEX problem, we aim to find a maximum vertex subset S of a given graph such that the subgraph $G[S]$ induced by S is a k-plex.

The applications and research on k-plex receive growing attention such as using k-plex to analyze social networks of terrorists (Krebs 2002), clustering and partitioning of graph-based data using k-plex (Du et al. 2007; Newman 2001), etc. Note that the complement graph of a k-plex is a graph of maximum degree at most $k - 1$. To find a maximum k-plex in a graph G is equivalent to find a maximum induced subgraph of degree bounded by $k - 1$ in the complement graph of G. The later problem is also known as the k'-BOUNDED-DEGREE VERTEX DELETION problem (to make the degree of a graph at most k' by deleting a minimum number of vertices). k'-BOUNDED-DEGREE VERTEX DELETION itself also has many applications in serval areas (Fellows et al. 2011; Xiao 2017).

The NP-completeness of MAXIMUM k-PLEX and k'-BOUNDED-DEGREE VERTEX DELETION problems with each fixed $k \geq 1$ (or $k' \geq 0$) was established many years ago (Lewis and Yannakakis 1980). For MAXIMUM 1-PLEX, known as MAXIMUM CLIQUE or MAXIMUM INDEPENDENT SET in the complement graph, it is a fundamental problem in exact exponential algorithms and it can be solved in $O^*(1.1996^n)$ time (Xiao and Nagamochi 2013) in an n-vertex graph. For $k = 2$, MAXIMUM 2-PLEX can be solved in $O^*(1.3656^n)$ time (Xiao and Kou 2016). A simple brute-force algorithm for MAXIMUM k-PLEX by enumerating and checking all vertex subsets of the graph runs in $2^n n^{O(1)}$ time. We are not aware of any algorithm faster than the trivial exponential bound 2^n for any $k \geq 3$.

In parallel, Balasundaram et al. (2011) gave an integer programming formulation and designed a branch-and-cut algorithm to solve MAXIMUM k-PLEX exactly. McClosky et al. (2012) derived a new upper bound on the cardinality of k-plexes and adapted some clique combinatorial algorithms to find maximum k-plexes, both of heuristic and exact nature. Moser et al. (2012) gave an exact algorithm with better experimental results. All the above exact algorithms run in $2^n n^{O(1)}$ time theoretically.

Our Contributions. This paper contributes to the k-plex

literature both from theory and practice. We investigate several structural properties of MAXIMUM k-PLEX, most of which are related to the lower bound and will be used to prune the search branches in our algorithm. Based on these properties, we design a branch-and-search algorithm for MAXIMUM k-PLEX and analyze its running time bound in a theoretical way. We prove Theorem 1. Some values of σ_k for small k are shown in Table 1.

Theorem 1. MAXIMUM k-PLEX *can be solved in* $\sigma_k^n n^{O(1)}$ *time, where* $\sigma_k < 2$ *is a value related to* k.

$k=$	1	2	3	4	5
$\sigma_k=$	1.6181	1.8637	1.9476	1.9786	1.9910
$k=$	6	7	8	9	10
$\sigma_k=$	1.9961	1.9983	1.9992	1.9996	1.9998

Table 1: The values of σ_k for some small k

This is the first algorithm that breaks the trivial exponential bound of 2^n for each fixed $k \geq 3$. In practice, our algorithm is efficient and easy to implement. Experimental results on large social networks from real-world and artificial graphs show that our algorithm is much faster than three well-known algorithms, especially on benchmark instances from the real-world graphs. Our codes and data in this paper are publicly available (*https://github.com/Lweb/KPLEX*).

Structural Properties

Let $G = (V, E)$ be a simple and undirected graph with $n = |V|$ vertices and $m = |E|$ edges. For a subgraph (resp., a vertex subset) X, the subgraph induced by $V(X)$ (resp., X) is denoted by $G[X]$, and $G[V - V(X)]$ (resp., $G[V - X]$) is also written as $G - X$. A vertex u is called a *neighbor* of v if there is an edge between u and v. For a subgraph or a vertex subset X, the set of neighbors of v in X is denoted by $N_X(v)$, and $V(X) \setminus N_X(v)$ is also written as $\overline{N_X}(v)$. The *degree* of v in X is $deg_X(v) \doteq |N_X(v)|$. The distance between vertices u and v in a graph G, denoted by $dis_G(u, v)$, is the number of edges in a shortest path between u and v in G. The *diameter* of a graph G is defined to be $diam(G) = \max_{u,v \in V(G)} dis_G(u, v)$.

For any integer $k \geq 1$, a vertex v in a graph G is *k-satisfied* if $deg_G(v) \geq |V(G)| - k$ or *k-unsatisfied* otherwise. A graph G is a k-plex if all vertices in it are k-satisfied, i.e., $\min_{v \in V(G)} deg_G(v) \geq |V(G)| - k$. MAXIMUM k-PLEX is to find a k-plex of maximum size in a given graph. In this paper, we will consider a general problem.

The constrained k-plex problem
Input: a graph $G = (V, E)$, an integer $k \geq 1$ and a vertex subset $F \subseteq V$;
Object: to find a maximum vertex set S such that $F \subseteq S$ and the induced subgraph $G[S]$ is a k-plex.

A k-plex containing the vertex subset F is also called an *F-constrained k-plex*. The constrained k-plex problem is to find a maximum F-constrained k-plex and an instance of it is denoted by $I = (G = (V, E), k, F)$.

We may always use $U = V \setminus F$ to denote the set of vertices not in F. The normal k-plex problem is the special case of the constrained k-plex problem with $F = \emptyset$.

Seidman and Foster (1978) gave several basic structural properties of the k-plex. Facets for the k-plex polytope and some other properties were developed by Balasundaram, Butenko, and Hicks (2011). The following two properties are frequently used in the literature.

Property 1. *Any induced subgraph of a k-plex is a k-plex.*

Property 2. *For a k-plex G with n vertices, if $k < \frac{n+2}{2}$, then $diam(G) \leq 2$.*

We extend Property 2 to the follows.

Property 3. *For a k-plex G with n vertices and any integer $c \geq 2$, if $|V| > 2k - c$, then $diam(G) \leq c$.*

Proof. We prove the converse negative proposition. Assume that $G = (V, E)$ is a k-plex with $diam(G) > c$ for some integer $c \geq 2$. Let $diam(G) = c + m, m > 0$. Let P be a shortest path between two vertices v and u such that the length of P is $diam(G)$. We use $D_i(v)$ to denote the set of vertices whose distance to v is exactly i, i.e., $D_i(v) = \{t \mid d_G(v, t) = i, t \in V\}$. It is clear that $|D_i(v)| > 0$ for $0 \leq i \leq c + m$ and $\sum_{i=0}^{c+m} |D_i(v)| = |V|$. It holds that $|D_0(v)| = 1$ since $D_0(v) = v$. According to the definition of k-plex, we know that $|D_1(v)| \geq |V| - k$. Next we give a lower bound on $|D_{c+m}(v)| + |D_{c+m-1}(v)|$. The vertex u is in $D_{c+m}(v)$. Each neighbor of u is either in $D_{c+m}(v)$ or $D_{c+m-1}(v)$. According to the definition of k-plex, we know that u has at least $|V| - k$ neighbors. Thus, $|D_{c+m}(v)| + |D_{c+m-1}(v)| \geq |V| - k + 1$. Since $c + m \geq 3$, we know that $c + m - 1 > 1$. We get that

$$\begin{aligned} |V| &= |D_0(v)| + |D_1(v)| + \textstyle\sum_{i=2}^{c+m-2} |D_i(v)| + \\ &\quad |D_{c+m-1}(v)| + |D_{c+m}(v)| \\ &\geq 1 + (|V| - k) + (c + m - 3) + (|V| - k + 1) \\ &= 2|V| - 2k + c + m - 1, \end{aligned}$$

which implies $|V| \leq 2k - c - m + 1 \leq 2k - c$. □

We reveal more properties of the constrained problem.

Reducible Vertices. In an instance $I = (G = (V, E), k, F)$, a vertex v in $U = V \setminus F$ is *F-reducible* if there is a maximum F-constrained k-plex containing v and *D-reducible* if no maximum F-constrained k-plex contains v. We can reduce an instance by deleting any D-reducible vertex from the graph and adding an F-reducible vertex to F preserving the optimality of the problem. It is hard to find out all F-/D-reducible vertices in polynomial time; otherwise, we can solve the NP-hard problem in a polynomial time. We analyze the properties for reducible vertices.

Lemma 1. *Given an instance $I = (G = (V, E), k, F)$ and a vertex $v \in U = V \setminus F$. If the induced subgraph $G[F \cup \{v\}]$ is not a k-plex, then v is D-reducible.*

Proof. Since $G[F \cup \{v\}]$ is not a k-plex, we know that no k-plex contains all vertices in $F \cup \{v\}$ by Property 1. Thus, no F-constrained k-plex contains v and then v is D-reducible. □

Lemma 2. *Given an instance $I = (G = (V, E), k, F)$ and a vertex $v \in U = V \setminus F$. If $|F \setminus N(v)| \geq k$, then v is D-reducible.*

Lemma 3. *Given an instance $I = (G = (V, E), k, F)$ and a vertex $v \in U = V \setminus F$. If there is a vertex $u \in F$ such that $|F \setminus N(u)| = k$ and v is not adjacent to u, then v is D-reducible.*

We can see that Lemmas 2 and 3 satisfy the condition in Lemma 1. The induced subgraph $G[F \cup \{v\}]$ is not a k-plex since v or u is k-unsatisfied in $G[F \cup \{v\}]$.

Lemma 4. *Let $v \in U$ be a vertex in an instance $I = (G = (V, E), k, F)$. If v and all vertices not adjacent to v are k-satisfied, then vertex v is F-reducible.*

Proof. Let $F' = F \cup \{v\}$. We show that any vertex set is a maximum F-constrained k-plex if and only if it is also a maximum F'-constrained k-plex.

Let S be an arbitrary maximum F-constrained k-plex and S' be an arbitrary maximum F'-constrained k-plex. Since $F \subset F'$, we know that S' is also an F-constrained k-plex. By the maximality of S, we know that $|S| \geq |S'|$. Note that for any k-plex not containing v, after adding v to it, it becomes a bigger k-plex. Any maximum F-constrained k-plex including S must contain v. Thus, S is also an F'-constrained k-plex. By the the maximality of S', we know that $|S'| \geq |S|$. We get that $|S| = |S'|$.

Hence S' is also a maximum F-constrained k-plex and S is also a maximum F'-constrained k-plex. □

Exchangeable Vertices. If there is a maximum F-constrained k-plex S containing vertex v but not vertex u and $(S \setminus \{v\}) \cup \{u\}$ is still a maximum F-constrained k-plex in the instance, we say that v is *exchangeable* with u.

Lemma 5. *Assume that vertex v is exchangeable with vertex u in an instance. There is a solution to the instance that either contains both of v and u or not contains v.*

Proof. Let S be a solution to the instance. Assume that $v \in S$ and $u \notin S$, otherwise we are done. By the definition of exchangeable vertices, we know that $S' = S \setminus \{v\} \cup \{u\}$ is still maximum F-constrained k-plex, which does not contain v. □

The lemma implies that exchangeable vertices can be used to design an effective branching rule to search a solution. When v is exchangeable with u, we can either delete v from the instance or include both of v and u to F. We identify several exchangeable vertices below.

A vertex v is *dominated* by another vertex u if any neighbor of v is either u or a neighbor of u. Note that in the definition, v and u are not required to be adjacent.

Lemma 6. *If a vertex v is dominated by another vertex u, then v is exchangeable with u.*

Proof. Let S be an arbitrary k-plex that contains v but not u. We only need to show that $S' = (S \setminus \{v\}) \cup \{u\}$ is also a k-plex. Any vertex (except u) adjacent to v is also adjacent to u. After replacing v with u in S, the degree of any vertex in $S \setminus \{v\}$ will not decrease. Hence all vertices in $S \setminus \{v\}$ are still k-satisfied in $G[S']$. Since any vertex in S adjacent to v is also adjacent to u, we know that $deg_{S'}(u) \geq deg_S(v)$. Since $|S'| = |S|$ and v is k-satisfied in $G[S]$, we know that u is also k-satisfied in $G[S']$. All vertices in S' are k-satisfied in $G[S']$ and then S' forms a k-plex. □

Reductions Based on Lower Bounds. As we will develop a branch-and-search algorithm (to be elaborated later) for solving the MAXIMUM k-PLEX, we move further to discuss some properties to be used to prune a search tree. Once we have obtained a feasible solution of size s, we may be able to abandon branches which will only reach feasible solutions of size at most s.

Given an integer s as a lower bound on the size of the solution. We are only interested in the instances with solution size greater than s. The following properties allow us to prune the search tree in our algorithm.

Lemma 7. *Given an instance $I = (G, k, F)$ and an integer s. For any vertex v in the graph, if $deg_G(v) \leq s - 1 - k$, then any solution to I of size at least s (if it exists) will not contain v.*

Proof. For any vertex set S of size at least s, v is not adjacent to at least $k + 1$ vertices in S. If S contains v, then S is not a k-plex. □

The above lemma also implies that we can abandon the instances with at most s vertices directly once a lower bound s of the solution size is given.

Lemma 8. *Given an instance $I = (G, k, F)$ and an integer s. If there is a vertex $u \in F$ and a vertex $v \in V \setminus F$ such that $dis_G(u, v) > \max(2, 2k - s + 1)$, then any solution to I of size at least s (if it exists) will not contain v.*

Proof. For any subgraph G' containing u and v, it holds that $dis_{G'}(u, v) \geq dis_G(u, v) > \max(2, 2k - s + 1)$. By Property 3, we know that if G' contains at least s vertices then G' is not a k-plex. So any k-plex of size at least s will not contain both u and v. □

The properties in this section will be used to design some reduction rules to reduce the instance directly and some branching rules with a good performance.

Branching Rules

In a branch-and-search algorithm, we may search a maximum solution to an instance by recursively branching on the current instance into several smaller instances until the instance becomes polynomially solvable or satisfies some properties. To evaluate the size of the search tree generated by this paradigm, we need to evaluate the size of the search tree in the algorithm. In our algorithm, we will simply select the number n_0 of vertices in $U = V \setminus F$ as the measure. Clearly, the problem can be solved directly when $n_0 \leq 0$. Let $C(n_0)$ denote the maximum number of leaves in the search tree generated by the algorithm for any instance with $|U| \leq n_0$. For a branching operation, where we branch on an instance with $|U| = n_0$ into l branches such that in the

i-th branch the size of U decreases by at least a_i, we obtain a recurrence relation

$$C(n_0) \leq C(n_0 - a_1) + C(n_0 - a_2) + \cdots + C(n_0 - a_l).$$

The largest root of the function $f(x) = 1 - \sum_{i=1}^{l} x^{-a_i}$ is called the *branching factor* of the recurrence. Let γ be the maximum branching factor among all branching factors in the algorithm. The size of the search tree that represents the branching process of the algorithm applied to an input instance with $|U| = n_0$ is given by $O(\gamma^{n_0})$. More details about the analysis and how to solve recurrences can be found in the monograph (Fomin and Kratsch 2010).

Our branch-and-search algorithm first applies some reduction rules to reduce instances. When the instances cannot be reduced anymore, we use branching rules to search a solution. We have three branching rules below.

Branching Rule 1: Branching on dominated vertices. If there are two vertices $v, u \in U$ such that v is dominated by u, then we branch into two branches by either deleting v from the instance or including both of v and u to F. The correctness of this rule is based on Lemma 5 and Lemma 6. In the first branch vertex v is removed from U, and in the second branch vertices v and u are removed from U. We can get the following recurrence at least for this operation, the branching factor of which is 1.6181.

$$C(n_0) \leq C(n_0 - 1) + C(n_0 - 2). \tag{1}$$

Branching Rule 2: Branching on F-vertices. This branching rule only considers k-unsatisfied vertices in F. Let v be a k-unsatisfied vertex in F, i.e.,

$$|\overline{N_G}(v)| = |V| - deg_G(v) \geq k + 1.$$

Any F-constrained k-plex contains at most $k - |\overline{N_F}(v)|$ vertices in $\overline{N_U}(v) = U \setminus N_U(v)$; otherwise, the degree of v will not satisfy the definition of k-plex. Let $q = k - |\overline{N_F}(v)|$ and $p = |\overline{N_U}(v)|$. We have that

$$p > q, \tag{2}$$

since v is a k-unsatisfied vertex. Note that $v \in \overline{N_F}(v)$ and then $|\overline{N_F}(v)| \geq 1$. We have that

$$q \leq k - 1. \tag{3}$$

Let

$$\overline{N_U}(v) = \{x_1, x_2, \ldots, x_p\}.$$

Our branching rule on a k-unsatisfied vertex $v \in F$ is to generate $q + 1$ branches:
- in the first branch, x_1 is deleted from the graph;
- for $i \in \{2, \ldots, q\}$, in the ith branch, $\{x_1, x_2, \ldots, x_{i-1}\}$ is included to F and x_i is deleted from the graph;
- in the $(q+1)$th branch, $\{x_1, x_2, \ldots, x_q\}$ is included to F and $\{x_q, x_{q+1}, \ldots, x_p\}$ is deleted from the graph.

The correctness of this branching rule is based on the following observation: Let S be an arbitrary solution to the instance. If S contains all the q vertices $\{x_1, x_2, \ldots, x_q\}$, then S cannot contain any vertices in $\{x_{q+1}, x_{q+2}, \ldots, x_p\}$, since any k-plex contains at most q vertices in $\overline{N_U}(v)$. For this case, the last branch will not lose the solution. Otherwise, we let i_0 is the smallest index such that x_{i_0} is not in S and it holds that $i_0 \leq q$. For this case, the i_0th branch will not lose the solution.

Next, we analyze the branching factor of this operation. For $i \in \{1, 2, \ldots, q\}$, in the ith branch exactly i vertices are removed from U. In the last branch, all the p vertices in $\overline{N_U}(v)$ are removed from U. We get a recurrence relation

$$\begin{aligned} C(n_0) \leq\ & C(n_0 - 1) + C(n_0 - 2) \\ & + \cdots + C(n_0 - q) + C(n_0 - p), \end{aligned} \tag{4}$$

where $p \geq q+1$ and $q \leq k-1$ by (2) and (3). The branching factor of this recurrence is a root of the function

$$x^p - x^{p-1} - x^{p-2} - \cdots - x^{p-q} - 1 = 0.$$

It is not hard to check that when $p = q + 1$ and $q = k - 1$, the branching factor reaches the largest value. For the worst case, the branching factor is a root of the function

$$x^k - x^{k-1} - x^{k-2} - \cdots - x^1 - 1 = 0,$$

which is equivalent to

$$x^{k+1} - 2x^k + 1 = 0. \tag{5}$$

The largest root of the above function, denoted by γ_k, is the branching factor of recurrence (4). Some values of γ_k for different k are given in Table 2. Note that $\gamma_1 = 1$. This means when $k = 1$, this operation is not a branching operation. It only generates one instance by deleting all vertices in $\overline{N_U}(v)$, which can be regarded as a reduction operation.

$k =$	1	2	3	4	5
$\gamma_k =$	1	1.6181	1.8637	1.9476	1.9786
$k =$	6	7	8	9	10
$\gamma_k =$	1.9910	1.9961	1.9983	1.9992	1.9996

Table 2: The values of γ_k for some small k

Branching Rule 3: Branching on U-vertices. We have shown that k-unsatisfied vertices in F may not always exist. When there are no such kind of vertices, Branching Rule 2 cannot by applied and we turn to consider k-unsatisfied vertices in U. Assume that there are no k-unsatisfied vertices in F. Now there must exist k-unsatisfied vertices in U; otherwise, the instance has only k-satisfied vertices and the instance can be solved directly by Lemma 4.

Let v be a k-unsatisfied vertex in U. Our branching rule first generates two branches by either deleting v from the graph or including v to F. In each branch, the number of vertices in U decreases by 1. We look at the second branch I_2 where v is included to F. Now v becomes a k-satisfied vertex in F and we can further branch on it with (4) by Branching Rule 2. Combining them, we get a recurrence

$$\begin{aligned} C(n_0) \leq\ & C(n_0 - 1) + C(n_0 - 2) + \\ & \cdots + C(n_0 - q - 1) + C(n_0 - p - 1), \end{aligned} \tag{6}$$

where $p \geq q+1$ and $q \leq k-1$ by (2) and (3). The branching factor of this recurrence is a root of the function

$$x^{p+1} - x^p - x^{p-1} - x^{p-2} - \cdots - x^{p-q} - 1 = 0.$$

It is not hard to check that when $p = q + 1$ and $q = k - 1$, the branching factor reaches the largest value. For the worst case that $p = q + 1$ and $q = k - 1$, the branching factor is a root of the function

$$x^{k+1} - x^k - x^{k-1} - x^{k-2} - \cdots - x^1 - 1 = 0,$$

which is equivalent to

$$x^{k+2} - 2x^{k+1} + 1 = 0. \quad (7)$$

The largest root of the above function, denoted by σ_k, is the branching factor for this case. We can see that

$$\sigma_k = \gamma_{k+1} > \gamma_k.$$

The Branch-and-Search Algorithm

As presented below, the Branch-and-Search (BS) Algorithm $\texttt{plex}(I = (G, k, F), bound)$ takes an instance I of the constrained k-plex problem and an integer $bound$ as the input, and checks whether I has an F-constrained k-plex of size at least $bound$. Note that for the purpose of presentation, the algorithm is described to solve a decision problem, i.e., only needs to answer yes or no. It can be modified to return a solution directly if it exists.

Algorithm 1 The Branch-and-Search (BS) Algorithm $\texttt{plex}(I = (G, k, F), bound)$.

Input: an instance I of the constrained k-plex problem and an integer $bound$.
Output: 1 if I has an F-constrained k-plex of size at least $bound$ or 0 otherwise.

1: **if** $G[F]$ is not a k-plex **then**
2: **return** 0
3: **else if** $F = V(G)$ **then**
4: **return** 1 if $|F| \geq bound$ or 0 if $|F| < bound$
5: **else if** there is a D-reducible vertex v identified by Lemma 2 or Lemma 3 **then**
6: **return** $\texttt{plex}((G \setminus \{v\}, k, F), bound)$
7: **else if** there is an F-reducible vertex v identified by Lemma 4 **then**
8: **return** $\texttt{plex}((G, k, F \cup \{v\}), bound)$
9: **else if** there is a vertex v such that $deg_G(v) \leq bound - 1 - k$ **then**
10: **return** $\texttt{plex}((G \setminus \{v\}, k, F), bound)$
11: **else if** there is a vertex $u \in F$ and a vertex $v \in V \setminus F$ such that $dis_G(u, v) > \max(2, 2k - bound + 1)$ **then**
12: **return** $\texttt{plex}((G \setminus \{v\}, k, F), bound)$
13: **else if** there are two vertices $v, u \in V \setminus F$ such that v is dominated by u **then**
14: **return** $\texttt{plex}((G \setminus \{v\}, k, F), bound) \wedge \texttt{plex}((G, k, F \cup \{v, u\}), bound)$
15: **else if** there are k-unsatisfied vertices in F **then**
16: let v be a such kind of vertex of minimum degree
17: **return** $\wedge_{i=1}^{q+1} \texttt{plex}(I_i, bound)$
18: **else if** there are k-unsatisfied vertices in U **then**
19: let v be a such kind of vertex of minimum degree
20: **return** $\wedge_{i=1}^{q+2} \texttt{plex}(I'_i, bound)$
21: **end if**

In the algorithm, lines 1-2 check whether the input satisfies the condition in Property 1. Lines 3-4 deal with the boundary cases. Lines 5-8 delete some reducible vertices based on Lemmas 2 to 4. The correctness of lines 9-12 are based on Lemma 7 and Lemma 8. Lines 13-14 branch on dominated vertices with Branching Rule 1, lines 15-17 branch on k-unsatisfied vertices in F with Branching Rule 2, and lines 18-20 branch on k-unsatisfied vertices in $U = V \setminus F$ with Branching Rule 3. We have analyzed that when the graph has no k-unsatisfied vertices, all vertices in $V \setminus F$ will become F-reducible vertices and will be included to F in lines 7-8. When the graph has some k-unsatisfied vertices, at least one condition in lines 15 and 18 can be fulfilled. These imply the algorithmic correctness.

In the algorithm, we have the three kinds of branches in lines 13-20. We have analyzed the branching factors of these three branching rules, the largest branching factor of them is σ_k, where $\sigma_k < 2$ is the biggest root of function (7). Except these three branches, each of other steps of the algorithm can be executed in polynomial time. Therefore, $\texttt{plex}(I = (G, k, F), bound)$ runs in $\sigma_k^n n^{O(1)}$ time.

To solve the original constrained k-plex problem, we only need to find the maximum value of $bound$ such that $\texttt{plex}(I = (G, k, F), bound) = 1$ and $\texttt{plex}(I = (G, k, F), bound + 1) = 0$. We can search for $bound$ in increasing order of value, which will increase the running time bound by a factor of n. Therefore, our problem can be solved in $\sigma_k^n n^{O(1)}$ time, which implies Theorem 1.

We can also use a binary search to find the maximum value of $bound$, which will reduce the polynomial part of the running time by a factor of $n/\log n$. We initially set the search space $[a, b]$ to be $[0, n]$, iteratively check whether the graph has a k-plex of size $\lceil \frac{a+b}{2} \rceil$ and then update the search space by $[a, b] \leftarrow [a, \lceil \frac{a+b}{2} \rceil]$ if no or $[a, b] \leftarrow [\lfloor \frac{a+b}{2} \rfloor, b]$ if yes. By using this method, we only need to execute $O(\log n)$ loops instead of n loops to compute the maximum value of $bound$.

Experimental Results

To evaluate the performance, we compare the branch-and-search algorithm (BS) with three well-known exact algorithms for MAXIMUM k-PLEX on two types of data sets to evaluate its performance. The three previous algorithms are IPBC by Balasundaram et al. (2011), OsterPlex by McClosky and Hicks (2012) and GuidedBranching by Moser et al. (2012). Our algorithm, namely BS algorithm, is implemented in C++ and the experiments run on a 2.5 GHz Intel Core i5-3210M processor with 4GB memory. The experimental environment of the three previous algorithms are a little bit different, but on the same level. The experiments of Balasundaram et al. (2011) were performed on Dell Precision PWS690 machines with a 2.66 GHz Xeon Processor, 3GB main memory, implemented using ILOG CPLEX 10.0. The experiments of McClosky and Hicks (2012) were run on a 2.2GHz Dual-Core AMD Opteron processor with 3GB main memory. Moser et al. (2012) used an AMD Athlon 64 3700+ machine with 2.2GHz, 1M L2 cache, and 3GB main memory.

The first data set consists of two batches of well-known scientific collaboration networks and one batch of news relation networks, which are large-scale, real-life social network instances and have been used to evaluate several previous algorithms for MAXIMUM k-PLEX. In a scientific collaboration network, the vertices represent scientists, and an edge connects two of them if they co-author some papers. The collaboration networks centered around Paul Erdös are called *Erdös collaboration networks* or *Erdös graphs*. Instances named ERDOS-x-y in Table 3 are the Erdös collaboration networks of all authors with an Erdös number at most y as of year x, where the Erdös number of an author is the length of the shortest path between Paul Erdös and the author in the collaboration networks (Grossman, Ion, and Castro 2007). Instances named GEOM-t in Table 4 are collaboration networks for computational geometers (Batagelj and Mrvar 2006), where two authors are adjacent if they have jointly published more than t articles. Instances named DAYS-t in Table 5 are text-mining networks based on news released by Reuter during 66 days beginning with the terrorist attacks in New York on September 11, 2001 (Batagelj and Mrvar 2006). Each vertex is a selected word that appeared in the news. Given a threshold t, two words are connected by an edge if there exist more than t sentences in which both appear. This is the meaning of t in the instance name. In Tables 3- 5, all the four algorithms compute the same optimal size of a maximum k-plex. However, our algorithm (BS) uses much less running time. It seems that the time used by our algorithm varies slightly for different small k. In fact, our algorithm can solve these instances within 0.1 second for $k \leq 10$. We think that it is contributed by the extremely low density of these social networks, which makes our reduction rules very efficient.

The second data set consists of the clique instances from the second DIMACS implementation challenge (DIMACS 1995). Those DIMACS instances were developed as a standard test bed for clique algorithms. For some instances, the algorithms may not be able to compute the optimal solution in time limitation three hours. For this case, the algorithms will return $[a, b]$ to denote that the optimal value is between a and b. Table 6 shows that our algorithm is much more efficient than other algorithms for most instances.

Conclusion

We have designed a practical algorithm for MAXIMUM k-PLEX. Experimental results on standard benchmark sets show that our algorithm runs much faster than previous exact algorithms on real-world social network instances, which are usually sparse graphs. For clique instances, which are dense with a large size of a maximum k-plex, our algorithm still has advantages on most instances. More importantly, we theoretically show the time complexity of our algorithm. Our algorithm is the first algorithm that brakes the trivial exponential bound of 2^n on this problem for each $k \geq 3$.

The increasing efficiency of our algorithms facilitates real-world social network analysis. For example, a maximum k-plex size can be viewed as a global measure characterizing the cohesiveness of a social network, and our algorithm can serve as a powerful tool to detect it. In many

Instances ($\|V\|, \|E\|$)	k	Running time in seconds			
		IPBC	OsterPlex	GuidedBranching	BS
ERDOS-97-1 (472, 1314)	2	1.5	0	0.26	0.01
	3	1.8	19	0.57	0.01
	4	2.2	1897	1.12	0.00
	5	5.7	-	6.12	0.01
ERDOS-97-2 (5488, 8972)	2	392.9	1253	4.76	0.01
	3	394.1	$\geq$3600	12.53	0.03
	4	424.0	$\geq$3600	8.86	0.01
	5	1042.8	-	45.07	0.01
ERDOS-98-1 (485, 1381)	2	1.7	0	0.14	0.01
	3	1.8	20	0.98	0.02
	4	2.8	1675	1.14	0.00
	5	7.9	-	6.11	0.01
ERDOS-98-2 (5822, 9505)	2	464.3	1514	5.88	0.02
	3	457.1	$\geq$3600	23.58	0.04
	4	614.7	$\geq$3600	10.31	0.01
	5	1664.6	-	52.81	0.02
ERDOS-99-1 (492, 1417)	2	1.8	0	0.17	0.01
	3	1.8	21	1.8	0.02
	4	1.8	1783	1.47	0.01
	5	9.9	-	8.04	0.01
ERDOS-99-2 (6100, 9939)	2	526.5	1757	7.05	0.02
	3	520.0	$\geq$3600	33.82	0.05
	4	526.3	$\geq$3600	17.23	0.02
	5	653.5	-	122.6	0.02

Table 3: Results for Erdös instances

Instances ($\|V\|, \|E\|$)	k	Running time in seconds			
		IPBC	OsterPlex	GuidedBranching	BS
GEOM-0 (7343, 11898)	2	2384.4	397	9.73	0.01
	3	2387.1	$\geq$3600	9.67	0.01
	4	2383.7	$\geq$3600	9.6	0.01
	5	2298.1	-	9.64	0.01
GEOM-1 (7343, 3939)	2	753.2	1118	5.23	0.00
	3	747.7	$\geq$3600	5.15	0.01
	4	743.7	$\geq$3600	5.45	0.01
	5	691.6	-	8.11	0.01
GEOM-2 (7343, 1976)	2	530.6	1145	3.36	0.01
	3	524.3	$\geq$3600	3.42	0.00
	4	522.2	$\geq$3600	3.46	0.00
	5	472.6	-	13.68	0.01

Table 4: Results for GEOM instances

Instances ($\|V\|, \|E\|$)	k	Running time in seconds		
		IPBC	GuidedBranching	BS
DAYS-3 (13332, 5616)	2	3367.8	20.44	0.01
	3	3395.4	21.49	0.01
	4	3489.8	20.45	0.01
	5	15336.9	78.64	0.01
DAYS-4 (13332, 3251)	2	2635.7	17.69	0.00
	3	2625.1	17.85	0.00
	4	2642.3	17.68	0.00
	5	6201.4	37.74	0.01
DAYS-5 (13332, 2179)	2	2462.9	0.11	0.01
	3	2445.5	0.37	0.01
	4	2426.3	0.31	0.00
	5	2820.8	2.09	0.00

Table 5: Results for DAYS instances

Instances ($\|V\|$, $\|E\|$)	k	k-plex size, running time in seconds		
		IPBC	GuidedBranching	BS
c-fat200-1	1	12, 17.1	12, 0.21	12, 0.01
(200, 1534)	2	12, 148.9	12, 1.10	12, 0.01
c-fat200-2	1	24, 10.4	24, 0.42	24, 0.01
(200, 3235)	2	24, 19.1	24, 3.53	24, 0.01
c-fat200-5	1	58, 2.1	58, 1.17	58, 0.01
(200, 8473)	2	58, 2.1	58, 22.44	58, 0.01
c-fat500-1	1	14, 1334.4	14, 3.95	14, 0.04
(500, 4459)	2	14, 1356.1	14, 11.01	14, 0.05
c-fat500-2	1	26, 535.7	26, 7.25	26, 0.05
(500, 9139)	2	26, 605.3	26, 50.21	26, 0.06
c-fat500-5	1	64, 141.6	64, 17.61	64, 0.05
(500, 23191)	2	64, 141.5	64, 350.56	64, 0.05
c-fat500-10	1	126, 39.9	126, 36.28	126, 0.07
(500, 46627)	2	126, 76.5	126, 1547.25	126, 0.07
hamming6-2	1	32, 0.0	32, 0.00	32, 0.04
(64, 1824)	2	32, 0.0	32, 1.77	32, 33.36
hamming6-4	1	4, 0.2	4, 0.05	4, 0.00
(64, 704)	2	6, 0.3	6, 0.24	6, 0.06
hamming8-4	1	16, 52.2	16, 243.11	16, 129.25
(256, 20864)	2	16, 8115.2	[16,171],$\geq$10800	[2,64],$\geq$10800
johnson8-2-4	1	4, 0.0	4, 0.00	4, 0.00
(28, 210)	2	5, 0.0	5, 0.02	5, 0.01
johnson8-4-4	1	14, 0.1	14, 0.44	14, 0.21
(70, 1855)	2	14, 4.4	14, 40.70	14, 265.32
MANN_a9	1	16, 0.0	16, 0.00	16, 11.7
(45, 918)	2	26, 0.0	26, 0.09	26, 4.78
brock200_1	1	[20,31], $\geq$10800	21, 794.73	21, 2206.61
(200, 14834)	2	[25,53],$\geq$10800	[24,134],$\geq$10800	[2,100],$\geq$10800
brock200_2	1	12, 152.5	12, 23.13	12, 3.54
(200, 9876)	2	[13,24],$\geq$10800	13, 606.16	13, 512.59
brock200_4	1	17, 6617.5	17, 204.58	17, 117.2
(200, 13089)	2	[19,41],$\geq$10800	20, 9691.01	[2,50],$\geq$10800
p_hat300-1	1	8, 127.0	8, 29.72	8, 0.64
(300, 10933)	2	[9,66],$\geq$10800	10, 502.48	10, 46.83
p_hat300-2	1	[25,51],$\geq$10800	25, 242.77	25, 91.38
(300, 21928)	2	[28,85],$\geq$10800	[28,200],$\geq$10800	[29,38],$\geq$10800
p_hat700-1	1	[11,40],$\geq$10800	11, 1464.41	11, 60.17
(700, 60999)	2	[10,291],$\geq$10800	[11,467],$\geq$10800	[12,22],$\geq$10800

Table 6: Results for DIMACS instances

applications of social network analysis, one may also be interested in finding all maximal cohesive subgroups in a social network. It remains a future work to design efficient algorithms for detecting all maximal k-plexes.

Acknowledgements

The work was supported by the National Natural Science Foundation of China, under grant 61370071, and the Fundamental Research Funds for the Central Universities, under grant ZYGX2015J057.

References

Alba, R. D. 1973. A graph-theoretic definition of a sociometric clique. *J. of Mathematical Sociology* 3(1):113–126.

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

Batagelj, V., and Mrvar, A. 2006. Pajek datasets. *http://vlado.fmf.uni-lj.si/pub/networks/data*.

Butenko, S., and Wilhelm, W. E. 2006. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research* 173(1):1–17.

Chen, Y.; Liestman, A.; and Liu, J. 2004. Clustering algorithms for ad hoc wireless networks. *Ad Hoc and Sensor Networks* 28:76.

DIMACS. 1995. Cliques, coloring, and satisfiability: second dimacs implementation challenge. Online reference at http://dimacs.rutgers.edu/Challenges/.

Du, N.; Wu, B.; Pei, X.; Wang, B.; and Xu, L. 2007. Community detection in large-scale social networks. In *Proceedings of the 9th WebKDD and 1st SNA-KDD*, 16–25. ACM.

Fellows, M. R.; Guo, J.; Moser, H.; and Niedermeier, R. 2011. A generalization of nemhauser and trotter's local optimization theorem. *Journal of Computer and System Sciences* 77(6):1141–1158.

Fomin, F. V., and Kratsch, D. 2010. *Exact exponential algorithms*. Springer Science & Business Media.

Freeman, L. C. 1992. The sociological concept of "group": An empirical test of two models. *American journal of sociology* 152–166.

Grossman, J.; Ion, P.; and Castro, R. 2007. The erdös number project (2007). *Acessado em Outubro de*.

Krebs, V. E. 2002. Mapping networks of terrorist cells. *Connections* 24(3):43–52.

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.

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

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.

Newman, M. E. 2001. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences* 98(2):404–409.

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

Washio, T., and Motoda, H. 2003. State of the art of graph-based data mining. *ACM SIGKDD Explorations Newsletter* 5(1):59–68.

Xiao, M., and Kou, S. 2016. Exact algorithms for the maximum dissociation set and minimum 3-path vertex cover problems. *Theoretical Computer Science*.

Xiao, M., and Nagamochi, H. 2013. Exact algorithms for maximum independent set. In *International Symposium on Algorithms and Computation (ISAAC)*, 328–338. Springer.

Xiao, M. 2017. On a generalization of Nemhauser and Trotter's local optimization theorem. *Journal of Computer and System Sciences* 84:97–106.