Fast computing maximum relaxed cliques in massive graphs

Paper ID: 3402 (full version)

Abstract

The k-defective clique, a relaxed clique in which k-violations of adjacency is allowed, is a fundamental graph model with numerous real-world applications. In this paper, we design efficient exact algorithms for the NP-hard maximum k-defective clique problem. First, we propose a rudimentary branch-and-search algorithm, MADEC with worst-case running time better than $O^*(2^n)$. To further reduce the computational costs in practice, we then exploit stronger reduction rules and upgrade MADEC to MADEC$^+$. Experiments show that for a wide range of massive real-life instances, MADEC$^+$ outperforms the state-of-the-art algorithms. For 22 out of 60 instances, MADEC$^+$ shows a speedup of more than an order of magnitude over the others.

1 Introduction

The traditional clique model defined on undirected graphs identifies a pairwise adjacent set of vertices in a graph. The clique is one of the earliest graph models for analyzing the cohesive group in a network. It is applied in various domains such as data mining [Cheng *et al.*, 2012], bio-informatics [Butenko and Wilhelm, 2006] and ad-hoc wireless network [Chen *et al.*, 2004].

However, recent studies point out that in real-life applications, data can be noisy or faulty, large and closely linked cohesive groups can hardly appear as ideal cliques [Pattillo *et al.*, 2012], so other forms of *relaxed clique* are sort for. For example, the *k-plex* [Gao *et al.*, 2018; Conte *et al.*, 2018], *k-club* [Pajouh *et al.*, 2016] and *quasi-clique* [Veremyev *et al.*, 2016] are such cliques which relax the vertex degree, pairwise distance of vertices in induced subgraph and edge density, respectively. The *k-defective clique*, in which k (a given integer) violations of adjacency to the clique restrictions are allowed, is also a popular relaxed clique. By definition, a subset of vertices S is a k-defective clique if the induced graph has at least $\binom{|S|}{2} - k$ edges. It is known that the k-defective clique is a very generalized model. When $k = 0$, a k-defective clique is equivalent to a clique. Meanwhile, any relaxed clique can be a k-defective clique by setting a suitable k. In this sense, the research of k-defective clique is of great importance.

Like the conventional clique, the k-defective clique has been also used in different domains. For example, in massive social networks, communities can be discovered by locating the large k-defective cliques [Pattillo *et al.*, 2012]. In a graph representing proteins and protein interactions, the missing link between two proteins can be predicted by enumerating maximal k-defective cliques [Yu *et al.*, 2006]. Even in transportation science, Sherali *et al.,* use the algorithm of maximum k-defective clique problem for airspace planning [Sherali *et al.*, 2002; Sherali and Smith, 2006].

One can see that the problem of detecting large k-defective cliques is a common task in these applications. In this paper, we study the maximum k-defective clique problem, which is to find the k-defective cliques of maximum size from a graph. Indeed, the NP-hardness of the maximum k-defective clique problem was early established by theorems in [Yannakakis, 1978]. As far as we know, the maximum 0-defective clique problem, i.e., the maximum clique problem, can be solved in $O^*(1.1996^n)$ by [Xiao and Nagamochi, 2017], but we are not aware of any algorithm faster than the trivial exponential bound $O^*(2^n)$ for $k \geq 1$ (n is number of vertices in the input graph).

In terms of existing algorithms for the maximum k-defective clique problem, Russian Doll Search (RDS) [Verfaillie *et al.*, 1996], a tree enumeration algorithm which solves nested subproblems is perhaps one of the most intensely studied algorithms. RDS was first studied in [Östergård, 2002] for the $k = 0$ case. Then, it was used in [Trukhanov *et al.*, 2013] and further improved in [Gschwind *et al.*, 2016] for $k \geq 1$. Apart from RDS, there is also a considerable body of work using integer programming methods for solving the $k = 0$ case such as [Rebennack *et al.*, 2011; Pardalos and Rodgers, 1992]. In contrast, when $k \geq 1$, the only known integer formulation was introduced in [Sherali and Smith, 2006]. In this work, several classes of facetial inequalities (i.e., the most tightened inequalities) of the k-defective clique polytope are investigated. However, as we illustrate in the experimental part of this paper, existing RDS and integer programming approaches have difficulties in solving large

instances.

Our contributions. We study efficient exact algorithms for the maximum k-defective problem in both theory and practice. Firstly, a basic branch-and-search algorithm for the maximum k-defective clique problem, MADEC, is proposed. We show that MADEC runs in $O^*(\gamma_k^n)$ (the O^* notation hides polynomial factors) where n is the number of vertices in the given graph and γ_k is a value related to k (see Table 1 for instances). To the best of our knowledge, it is the first non-trivial theoretical time bound for the exact algorithms of the maximum k-defective clique problem for $k \geq 1$.

Table 1: Some values of γ_k.

$k =$	1	2	3	4	5
$\gamma_k =$	1.9276	1.9836	1.9960	1.9990	1.9996

Moreover, we study speedup techniques for MADEC for solving realistic large real-life instances by exploiting decent properties of the problem. We carry out empirical studies of the algorithms, demonstrating that, for a large number of instances, MADEC$^+$ improves the best-known RDS and integer programming methods by more than an order of magnitude. Further experiments also shed light on the efficiency of our proposed ideas.

2 Problem statement

Let $G = (V, E)$ be a simple and undirected graph, where V and E are the vertex and edge set, respectively. The *complement graph* of G is denoted by $\overline{G} = (V, \overline{E})$. For a vertex set $S \subseteq V$, $G[S]$ is the *subgraph induced by* S, $E(S)$ is then the edge set of $G[S]$. The graph $G[V \setminus S]$ is also denoted by $G - S$. For any vertex v, the set of *neighbor vertices* of v in G is denoted by $N(v)$, in $\overline{G}$ by $\overline{N}(v)$. For convenience, let us also abbreviate $N(v) \cap S$ to $N_S(v)$ and $\overline{N}(v) \cap S$ to $\overline{N}_S(v)$. Moreover, a vertex set $S \subseteq V$ is a *clique* if $G[S]$ is a complete graph, S is an *independent set* if $\overline{G}[S]$ is an empty graph.

The *distance* between two connected vertices u and v in G, which is the shortest length of path between u and v in G, is denoted by $dist_G(u, v)$. The *diameter* of graph G, $diam(G[S])$, is defined as the maximum distance among all the pairs of vertices in G.

The k-defective clique Let $G = (V, E)$ be a given graph, k be a non-negative integer. A k-defective clique of G is a vertex set $S \subseteq V$ such that $G[S]$ contains at least $\binom{|S|}{2} - k$ edges, or in another word, there are no more than k edges in $\overline{G}[S]$. The maximum k-defective clique problem, is to find the k-defective cliques with maximum size. In this paper, we consider a more general problem.

The constrained k-defective clique problem
Input: a graph $G = (V, E)$, a vertex set $P \subseteq V$ and an integer $k \geq 1$.
Objective: find a maximum k-defective S from G such that $P \subseteq S$.

A maximum k-defective clique S such that $P \subseteq S$ is a *solution* to the input instance $I = (G, P, k)$. When $P = \emptyset$, the constrained k-defective clique problem is equal to the original maximum k-defective clique problem.

Property 1 (Hereditary [Pattillo *et al.*, 2013])**.** *If S is a k-defective clique of G, then any subset of S is also a k-defective clique.*

This property is equivalent to say that k-defective clique has *hereditary property* in graph theory. Since the k-defective clique is also *non-trivial* (i.e., not every subset of vertices is a k-defective clique), *interesting* (i.e., there are arbitrary large vertex set being a k-defective clique), the maximum k-defective clique problem is NP-hard by the theorem of [Yannakakis, 1978].

Property 2. *Let S be a k-defective clique of G, $n = |S|$. The following statements hold.*

1. *If $k = 0$, then $diam(G[S]) = 1$.*
2. *If $k \geq 1$, $n \geq \frac{3}{2} + \sqrt{2k-2}$ and $G[S]$ is a connected graph, then $diam(G[S]) \leq \lfloor \frac{(2n+1)-\sqrt{4n^2-12n+17-8k}}{2} \rfloor$.*

Proof. The first statement holds because $G[S]$ is a complete graph when $k = 0$.

For the second statement, let us first abbreviate $diam(G[S])$ to d. Now, suppose the shortest path between two vertices p_0 and p_d in $G[S]$ is $\{\{p_0, p_1\}, \{p_1, p_2\}, \ldots, \{p_{d-1}, p_d\}\}$ where d is the length of the path. We then partition S into d subsets $\{D_0, ..., D_d\}$, each set D_i ($i = 0, ..., d$) containing vertices whose distance to p_0 is exactly i, i.e., $D_i = \{u \in S | dist_{G[S]}(u, p_0) = i\}$. Then, we have $|D_i| > 0$ for $i = 0, ..., d$, and more importantly, the neighbors of each vertex in D_i ($i > 0$) can be only in D_{i-1}, D_i, D_{i+1}. With this in mind, we can reduce $G[S]$ to a graph $G'[S']$ so that each vertex and edge of $G[S]$ have a unique copy in $G'[S']$. The procedure is illustrated in Figure 1. The new graph, $G'[S']$, has a special structure that we can easily bound the number of its edges as follows.

$$\begin{cases} \binom{n}{2} - k \leq |E'| \leq \binom{n-d}{2} \text{ if } d = 2, \\ \binom{n}{2} - k \leq |E'| \leq \binom{n-d}{2} + 2(n-d) \text{ if } d > 2. \end{cases}$$

Without too much effort, one can reduce above inequalities to $d \leq \frac{(2n+1)-\sqrt{4n^2-12n+17-8k}}{2}$ under the condition that $n \geq \frac{3}{2} + \sqrt{2k-2}$, which completes the proof. □

The bounds of $diam(G[S])$ are further used to accelerate our algorithm in Section 4.

3 The rudimentary algorithm, MADEC

In this section, we introduce the rudimentary version of branch-and-search algorithm for the MAximum k-DEfective Clique problem, MADEC. MADEC decomposes $I = (G, P, k)$ into smaller problem instances, recursively solving each instance by finally combining their solutions into the original problem. The *reduction rules* and *branching rules* are recursively used in solving the problem instances.

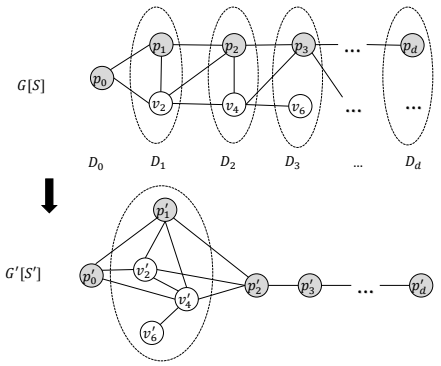

Figure 1: The procedure of constructing $G'[S']$. For each vertex $u \in S$, make a copy $u' \in S'$. For each edge $\{p_{i-1}, p_i\}$ $(i = 1, ..., d)$in the shortest path between p_0, p_d in $G[S]$, add an edge $\{p'_{i-1}, p'_i\}$ in $G'[S']$; For each $\{p_i, v\}$ where $v \in D_i$ $(i = 1, ..., d)$, add an edge $\{p'_1, v'\}$ in $G'[S']$; for each $\{p_{i-1}, v\}$ where $v \in D_i$ $(i = 1, ..., d)$, add $\{p'_0, v'\}$ in $G'[S']$; for each $\{v, p_i\}$ where $v \in D_{i-1}$ $(i = 2, ..., d)$, add $\{v', p'_2\}$ in $G'[S']$. For each of the remaining edges $\{u, w\} \in E[S]$, add an edge $\{u', w'\}$ in $G'[S']$. The number of edges in $G'[S' \setminus \{p'_0, p'_2, p'_3, \ldots, p'_d\}]$ (the subgraph in the circle) is bounded by $\binom{n-d}{2}$. At the same time, $G'[p'_2, p'_3, \ldots, p'_d]$ forms a simple path from p'_2 to p'_d.

- The reduction rules are generally used to simplify the problem instance or stop the algorithm.
- The branching rules are designed to decompose an instance into smaller subinstances without missing any solutions.

3.1 Reduction and branching rules

Let $I = (G = (V, E), P, k)$ be a given instance. The following reduction rule can be used to halt the search.

Reduction 1. *If P is not a k-defective clique, I has no solution. If V is a k-defective clique, the only solution to I is V.*

Moreover, given a vertex $v \in V \setminus P$, the following rules can reduce the scale of instance I.

Reduction 2. *If $|N(v)| = |V| - 1$, then v is in all solutions of I.*

Reduction 3. *If $|\overline{N}_P(v)| > k - |\overline{E}(P)|$, then v is not in any solution of I.*

The correctness of these rules can be simply verified by the definition of k-defective clique. Now, we turn attention to the branching rules. Assume $I = (G = (V, E), P, k)$ is an instance which cannot be reduced any more. Let us separate $V \setminus P$ into two sets, the set of *Defective Candidates*, $C^+ = \{v \in V \setminus P : |\overline{N}_P(v)| > 0\}$ and the set of *Non-defective Candidates*, $C^- = \{u \in V \setminus P : |\overline{N}_P(u)| = 0\}$ (see Figure 2 for example).

Apply Branching Rule 1 when $|\overline{E}(P \cup C^+)| > k$. If $|\overline{E}(P \cup C^+)| > k$, C^+ is a non-empty set, otherwise P is not k-defective clique, contradicting to the assumption that I is not reduced by Reduction Rule 1. Suppose

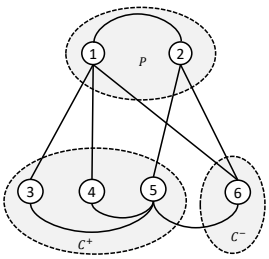

Figure 2: An eaxmple of instance $I = (G, P, k)$, $k = 3$. Note that $\overline{E}(P \cup C^+) > k$.

$C^+ = \{v_1, v_2 \cdots, v_m\}$ where m is the number of defective candidates. Let p be the largest index such that $P \cup \{v_1, ..., v_p\}$ is still a k-defective clique. For example, in Figure 2, $p = 3$ since $P \cup \{3, 4\}$ is a 3-defective clique but $P \cup \{3, 4, 5\}$ not. Note that p can be at most $m - 1$, otherwise $P \cup C^+$ is a k-defective clique. By Branching Rule 1, I is decomposed I into $p + 1$ smaller instances.

- In the first instance, v_1 is deleted from G,
- for $i \in \{2, ..., p + 1\}$, in the ith instance, $\{v_1, ..., v_{i-1}\}$ is moved to P and v_i is deleted from G.

Let us denote the subinstances as $Br1(I, 1), ..., Br1(I, p + 1)$ respectively. Take Figure 2 as an example. By this rule, four instances, $Br1(I, 1) = (G - \{3\}, P, 3)$, $Br1(I, 2) = (G - \{4\}, P \cup \{3\}, 3)$, $Br1(I, 3) = (G - \{5\}, P \cup \{3, 4\}, 3)$ are generated.

Apply Branching Rule 2 when $|\overline{E}(P \cup C^+)| \leq k$. In this case, the first rule may not work as C^+ could be an empty set, however, C^- is a non-empty set, otherwise $P \cup C^+$ is a k-defective clique and reduced by Reduction Rule 1. By Branching Rule 2, we randomly take a vertex, say u, from C^- and generate two sub-instances as follows.

- In the first instance, u is deleted from G,
- in the second instance, u is moved from C^- to P.

For the convenience of further explanation, let us denote the two instances as $Br2(I, 1)$ and $Br2(I, 2)$.

3.2 The whole algorithm

With the above rules, we present the whole algorithm. Given an instance $I = (G = (V, E), P, k)$, $MADEC(I)$ works as follows.

1. If P is not a solution, return an empty set as a solution. If V is a k-defective, return V as the solution. Otherwise, go to the next step.
2. Update I by reducing vertices from $V \setminus P$ by Reduction Rule 2 and 3. Go to the next step.
3. If I satisfies $|\overline{E}(P \cup C^+)| > k$, generate $\mathcal{I} = \{Br1(I, 1), ..., Br1(I, p + 1)\}$ by Branching Rule 1. Otherwise, generate $\mathcal{I} = \{Br2(I, 1), Br2(I, 2)\}$ by Branching Rule 2. Return $\arg\max_{|P|}\{P \leftarrow MADEC(I'), \forall I' \in \mathcal{I}\}$.

Complexity analysis Usually, the time analysis of branch-and-search algorithms is based on upper bounding the size of *search tree* generated by the algorithm. Assume the size of instance I is measured by the parameter n. We use $T(n)$ to denote the maximum number of leaves in the search tree generated by the instance with size at most n. In order to bound $T(n)$, for a branching rule where I is decomposed into l subinstances such that in the ith subinstance, the size of I decreases at least a_i, we obtain a recurrence relation

$$T(n) \leq T(n-a_1) + \ldots + T(n-a_l)$$

The largest real root of function $f(x) = 1 - \sum_{i=1}^{l} x^{-a_i}$ is called *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 the instance with size n is given by $O^*(\gamma^n)$. For interested readers, we refer to the monograph of [Fomin and Kratsch, 2010] for more details.

Let $I = (G = (V, E), P, k)$ be an input instance of MADEC, $n = |V \setminus P|$. We first analyze the branching factor, σ_1, if only Branching Rule 1 is employed to generate the search tree. For $i \in \{1, ..., p+1\}$, the size of ith subinstance, $Br1(I, i)$, is decreased by i. Thus, we obtain the following recurrence for this branching operation

$$T(n) \leq T(n-1) + \ldots + T(n-(p+1)) \qquad (1)$$

where $p \leq k$. In the worst case, $p = k$, the branching factor of this recurrence relation reaches maximum, which is the largest root of function

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

On the other hand, if only Branching Rule 2 is applied in the algorithm, we obtain the following recurrence relation likewise.

$$T(n) \leq T(n-1) + T(n-1).$$

The branching factor of this recurrence relation is 2, which leads to the trivial time bound $O^*(2^n)$. However, this bound can be tightened by expanding the recurrence relation.

Assume an instance $I = (G = (V, E), P, k)$, C^+ and C^- are defective and non-defective candidates of I. Suppose I satisfies $|\overline{E}(P \cup C^+)| \leq k$. We mimic the procedure of generating the search tree by applying Branching Rule 2 whenever possible.

In the first step, Branching Rule 2 is first applied, resulting two instances $Br2(I, 1)$ and $Br2(I, 2)$. Let us also denote $Br2(I, 2) = (G' = (V', E'), P', k)$, C'^+ and C'^- as the defective and non-defective candidate set of $Br2(I, 2)$, respectively. By the definitions of C^- and Branching Rule 2, P' includes a new vertex u which is adjacent to all vertices in P, but not all vertices in $V \setminus P$. As a result, $C^+ \subset C'^+$ and we have $|\overline{E'}(P' \cup C'^+)| > |\overline{E}(P \cup C^+)|$.

Suppose we recursively use Branching Rule 2 to decompose I. As $|\overline{E}(P \cup C^+)|$ is bounded by k, following the conclusion above, instance $I_2^q = \underbrace{Br2(...Br2(Br2}_{q\ times}(I, 2), 2) \ldots, 2)$ $(q \leq k+1)$ will not satisfy the condition of applying Branching Rule 2. In the worst case where $q = k+1$, we have to turn to Branching Rule 1 for I_2^{k+1} if I_2^{k+1} can not be reduced by reduction rule. Figure 3 shows the whole procedure.

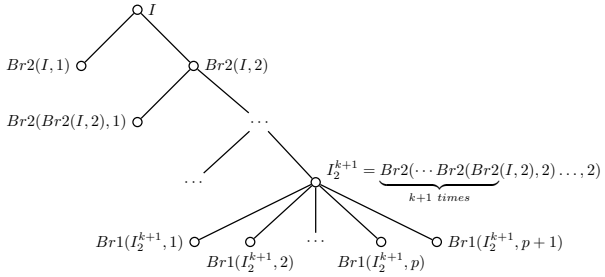

Figure 3: The search tree of applying Branching Rule 2 in the worst case

According the above analysis, we can rewrite the recurrence relation of Branching Rule 2 as

$$T(n) \leq T(n-1) + \ldots + T(n-(k+1)) + \ldots + T(n-(k+1)-(p+1))$$

where $p \leq k$. When $p = k$, the branching factor reaches the maximum, which is the largest root of function

$$x^{2k+3} - 2x^{2k+2} + 1 = 0. \qquad (2)$$

As a result, we draw the following conclusion.

Theorem 1. *MADEC($I = (G = (V, E), \emptyset, k)$) runs in $O^*(\gamma_k^n)$ where $n = |V|$ and γ_k is the largest real root of Function (2).*

Table 1 presents some values of γ_k when k is small. As far as we know, it is the first algorithm that breaks the trivial exponential bound of 2^n when $k \geq 1$.

4 Improve MADEC with speedup rules

Though MADEC enjoys a good time complexity, its experimental performance is not satisfying, especially for large real-life networks. We introduce stronger reduction rules to boost its performance.

To make use of the following rules, we need to maintain a lower bound, lb, of the solutions to the input problem. lb can be initialized by a fast heuristic algorithm before the start of the exact search, and updated when a better lower bound is found in the search. As far as we know, there is a considerable number of heuristic algorithms for the maximum clique problem (see [Wu and Hao, 2015]), but none for the maximum k-defective clique. A natural idea is to use the existing heuristic to find a maximal clique, then expand the clique by adding vertices of maximum degree, ties breaking randomly. In our implementation, we make use of the clique heuristic in [Rossi *et al.*, 2014] which is quite successful in dealing with large graphs. The complexity of this heuristic is $O(|E|)$.

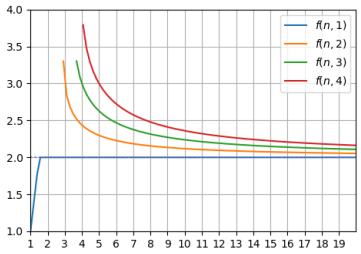

Figure 4: The curve of $f(n,k)$ with $k = \{1,2,3,4\}$

4.1 Reduction by degrees

When we have a lower bound of the solution, we can reduce vertices by their degrees. The first reduction rule is employed as follows.

Reduction 4. *Given an instance $I = (G = (V,E), P, k)$, a lower bound lb, for any vertex $v \in V$, if $|N(v)| \leq lb - k - 1$, then any solution to I of size larger than lb does not contain v.*

This rule can be used more than only in the exact search. A preprocessing procedure, *PEEL*, can be deployed to reduce the scale of the graph based on this rule. PEEL recursively removes vertices with degree equal or less than $lb - k - 1$. The procedure can be accomplished in $O(|E|)$.

Another degree based reduction rule also holds with a lower bound lb.

Reduction 5. *Given an instance $I = (G = (V,E), P, k)$, a lower bound lb, for any vertex $v \in V \setminus P$, if $|N(v)| \leq lb - k - 1 - |\overline{E}(P)|$, then any solution to I of size larger than lb does not contain v.*

The above two rules can be seen as extensions of Reduction Rules 2 and 3.

4.2 Reduction by low diameter

Property 2 can be also used to design reduction rules. Let us define $f(n,k) = \frac{(2n+1)-\sqrt{4n^2-12n+17-8k}}{2}$. As shown by Figure 4, when $n \geq \frac{3}{2} + \sqrt{2k-2}$, for all $k \in \{1,2,3,4\}$, $f(n,k)$ is monotone non-increasing as n increases. Indeed, this observation can be generalized to any $k \geq 1$. On the other hand, if a k-defective clique S satisfies $|S| \geq k+2$, then $G[S]$ is connected and $|S| \geq \frac{3}{2} + \sqrt{2k-2}$. As as result, we can obtain the following reduction rule by Property 2.

Reduction 6. *Given an instance $I = (G = (V,E), P, k)$, a lower bound $lb \geq k+2$, for any vertex $v \in V \setminus P$, if $\exists u \in P$ such that $dist_G(v,u) > \lfloor f(n,k) \rfloor$, then any solution to I of size larger than lb does not contain v.*

However, this reduction rule is not efficient in practice since there are considerable computational overheads of computing $dist_G(v,u)$. This drawback can be overcome by a simple modification to the reduction rule. Again, as shown by Figure 4, for $k \in \{1,...,4\}$, when $n \geq k+2$, $\lfloor f(n,k) \rfloor \equiv 2$. Without too many efforts, one can prove that for any k-defective clique S with n vertices, if $n \geq k+2$, the diameter of $G[S]$ is bounded by 2. Therefore, we present the following reduction rule.

Reduction 7. *Given an instance $I = (G = (V,E), P, k)$, a lower bound $lb \geq k+2$, for any vertex $v \in V \setminus P$, if $\exists u \in P$ such that $v \notin N(u)$ and $v \notin \bigcup_{w \in N(u)} N(w)$, then any solution to I of size larger than lb does not contain v.*

4.3 Graph coloring upper bounding

A well-known property of clique model is that the chromatic number of a graph is an upper bound of the maximum clique [Wu and Hao, 2015; San Segundo *et al.*, 2011; Rossi *et al.*, 2014]. Since the maximum k-defective clique is a generalized version of the maximum clique problem, we also generalized this property.

Property 3 (color-bound)**.** *If graph $G = (V,E)$ can be c-colored, namely, V is partitioned by c disjoint independent sets $V_1, \ldots, V_c$, then $\sum_{i=1}^{c} \min(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |V_i|)$ is an upper bound of the size of a maximum k-defective clique in G, called color-bound of k-defective clique.*

Proof. Suppose that S^* is a maximum k-defective clique of G and $V_1, \ldots, V_c$ partitions S^* into $S_1, \ldots, S_c$ where $S_i \subseteq V_i$, respectively. By Property 1, S_i is a k-defective clique. Meanwhile, $G[S_i]$ is an empty graph since S_i is an independent set. Hence, the size of maximum k-defective clique in $G[S_i]$ is $\min(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |V_i|)$, indicating that $|S^*| = \sum_{i=1}^{c} |S_i| \leq \sum_{i=1}^{c} \min(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |V_i|)$. □

When $k = 0$, Lemma 3 indicates that c is an upper bound of the maximum clique size. To estimate a valid upper bound of the maximum s-defective clique of graph $G = (V,E)$, we adopt a fast coloring procedure in [San Segundo *et al.*, 2011] which is used to solve the maximum clique problem. This procedure, denoted by $ColorBound(G,k)$, includes the following steps.

1. Initialize the color number c as 1.
2. Open an empty color class (set) V_c, build V_c iteratively. For each iteration, find a vertex in V which is not adjacent to any vertices in V_c, then move the vertex to V_c, repeat the above operations until no such vertex exists.
3. Remove vertices of V_c from V. If V is not empty, increase c by 1 and go back to Step 2. Otherwise, return $\sum_{i=1}^{c} \min(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |V_i|)$ as a color-bound of G.

The time complexity of $ColorBound(G,k)$ is bounded by $O(|V|^3)$. However, it is very fast in practice, especially employing the bit-parallel technique [San Segundo *et al.*, 2011] which accelerates Step 2. By Lemma 3, the following reduction rule holds.

Reduction 8. *Given an instance $I = (G = (V,E), P, k)$, a lower bound lb, if $|P| + ColorBound(G[V \setminus P], k) \leq lb$, then there is no solution to I of size larger than lb.*

4.4 The improved algorithm, MADEC$^+$

Putting all together, we end up with MADEC$^+$ as presented in Algorithm 1. MADEC$^+$ starts from a heuristic search for the purpose of initializing a good lower bound, then enters the branch-and-search procedure, $BBSearch$. $BBSearch$ works like MADEC except that more speedup reduction rules are applied. The lower bound, lb, is defined as a global variable which is consistently updated in the $BBSearch$. The ones who are interested in the set of maximum k-defective cliques can also record the solutions in company with lb.

Algorithm 1: The maximum k-defective clique algorithm: MADEC$^+$

```
1  MADEC+(G, k)
2  begin
3      lb ← HeuristicSearch(G, k)
4      G ← Peel(G, lb)
5      BBSearch(I = (G, ∅, k))
6  return lb
7
8  BBsearch(I = (G = (V, E), P, k))
9  begin
10     if P is not a k-defective clique or |V| ≤ lb or
       ColorBound(G[V \ P], k) + |P| ≤ lb then
11         return
12     if |P| ≥ lb then
13         lb ← |P|
14     if ∃v ∈ V \ P such that v is reducible by Rule 2
       then
15         BBSearch((G, P ∪ {v}, k))
16     else if ∃v ∈ V \ P such that v is reducible Rule3
       or 4 or 5 or 7 then
17         BBSearch((G − {v}, P, k))
18     else if Ē(P ∪ {C+}) > k then
19         ∀I' ∈ {Br1(I, 1), ..., Br1(I, p + 1)}
           generated by Branching Rule 1, Call
           BBSearch(I')
20     else
21         ∀I' ∈ {Br2(I, 1), Br2(I, 2)} generated by
           Branching Rule 2, Call BBSearch(I')
22 return
```

If the lb keeps 1 before the end of the algorithm, MADEC$^+$ degenerates to the original MADEC. Therefore, the complexity of MADEC$^+$ is the same as MADEC (see Section 3.2).

5 Computational experiments

In this section, we evaluate MADEC and MADEC$^+$ by experiments. The codes are written in C++ and compiled by g++ with optimization option '-O2'. All the experiments are conducted on a computer with an AMD Opteron 4184 processor (2.8 GHz and 2 GB RAM) running CentOS 6.5.

We also compare MADEC and MADEC$^+$ with two best-known algorithms, the Russian Doll Search algorithm (RDS) in [Gschwind *et al.*, 2016] and the GNU Linear Programming Kit (GLPK) with the formulation given by [Sherali and Smith, 2006]. Note that we optimized the implementation of RDS so that the performance can match that reported in [Gschwind *et al.*, 2016] [1].

Table 2: Computational time of all algorithms

graph name ($\|V\|$, $\|E\|$)	k	LB	opt	Running time in seconds			
				MADEC$^+$	MADEC	RDS	GLPK
football.graph	1	9	9	**0.05**	0.19	0.17	193.43
	2	9	9	**0.22**	0.68	1.01	189.7
(115, 613)	3	9	9	**0.52**	4.35	6.6	365.55
	4	9	9	**0.85**	7.67	12.97	297.26
cond-mat.graph	1	18	18	0.0	0.0	0.0	0.0
	2	18	18	0.0	0.0	0.0	0.0
(16726, 47594)	3	18	18	**0.03**	0.6	0.62	2.32
	4	18	18	**0.29**	11.69	8.67	75.78
cond-mat-2005.graph	1	30	30	0.0	0.0	0.0	0.0
	2	30	30	0.0	0.0	0.0	0.0
(40421, 175691)	3	30	30	0.0	0.0	0.0	0.0
	4	30	30	**0.03**	10.79	5.31	0.27
rgg_n_2_17_s0.graph	1	15	15	0.0	0.0	0.0	0.09
	2	15	16	**0.04**	2.03	0.5	357.41
(131072, 728472)	3	15	16	**5.91**	9897.15	2114.12	X
	4	15	16	**159.9**	>18k	>18k	X
cond-mat-2003.graph	1	25	25	0.0	0.0	0.0	0.0
	2	25	25	0.0	0.0	0.0	0.0
(31163, 120029)	3	25	26	**0.02**	0.82	0.57	0.26
	4	25	26	**0.31**	18.21	12.23	9.19
email.graph	1	12	12	0.0	0.0	0.0	0.0
	2	12	12	**0.35**	1.24	1.1	183.47
(1133, 5451)	3	12	12	**6.8**	72.95	65.96	X
	4	12	13	**14.96**	472.33	197.87	X
astro-ph.graph	1	56	57	0.72	0.1	**0.06**	5.11
	2	56	57	1.63	3.37	**1.12**	5.36
(16706, 121251)	3	56	57	**25.39**	281.78	70.22	253.23
	4	56	57	**107.76**	3729.68	735.55	252.95
rgg_n_2_19_s0.graph	1	18	19	0.0	0.0	0.0	0.0
	2	18	19	**0.02**	0.15	0.15	3.11
(524288, 3269202)	3	18	19	**0.15**	49.4	53.86	4976.02
	4	18	20	**0.51**	2143.04	1081.19	X
web-Google.txt	1	44	45	0.1	0.64	**0.04**	6.16
	2	44	46	**0.59**	88.48	7.45	2566.6
(875713, 4322051)	3	44	46	**0.61**	535.46	84.55	X
	4	44	47	**0.61**	3562.65	753.61	X
dolphins.graph	1	5	6	0.02	0.0	0.0	0.89
	2	5	6	**0.04**	0.05	0.05	2.48
(62, 159)	3	5	6	**0.1**	0.21	0.48	6.49
	4	5	7	**0.1**	0.55	0.53	11.92
adjnoun.graph	1	5	6	0.15	**0.03**	0.06	17.44
	2	5	6	0.66	**0.21**	0.5	43.02
(112, 425)	3	5	7	**1.43**	2.3	3.54	94.81
	4	5	7	**2.91**	4.7	8.56	186.88
polbooks.graph	1	6	7	0.06	0.05	**0.03**	12.31
	2	6	7	**0.35**	0.41	0.55	94.93
(105, 441)	3	6	8	**0.85**	2.81	3.48	133.03
	4	6	8	**1.84**	5.33	5.81	153.07
power.graph	1	6	6	0.0	0.0	0.0	0.0
	2	6	6	0.03	**0.02**	0.05	1.03
(4941, 6594)	3	6	7	**0.17**	47.03	55.4	X
	4	6	7	**21.04**	>18k	>18k	X
cnr-2000.graph	1	84	85	0.0	0.0	0.0	0.0
	2	84	85	0.02	0.0	0.0	0.0
(325557, 2738969)	3	84	86	0.11	0.0	0.0	0.0
	4	84	86	**0.53**	5814.22	1009.61	14.5
rgg_n_2_20_s0.graph	1	17	18	0.68	2.59	**0.18**	1532.32
	2	17	18	**4.13**	114.71	42.01	X
(1048576, 6890893)	3	17	18	**123.19**	>18k	>18k	X
	4	17	19	**3114.67**	>18k	>18k	X

We test all algorithms on 44 graphs which are also used in the literature of relaxed clique algorithms [Trukhanov *et al.*, 2013; Gschwind *et al.*, 2016]. These graphs are included in two well-known benchmark sets, the Stanford Large Network Dataset Collection (SNAP) and the 10^{th} DIMACS Implementation Challenge, which are collected from real-world applications. For each graph G, we run 4 instances, i.e., $(G, \emptyset, k), k \in \{1, 2, 3, 4\}$. For the sake

[1] The original implementation is not accessible. Our implementation is published at https://github.com/Ijcai2019Paper3402/ijcai2019-paper3402.

of space, we report the results of 15 representing graphs. The other graphs are omitted since all associated instances can be either solved within 5 seconds by all algorithms or cannot be solved by any tested algorithm in 5 hours.

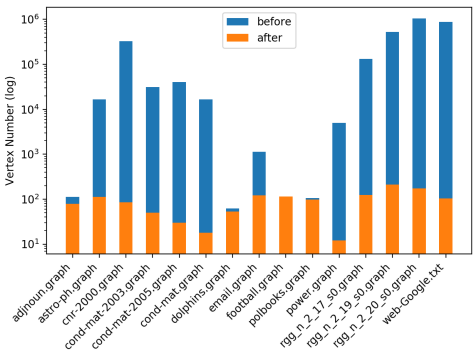

Figure 5: The number of vertex (in log scale) before and after preprocessing.

To compare the algorithm effectively, we use PEEL (Section 4.1) before the start of all the algorithms. With a lower bound obtained by our heuristic, PEEL is powerful in reducing the size of graphs. Figure 5 shows the vertex number (in log scale) before and after the preprocessing.

In Table 2, we show the computational time for each instance independently. The first column gives information about graphs. From the second to the last column, we present the lower bound found by the heuristic search, the optimal solution, the computational times of the corresponding algorithm successively. >18k and X indicate that the algorithm fails in 5 hours and by a lack of memory, respectively . The bold values mark the shortest time of one instance. Clearly, MADEC$^+$ runs faster than others for 37 instances. More impressively, for 22 instances, MADEC$^+$ achieves a speedup of more than an order of magnitude over RDS and GPLK.

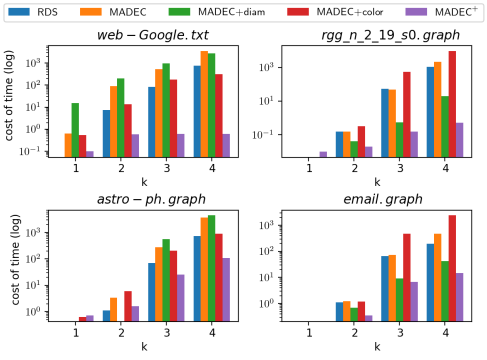

Figure 6: The comutational times in seconds (in log scale) for 4 large instances.

To get a deeper understanding of proposed ideas, we compare RDS, MADEC, MADEC$^+$ with two reference algorithms which are MADEC+diam, the MADEC with lower diameter reduction (Reduction Rule 6) and MADEC+color, the MADEC with color-bound (Reduction Rule 8). Figure 6 and 7 show the time consumption and the number of nodes in the search tree, respectively. Without exception, MADEC$^+$ almost dominates others in terms of the search time and the tree size. However, the computational times of MADEC+diam and MADEC+color are longer than that of MADEC on certain instances, even though the tree sizes are smaller, indicating that a single reduction rule does not always improve the overall performance in practice.

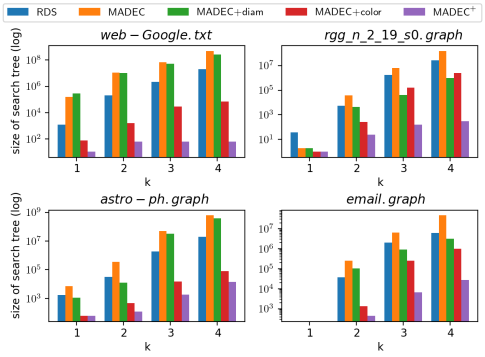

Figure 7: The number of tree nodes (in log scale) for 4 large instances.

6 Conclusion

In this paper, we design, MADEC, an exact algorithm for the maximum k-defective problem with running time better than trivial bound $O^*(2^n)$. We also upgrade MADEC to MADEC$^+$ with regard to practical performance. Numerical experiments show the superiority of MADEC$^+$ over other algorithms on well-known SNAP and 10th DIAMCS graphs.

For future work, from the algorithmic perspective, the success of this algorithm inspire us to optimize solution methods for other relaxed clique problem, from the practical perspective, the algorithms can be extended to applications in the social network, bio-informatics as mentioned in the introduction part.

References

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

[Chen *et al.*, 2004] Yuanzhu Chen, Arthur Liestman, and Jiangchuan Liu. Clustering algorithms for ad hoc wireless networks. *Ad hoc and sensor networks*, 28:76, 2004.

[Cheng *et al.*, 2012] James Cheng, Linhong Zhu, Yiping Ke, and Shumo Chu. Fast algorithms for maximal clique enumeration with limited memory. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 1240–1248. ACM, 2012.

[Conte *et al.*, 2018] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. D2k: Scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1272–1281. ACM, 2018.

[Fomin and Kratsch, 2010] Fedor V Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.

[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 *IJCAI*, pages 1449–1455, 2018.

[Gschwind *et al.*, 2016] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. Maximum weight relaxed cliques and russian doll search revisited. *Discrete Applied Mathematics*, 2016.

[Östergård, 2002] Patric RJ Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[Pajouh *et al.*, 2016] Foad Mahdavi Pajouh, Balabhaskar Balasundaram, and Illya V Hicks. On the 2-club polytope of graphs. *Operations Research*, 64(6):1466–1481, 2016.

[Pardalos and Rodgers, 1992] Panos M Pardalos and Gregory P Rodgers. A branch and bound algorithm for the maximum clique problem. *Computers & operations research*, 19(5):363–375, 1992.

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

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

[Rebennack *et al.*, 2011] Steffen Rebennack, Marcus Oswald, Dirk Oliver Theis, Hanna Seitz, Gerhard Reinelt, and Panos M Pardalos. A branch and cut solver for the maximum stable set problem. *Journal of combinatorial optimization*, 21(4):434–457, 2011.

[Rossi *et al.*, 2014] Ryan A Rossi, David F Gleich, Assefaw H Gebremedhin, and Md Mostofa Ali Patwary. Fast maximum clique algorithms for large graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 365–366. ACM, 2014.

[San Segundo *et al.*, 2011] Pablo San Segundo, Diego Rodríguez-Losada, and Agustín Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2):571–581, 2011.

[Sherali and Smith, 2006] Hanif D Sherali and J Cole Smith. A polyhedral study of the generalized vertex packing problem. *Mathematical Programming*, 107(3):367–390, 2006.

[Sherali *et al.*, 2002] Hanif D Sherali, J Cole Smith, and Antonio A Trani. An airspace planning model for selecting flight-plans under workload, safety, and equity considerations. *Transportation Science*, 36(4):378–397, 2002.

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

[Veremyev *et al.*, 2016] Alexander Veremyev, Oleg A Prokopyev, Sergiy Butenko, and Eduardo L Pasiliao. Exact mip-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications*, 64(1):177–214, 2016.

[Verfaillie *et al.*, 1996] Gérard Verfaillie, Michel Lemaître, and Thomas Schiex. Russian doll search for solving constraint optimization problems. In *Proceedings of AAAI/IAAI*, pages 181–187, 1996.

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

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

[Yannakakis, 1978] Mihalis Yannakakis. Node-and edge-deletion np-complete problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 253–264. ACM, 1978.

[Yu *et al.*, 2006] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006.