Computing maximum k-defective cliques in massive graphs

Xiaoyu Chen [a], Yi Zhou [a,*], Jin-Kao Hao [b,c],Mingyu Xiao [a],

[a]*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China*

[b]*LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[c]*Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France*

***Computers and Operations Research*, Oct. 2020.**
https://doi.org/10.1016/j.cor.2020.105131

Abstract

A k-defective clique (k is a non-negative integer) of an undirected graph G is a subset of its vertices, which induces a nearly complete graph with a maximum of k missing edges. The maximum k-defective clique problem (MDCP) is to determine the k-defective clique of the maximum size in the graph. As a relaxation of the popular maximum clique problem, the MDCP is a relevant model for a number of practical applications such as complex network analysis. However, it is computationally challenging to solve the problem. In this study, we investigate a set of general and dedicated graph reduction and pruning techniques to improve exact search algorithms based on the branch-and-bound framework. We present results of extensive computational experiments on 141 benchmark graphs from several popular sources, including both random graphs and massive real-world networks. Comparisons with two state-of-the-art methods in the literature demonstrate that our approach is on par with the reference methods and performs remarkably well on massive graphs.

Keywords: Relaxed clique; Exact search; Branch-and-bound; Massive graphs; Graph reduction.

* Corresponding author.
Email addresses: `zhou.yi@uestc.edu.cn` (Yi Zhou), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao), `myxiao@gmail.com` (Mingyu Xiao).

1 Introduction

For an undirected graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq \binom{V}{2}$, a *clique* S of G is a subset of V such that S induces a complete subgraph $G[S]$ of G, i.e., $\forall u, v \in S, \{u, v\} \in E$. A clique is *maximum* if its cardinality is the largest among all the cliques of the graph. The *maximum clique problem* (MCP) is to obtain the maximum clique of a specific graph. The MCP is a general and convenient graph model that can be used to identify fully connected structures in numerous application settings. However, it is restrictive for many other practical problems such as complex network analysis, where dense (not necessarily fully connected) structures are of particular interest [12]. Hence, several generalizations or relaxations of the conventional clique model have been proposed, including the *k-plex* [3], *k-club* [4], *quasi-cliques*[1], *bicliques* [26], and *k-defective clique* [25].

In this study, our primary interest is in the *maximum k-defective clique* problem. Formally, a *k-defective clique* (k is a non-negative integer) of a graph G is a subset of vertices $S \subseteq V$, such that S induces a subgraph $G[S]$ of G with at least $\binom{|S|}{2} - k$ edges. The *maximum k-defective clique problem* (MDCP) is to obtain a k-defective clique with the maximum cardinality in a graph. It is clear that the popular maximum clique problem is a special case of the MDCP when $k = 0$.

To our best knowledge, the concept of k-defective clique was first introduced by Yu et al. [25]. They used large k-defective cliques to predict missing edges between two proteins in a biological network, where nodes represent proteins and edges represent protein interactions [25]. Other applications of the MDCP exist in transportation science, e.g., Sherali *et al.* used the maximum k-defective clique model to solve an airspace planning problem [17,18]. However, the MDCP is a computationally challenging problem as from the complexity viewpoint, its decision version can be shown as NP-complete by the theorem of [24].

1.1 Existing algorithms for MDCP

Unlike the conventional MCP problem for which many solutions have been investigated [22], our literature review shows that only a few algorithms have been proposed for solving the MDCP. Specifically, in [20], Trukhanov et al. presented the first exact algorithm for the MDCP, which was based on the Russian doll search (RDS) [21]. In principle, the RDS algorithm operates as follows. First, the n vertices of a graph are sorted by *degeneracy order* which is a sequence $(v_1, ..., v_n)$ $(n = |V|)$ such that every vertex v_i has the smallest degree in the subgraph induced by $\{v_{i+1}, ..., v_n\}$. Subsequently, a series of n nested subproblems are successively solved through n rounds of the search, whereas the optimal objective values of the solved subproblems are recorded and used to obtain better bounds. Specifically, at the ith round with i traversing from n to 1, the subproblem, which requires v_i in the solution and $\{v_i, ..., v_n\}$ potentially in the solution, is solved via a simple branch-and-bound algorithm. The optimal solution to the initial graph is thus obtained at the last round of the search. In [19] and [6], the RDS algorithm of [20] are further improved with new pre-processing rules and a better implementation.

Integer linear programming (ILP) is another well-known approach for solving the MCP [11,14] and MDCP [18]. For an undirected graph $G = (V, E)$, the MDCP can be formulated by the following ILP model.

$$\begin{aligned}
\text{maximize} \quad & \sum_{i \in V} x_i & & (1)\\
\text{s.t.} \quad & \sum_{\{i,j\} \in \overline{E}} z_{ij} \le k & & \\
& z_{ij} \ge x_i + x_j - 1, z_{ij} \ge 0 & \forall \{i, j\} \in \overline{E} & \\
& x_j \in \{0, 1\} & \forall j \in V, &
\end{aligned}$$

where $\overline{E}$ is the set of edges in the complement graph of G. In the formulation, each vertex $i \in V$ is associated to a binary variable x_i that indicates whether i is part of the k-defective clique. Several classes of facet-defining inequalities (i.e., the most tightened inequalities) of the k-defective clique polytope have also been investigated in [17].

To summarize, research regarding solutions for the MDCP is still new. On one hand, existing algorithms such as those in [6,18,20] only investigate partial structural properties of this problem. On the other hand, the k-defective clique model is mainly used for real-world complex network analyses (thereby resulting in massive graphs with millions of vertices and edges). To solve real-world problems, it is important to devise highly scalable algorithms.

1.2 Contributions

In this study, we solve the MDCP by investigating new and effective exact algorithms. Our main contributions can be summarized as follows.

First, we present the two structural properties of MDCP; one is the well-known heredity property, and the other is the characterization of the diameter of a subgraph induced by a k-defective clique. We also present a branch-and-bound algorithm called MADEC (MAximum k-DEfective Clique) and, for the first time, prove its worst-case time complexity.

Next, we enhance the efficiency and scalability of MADEC by investigating additional reduction, bounding, and pruning techniques. Before a search is performed, the enhanced algorithm (called MADEC$^+$) applies a fast MCP heuristic to obtain an initial lower bound of reasonable quality and a vertex degree-based preprocessing procedure to reduce the input graph. During the branch-and-bound search, MADEC$^+$ uses a *two-hop reduction* rule to exclude irrelevant vertices and a fast greedy graph coloring heuristic to upper-bound the optimal solution and hence prune the search space. To the best of our knowledge, this is the first study that uses two-hop reduction and graph coloring heuristics in a branch-and-bound algorithm for the MDCP.

Finally, we present extensive experiments to evaluate the proposed algorithms and techniques. Computational results on 141 random graphs and real-life large graphs from the literature demonstrate the competitiveness of our algorithms over existing approaches. Additional experiments were performed to demonstrate the effectiveness of the key techniques introduced herein.

The remainder of this paper is organized as follows. In the next section, we introduce the basic notations and two important properties of the MDCP. Subsequently, in Section 3, we elaborate the first version of our branch-and-bound algorithm, MADEC. In Section 4, we present our pruning techniques and the improved algorithm, MADEC$^+$. Section 5 provides the experimental assessment of the proposed algorithms. Finally, conclusions and perspectives are provided in the last section.

2 Problem statement and background

2.1 Notations

For an undirected graph $G = (V, E)$, let $S \subseteq V$ be a vertex subset of G, $u, v \in V$ two vertices and $\overline{G} = (V, \overline{E})$ the *complement graph* of G (i.e., $\{u, v\} \in \overline{E}$ if and only if $\{u, v\} \notin E$). The following notations are used herein.

$G[S] = (S, E(S))$ denotes the subgraph induced by S such that for any $x, y \in S$, $\{x, y\} \in S$ if and only if $\{x, y\} \in E$.

$N_V(v)$ and $\overline{N}_V(v)$ denote the set of *neighbor vertices* of v in G and $\overline{G}$, respectively. For simplicity, we abbreviate $N_V(v) \cap S$ to $N_S(v)$ and $\overline{N}_V(v) \cap S$ to $\overline{N}_S(v)$.

$dist_G(u, v)$ denotes the *distance* between u and v in G, which is the length of a shortest path between u and v (i.e., the number of edges in such a path).

$diam(G[S])$ denotes the *diameter* of G, which is the maximum distance among all the pairs of vertices in G.

S is an *independent set* of G if no edge links any two vertices of S in G.

2.2 Properties of k-defective clique

Property 1 (Heredity [13]) *If S is a k-defective clique of G, then any subset of S is also a k-defective clique of G.*

This simple property is important for verifying the maximality of a k-defective clique. If a k-defective clique S cannot form a larger k-defective clique with another vertex, then S is a maximal k-defective clique owing to its hereditary property. The validity of our proposed algorithms relies on this property.

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

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

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

As for the second statement, we first abbreviate $diam(G[S])$ to d ($d \geq 2$). Suppose a shortest path in $G[S]$ of maximum length is between vertices a and b and assume that it is defined by the sequence $P = (p_0, p_1, \ldots, p_d)$, where $p_0 = a$, $p_d = b$. We can partition S into $d + 1$ subsets, i.e., $D_0, \ldots, D_d$. Each D_i ($i = 0, \ldots, d$) comprises vertices whose distance to p_0 is exactly i, i.e., $D_i = \{u \in S : dist_{G[S]}(u, p_0) = i\}$. For $i = 0, \ldots, d$, it is clear that D_i is non-empty. Moreover, the neighbors of each vertex in D_i ($i = 1, \ldots, d-1$) can only be in D_{i-1}, D_i, D_{i+1}. Hence, a new graph $G' = (S, E')$ can be constructed from $G[S]$ as follows.

(1) For $i = 1, \ldots, d$, insert an edge $\{p_{i-1}, p_i\}$ in E'.
(2) For each $i = 1, \ldots, d$ and for each edge $\{p_i, v\}$ where $v \in D_i$, insert an edge $\{p_1, v\}$ in E'.
(3) For each $i = 1, \ldots, d$ and for each edge $\{p_{i-1}, v\}$ where $v \in D_i$, insert an edge $\{p_0, v\}$ in E'.

(4) For each $i = 2, \ldots, d$ and for each edge $\{v, p_i\}$ where $v \in D_{i-1}$, insert an edge $\{v, p_2\}$ in E'.
(5) For each edge $\{u, w\}$ in $G[S]$ such that neither u nor w is in path P, insert an edge $\{u, w\}$ in E'.

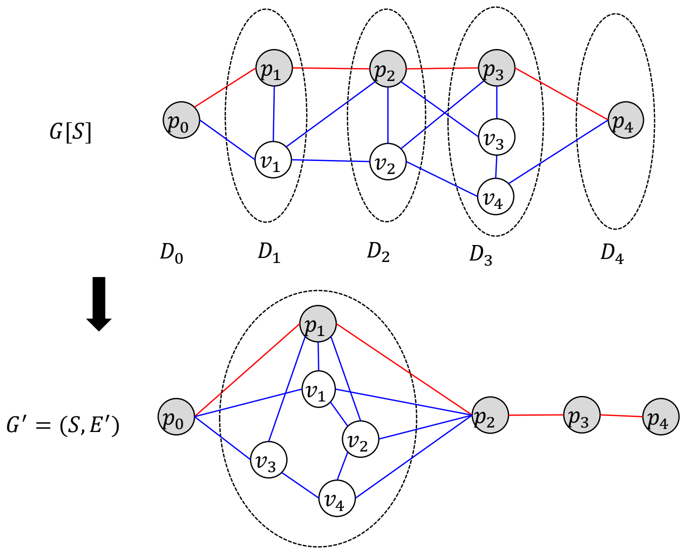

Fig. 1. Example of building $G' = (S, E')$ from $G[S]$.

Figure 1 shows an example of building graph G' from $G[S]$. Each edge in $G[S]$ is uniquely mapped to an edge in G'. The edges marked in red form the longest shortest path $(p_0, p_1, \ldots, p_4)$ in $G[S]$. The remaining edges in $G[S]$ are marked in blue. One can observe that the number of edges $G'[S \setminus \{p_0, p_2, p_3, \ldots, p_d\}]$ is bounded by $\binom{s-d}{2}$. Meanwhile, $(p_2, p_3, \ldots, p_d)$ forms a path of length $d-2$ in G'. Therefore, we can bound the number of edges in E' as follows:

$$\binom{s}{2} - k \leq |E'| \leq \binom{s-d}{2} + 2(s-d) + (d-2).$$

A simple reformulation of the above inequalities yields $d \leq \frac{(2s+1)-\sqrt{4s^2-12s+17-8k}}{2}$ under the condition $s \geq \frac{3}{2} + \sqrt{2k-2}$, which completes the proof. Λ

In Section 4, we show how these bounds of $diam(G[S])$ can be used to accelerate our algorithm.

2.3 Relationship between MCP and MDCP

The MDCP is closely related to the popular MCP, for which a considerable number of exact algorithms have been proposed [22]. For an MDCP instance defined by graph $G = (V, E)$ with $n = |V|$, $m = |E|$ and a non-negative integer k, one can solve the MDCP instance by calling a MCP algorithm for multiple rounds. In each round, one first builds a graph G' from G by adding k new edges, then calls the MCP algorithm to obtain the maximum clique in G'. It is clear that a clique in G' is also a k-defective clique in G. There are $\binom{\binom{n}{2}-m}{k}$ possibilities of building k' new edges in G. Hence, one must perform $\binom{\binom{n}{2}-m}{k}$ rounds of the MCP algorithm. Denoting $m' = \binom{\binom{n}{2}-m}{k}$, among all the m' maximum cliques in these rounds, the largest one is the solution to the MDCP instance.

In the worst case, m' can be as large as n^{2k}. Consequently, this method yields the worst-case time of $\mathcal{O}(n^{2k}MCP(n))$, where $MCP(n)$ is the run time of the underlying MCP algorithm. In practice, this solution approach becomes prohibitively time consuming when the input graph is large, even when state-of-the-art MCP algorithms discussed in [22] are used.

In this study, we investigated dedicated branch-and-bound algorithms that explore a set of problem-specific reduction, bounding, and pruning rules.

3 Basic branch-and-bound algorithm for MDCP

This section introduces a basic branch-and-bound algorithm for the MDCP, which we call the MADEC problem. Essentially, MADEC enumerates the candidate solutions by building a search tree in a depth-first manner. In each branch, MADEC solves a *constrained k-defective clique problem.*

The constrained k-defective clique problem
Input: a graph $G = (V, E)$, a k-defective clique $P \subseteq V$ and a non-negative integer k.
Objective: obtain the maximum k-defective clique S from G such that $P \subseteq S$.

The maximum k-defective clique S is such that $P \subseteq S$ is a *solution* to the input instance $I = (G, P, k)$. For the input graph G, when $P = \emptyset$, the constrained k-defective clique problem is equivalent to the MDCP. Therefore, any algorithm for the constrained k-defective clique problem can solve the MDCP problem by setting $P = \emptyset$.

Assume $I = (G = (V, E), P, k)$ is the input instance of a branch. MADEC first uses a set of *reduction rules* to simplify the instance I or terminate the search. In articles such as [23], the reduction rule, which decides whether instance I contains a solution of size larger than the current best-known solution, is also known as the bounding rule.

When the instance I cannot be further reduced while the optimal constrained k-defective clique of I is still unknown, MADEC applies a *branching rule* to branch I into smaller subinstances. For example, a simple (and trivial) branching rule is to branch I into $I_1 = (G - \{v\}, P, k)$ and $I_2 = (G, P \cup \{v\}, k)$ (v is an arbitrary vertex in G).

3.1 Reduction and branching rules

Let $I = (G = (V, E), P, k)$ be an instance. We establish three reduction rules below to reduce the instance above and two branching rules to generate subinstances when no reduction is applicable.

3.1.1 Reduction rules

The following reduction rule can be used to terminate the search.

Reduction 1 *If V is a k-defective clique, then the only solution to I is V. If $V \setminus P = \emptyset$, then P is the optimal solution. Hence, in either of the two cases, the search of I can be terminated.*

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

Reduction 2 *If $|N_V(v)| = |V| - 1$, then v is in all optimal solutions of I and can be moved to P without any missing optimal solution of I.*

Reduction 3 *If $|\overline{N}_P(v)| > k - |\overline{E}(P)|$, then v is not in any optimal solution of I and can be removed from G without any missing optimal solution of I.*

These rules can be validated by the definition of the k-defective clique.

Next, we describe the branching rules. Assume $I = (G = (V, E), P, k)$ is an instance that cannot be further reduced. Next, we 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 *Nondefective Candidates*, $C^- = \{u \in V \setminus P : |\overline{N}_P(u)| = 0\}$ (see Figure 2 for an example).

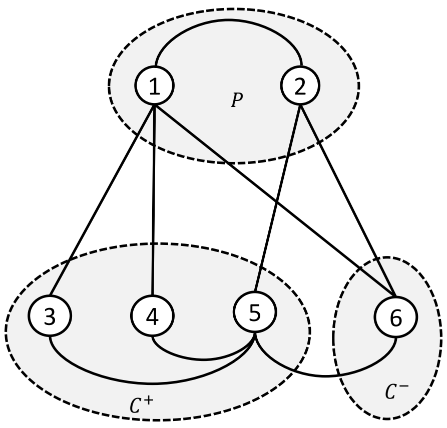

Fig. 2. Example of instance $I = (G, P, k)$, $k = 3$. Note that $|\overline{E}(P \cup C^+)| > k$.

3.1.2 Branching Rule 1 when $|\overline{E}(P \cup C^+)| > k$

We first design a branching rule to address the case where $|\overline{E}(P \cup C^+)| > k$. Suppose C^+ is arbitrarily ordered as $v_1, v_2 \cdots, v_q$, q being the size of C^+. Let p be the largest index such that $P \cup \{v_1, \ldots, v_p\}$ is a k-defective clique but $P \cup \{v_1, \ldots, v_{p+1}\}$ is not. Because $|\overline{E}(P \cup C^+)| > k$, we cannot move all the q vertices from C^+ to P such that P is still a k-defective clique; therefore, $p < q$. Meanwhile, p is at least 1; otherwise, v_1 should be reduced by the Reduction Rule 3. Hence, for the case $|\overline{E}(P \cup C^+)| > k$, we design the following branching rule, which separates I into $p+1$ smaller instances, where $1 \leq p < q$.

Branching 1 *If $|\overline{E}(P \cup C^+)| > k$, then we can generate $p+1$ instances, $Br1(I, 1)$, …, $Br1(I, p+1)$, such that the optimal solution to I is the best solution of all the $p+1$ instances.*

- *In $Br1(I, 1)$, v_1 is deleted from G, i.e., $Br1 = (G - \{v_1\}, P, k)$,*
- *in $Br1(I, i)$ $(i = 2, \ldots, p+1)$, $\{v_1, \ldots, v_{i-1}\}$ is moved to P and v_i is deleted from G, i.e., $Br1(I, i) = (G - \{v_i\}, P \cup \{v_1, \ldots, v_{i-1}\}, k)$.*

For example, in Figure 2, suppose $C^+ = \{3, 4, 5\}$, $C^- = \{6\}$. Because $P \cup \{3, 4\}$ is a 3-defective clique, which is not the case for $P \cup \{3, 4, 5\}$, we have $p = 2$ and $q = 3$. By the branching rule 1, three instances, $Br1(I, 1) = (G - \{3\}, P, 3)$, $Br1(I, 2) = (G - \{4\}, P \cup \{3\}, 3)$, and $Br1(I, 3) = (G - \{5\}, P \cup \{3, 4\}, 3)$ are generated.

3.1.3 Branching Rule 2 when $|\overline{E}(P \cup C^+)| \leq k$

Next, we address the case where $|\overline{E}(P \cup C^+)| \leq k$. Clearly, C^- is not empty in this case; otherwise, $P \cup C^+$ would be a k-defective clique and I can be reduced by the Reduction Rule 1. Hence, our Branching Rule 2 can be used to branch the instance.

Branching 2 *If $|\overline{E}(P \cup C^+)| \le k$, let u be a random vertex from C^-; therefore, we can generate two instances $Br2(I,1)$ and $Br2(I,2)$ such that the optimal solution to I is the better one of the solutions of $Br2(I,1)$ and $Br2(I,2)$.*

- *In $Br2(I,1)$, u is deleted from G, i.e., $Br2(I,1) = (G - \{u\}, P, k)$,*
- *in $Br2(I,2)$, u is moved from C^- to P, i.e., $Br2(I,1) = (G, P \cup \{u\}, k)$.*

Branching Rules 1 and 2 can be applied to produce new smaller subinstances when the current problem cannot be further reduced using the reduction rules of Section 3.1.1.

3.2 The entire algorithm

By integrating the Reduction Rules 1–3 and Branching Rules 1 and 2 above with the general branch-and-bound framework, we obtained the MADEC algorithm, as presented in Algorithm 1. In the algorithm, lb is a lower bound on the maximum size of the k-defective clique in G and updated when a better lower bound is obtained. The branch-and-bound recursive search is named BBSearch. When BBSearch finishes the tree search, lb records the optimal value.

According to the proof provided in Appendix A, we obtained the following theorem that identifies the worst-case run time of the MADEC algorithm.

Theorem 1 *For a graph $G = (V, E)$ where $|V| = n$ and k is a non-negative value, MADEC executes in $O(P(n)\gamma_k^n)$, where $P(n)$ is a polynomial function related to n, and γ_k is the largest real root of $x^{2k+3} - 2x^{2k+2} + 1 = 0$, e.g., $\gamma_k = 1.9276, 1.9276, 1.9960$, 1.9990, and 1.9996 for $k = 1, 2, 3, 4$, and 5, respectively.*

4 Improving MADEC with additional strategies

Next, we investigate additional reduction and bounding rules to improve the practical performance of MADEC, which are particularly relevant for addressing large real-life networks.

4.1 Using a good initial lower bound

To improve the MADEC algorithm, a straightforward strategy is to start the search with an optimal initial lower bound. This can be realized using a MCP heuristic to obtain the maximal clique as the size of a clique in a graph is also

Algorithm 1: Maximum k-defective clique algorithm – MADEC

```
Input: A graph $G = (V, E)$, value $k$
Output: Size of maximum $k$-defective clique in $G$
1  $MADEC(G, k))$
2  begin
3      $lb \leftarrow 0$
4      $BBSearch(G, \emptyset, k)$
5      return $lb$
6  $BBSearch(G = (V, E), P, k)$
7  begin
8      if $P$ is not a $k$-defective clique in $G$ then
9          return
10     else if $V$ is a $k$-defective clique then
11         $P \leftarrow V$
12     if $|P| > lb$ then
13         $lb \leftarrow |P|$
14     if $V \setminus P \neq \emptyset$ then
15         if $\exists v \in V \setminus P$ such that $v$ is reducible by Rule 2 then
16             $BBSearch(G, P \cup \{v\}, k)$
17         else if $\exists v \in V \setminus P$ such that $v$ is reducible by Rule 3 then
18             $BBSearch(G - \{v\}, P, k)$
19         else if $\overline{E}(P \cup \{C^+\}) > k$ then
20             for each $I' \in \{Br1(I, 1), \ldots, Br1(I, p + 1)\}$ generated by
               Branching Rule 1, Call $BBSearch(I')$
21         else
22             for each $I' \in \{Br2(I, 1), Br2(I, 2)\}$ generated by Branching
               Rule 2, Call $BBSearch(I')$
```

a valid lower bound on the maximum k-defective clique. Heuristic algorithms for the MCP are abundant [22]. In our case, we adopted the simple heuristic algorithm in [15] (called FastLB), which is fast and particularly effective for processing massive graphs.

4.2 Reduction by vertex degrees

Based on a nontrivial lower bound, we can prune vertices whose degrees satisfy some specific conditions before starting an exact search.

Reduction 4 *Given a graph $G = (V, E)$ and a lower bound lb, then for any vertex $v \in V$ such that $|N_V(v)| \leq lb - k - 1$, v is not contained in any*

k-defective clique of size larger than lb and can be safely removed from the graph.

This rule can be validated by the definition of the k-defective clique. Before the start of the exact search, we recursively removed vertices from the input graph without deleting any optimal solution via a preprocessing procedure called *Peel*, as presented in Alg. 2. *Peel* recursively removes vertices of degree equal to or less than $lb - k - 1$ and performs in $O(|E|)$ time.

Algorithm 2: The preprocessing procedure – Peel.

```
Input: A graph $G$, a lower bound $lb$
Output: A smaller graph after reduction
Peel($G = (V, E), lb, k$)
begin
    Initialize a queue $Q$
    for $v \in V$ do
        if $|N_V(v)| < lb - k - 1$ then
            $Q$.push_back($v$)
            $G \leftarrow G - \{v\}$
    while $Q \neq \emptyset$ do
        $v \leftarrow Q$.pop()
        for $u \in N_V(v)$ do
            if $|N_V(u)| < lb - k - 1$ and $u \notin Q$ then
                $Q$.push_back($u$)
                $G \leftarrow G - \{u\}$
    return $G$
```

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

Reduction 5 *Given an instance $I = (G = (V, E), P, k)$ and a lower bound lb, then for any vertex $v \in V \setminus P$ such that $|N(v)| \leq lb - k - 1 - |\overline{E}(P)|$, v is not contained in any k-defective clique of size larger than lb and can be safely removed from the graph.*

The two rules above can be regarded as extensions of Reduction Rules 2 and 3. By utilizing them, we can safely remove irrelevant vertices from $V \setminus P$ and hence reduce the search space that must be examined during the subsequent exact search procedure. The idea of *Peel* was first introduced to obtain the maximum quasi-clique in [1]. The basic ideas of the reduction rules above have also been exploited in the RDS algorithm presented in [19].

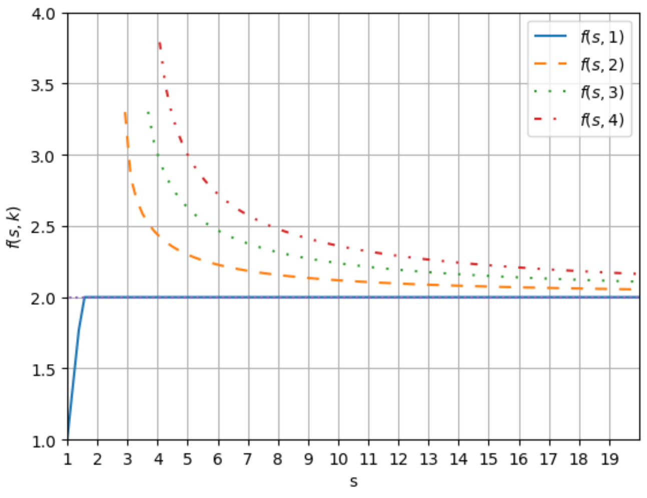

Fig. 3. Graphs of $f(s,1)$, $f(s,2)$, $f(s,3)$, and $f(s,4)$ with respect to different s.

4.3 Reduction by two-hop constraint

Next, we develop a two-hop reduction rule that originates from Property 2.

Let us define $f(s,k) = \frac{(2s+1)-\sqrt{4s^2-12s+17-8k}}{2}$. As shown in Figure 3, $f(s,1)$, $f(s,2)$, $f(s,3)$ and $f(s,4)$ are monotone non-increasing as s increases in range $[\frac{3}{2}+\sqrt{2k-2}, +\infty)$. In fact, let k be a positive integer, the derivative of $f(s,k)$ with respect to s, i.e., $1-\sqrt{\frac{(2s-3)^2}{(2s-3)^2+(8-8k)}}$, is negative when $s > \frac{3}{2}+\sqrt{2k-2}$. Hence, for any integer $k \geq 1$, $f(s,k)$ is a monotone non-increasing function as s increases from $\frac{3}{2}+\sqrt{2k-2}$. Meanwhile, for a complete graph of n vertices, at least $n-1$ edges must be removed to disconnect the vertices. By definition, a k-defective clique S can be obtained by removing a maximum of k edges from a complete graph $G[S]$. Hence, in a k-defective clique S, if $k < |S|-1$, then $G[S]$ must be a connected graph. Because $\frac{3}{2}+\sqrt{2k-2} < k+2$ for any $k > 1$, the following reduction rule holds.

Reduction 6 *Given an instance $I = (G = (V,E), P, k)$ and a lower bound $lb \geq k+2$, then for any vertex $v \in V \setminus P$ such that there exists $u \in P$ satisfying $dist_G(v,u) > \lfloor f(lb,k) \rfloor$, v is not contained in any k-defective clique of size larger than lb and can be safely removed from the graph.*

Reduction Rule 6 is a direct application of Property 2. However, the computational overheads of computing $dist_G(v,u)$ for any $u \in P$ and $v \in V \setminus P$ increases the overall running time of the algorithm significantly. To reduce the running time for calculating the distances, we used a simpler version of Reduction Rule 6. If $k \geq 1$, one can easily verify that $\lfloor f(s,k) \rfloor = 2$ holds for any $s \geq k+2$. Therefore, the following reduction rule holds.

Reduction 7 (Two-hop reduction) *Given an instance $I = (G = (V, E), P, k)$ and a lower bound $lb \geq k + 2$, for any vertex $v \in V \setminus P$, if there exists $u \in P$ such that $v \notin N_V(u)$ and $v \notin \bigcup_{w \in N_V(u)} N_V(w)$, then v is not contained in any k-defective clique of size larger than lb and can be safely removed from the graph.*

The two-hop constraint was also introduced in [19] but was mainly used for scale-reduction prior to the start of the search algorithm.

4.4 Bounding by color-bound

A well-known property of clique models is that the chromatic number of a graph is the upper bound of the maximum clique [22,16,15,10]. Because the maximum k-defective clique is a generalized version of the MCP, we generalize this property in the context of the MDCP.

Lemma 1 (color-bound) *If $G = (V, E)$ can be c-colored, i.e., V is partitioned into c disjoint independent sets (also known as color classes) $\mathcal{P} = \{V_1, \ldots, V_c\}$, then $\sum_{i=1}^{c} \min\left(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |V_i|\right)$ is the upper bound of the size of the maximum k-defective clique in G, known as the color-bound for a k-defective clique.*

Proof. Suppose that S^* is a maximum k-defective clique of G. According to the partition $\mathcal{P}$, S^* is partitioned into c independent sets $S_1, \ldots, S_c$, where $S_i \subseteq V_i$. Clearly, S_i ($i = 1, \ldots, c$) is an independent set as well as a k-defective clique by Property 1. Following the definition of the k-defective clique, we have $\binom{|S_i|}{2} - k \leq 0$, indicating that $|S_i|$ is $\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor$ at the most. As $|V_i|$ can be less than $\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor$, we have $|S_i| \leq \min\left(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |V_i|\right)$. Therefore, $|S^*| = \sum_{i=1}^{c} |S_i| \leq \sum_{i=1}^{c} \min\left(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |V_i|\right)$. Λ

When $k = 0$, Lemma 1 indicates that c is an upper bound of the maximum clique size. To estimate a valid upper bound of the maximum k-defective clique of G, we adopted the fast coloring procedure presented in [16]. This procedure, denoted by $ColorBound(G, k)$, is as follows:

(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 that is not adjacent to any vertices in V_c and then move the vertex to V_c; repeat the operations above until no such vertex exists.
(3) Remove vertices of V_c from V. If V is not empty, increase c by 1 and return to Step 2. Otherwise, return $\sum_{i=1}^{c} \min\left(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |V_i|\right)$ as a color-bound of G.

The time complexity of $ColorBound(G, k)$ is bounded by $O(|V|^3)$. However, it is rapid in practice, in particular owing to the use of the bit-parallel technique [16], which accelerates Step 2. By Lemma 1, the following reduction rule holds.

Reduction 8 *Given an instance $I = (G = (V, E), P, k)$ and a lower bound lb, if $|P| + ColorBound(G[V \setminus P], k) \leq lb$, then a solution to I of size larger than lb does not exist, and we can safely terminate the search of I.*

To our knowledge, this is the first time that graph coloring is used for bounding the maximum k-defective clique.

4.5 The enhanced algorithm–MADEC$^+$

By integrating the reductions above and the pruning rules in MADEC, we obtained an improved algorithm, known as MADEC$^+$and presented in Alg. 3. MADEC$^+$ starts from the MCP heuristic FastLB to initialize the lower bound lb. In the subsequent branch-and-bound search, lb is updated each time a better solution (i.e., a larger k-defective clique) is obtained. The new BBSearch uses the same branching rules as MADEC but incorporates the new reduction techniques introduced in this section.

Because Reduction Rules 5, 7 and 8 are based on a lower bound obtained heuristically, in the worst case, these rules may fail to reduce any search space. Therefore, MADEC$^+$ can degenerate into a simple MADEC algorithm. Hence, the worst-case run time of MADEC$^+$ is the same as that of MADEC (see Section 3.2).

4.6 Discussions

We now discuss the differences between our approach and the RDS approach ([20,19,6]) which is also based on the branch-and-bound framework.

The branching rules between MADEC$^+$ and RDS are intrinsically different. As discussed in Section 1.1, RDS successively solves a series of n nested subproblems with an underlying branch-and-bound algorithm, while using the optimal objective values of the solved subproblems to bound the current search. In fact, the subproblem solved by RDS at the ith round can be formulated as a constrained k-defective clique problem with $V = \{v_i, \ldots, v_n\}$ and $P = \{v_i\}$. However, unlike MADEC, RDS generates $n - i + 1$ subproblems, where in the jth subproblem ($j = 1, \ldots, n - i + 1$), v_{i+j} is moved to P and $v_{i+1}, \ldots, v_{i+j-1}$ are removed from G.

Algorithm 3: The maximum k-defective clique algorithm: MADEC$^+$

```
Input: A graph $G = (V, E)$, value $k$
Output: The size of maximum $k$-defective clique in $G$
$MADEC^+(G, k)$
begin
    $lb \leftarrow FastLB(G, k)$
    $G \leftarrow Peel(G, lb, P, k)$
    $BBSearch(I = (G, \emptyset, k))$
    return $lb$

$BBSearch(I = (G = (V, E), P, k))$
begin
    if $P$ is not a $k$-defective clique or $|V| \leq lb$ or
      $ColorBound(G[V \setminus P], k) + |P| \leq lb$ then
        return
    else if $V$ is a $k$-defective clique then
        $P \leftarrow V$
    if $|P| \geq lb$ then
        $lb \leftarrow |P|$
    if $V \setminus P \neq \emptyset$ then
        if $\exists v \in V \setminus P$ such that $v$ is reducible by Rule 2 then
            $BBSearch(G, P \cup \{v\}, k)$
        else if $\exists v \in V \setminus P$ such that $v$ is reducible by Rules 3, 4 or 7 then
            $BBSearch(G - \{v\}, P, k)$
        else if $\overline{E}(P \cup \{C^+\}) > k$ then
            for each $I' \in \{Br1(I, 1), \ldots, Br1(I, p + 1)\}$ generated by
              Branching Rule 1, Call $BBSearch(I')$
        else
            for each $I' \in \{Br2(I, 1), Br2(I, 2)\}$ generated by Branching
              Rule 2, Call $BBSearch(I')$
```

Furthermore, another important difference between MADEC$^+$ and RDS is the bounding and pruning rules. Two-hop reduction is used during the search in MADEC$^+$; however, this property is applied only for graph preprocessing in RDS ([19]). Moreover, the color-bound is proposed for the first time in this study.

5 Computational experiments

We performed computational experiments to evaluate the two proposed algorithms (MADEC and MADEC$^+$), which were written in C++ and compiled in g++. Hence, we adopted two best-known approaches–the RDS algorithm in [6] and the ILP solver CPLEX (based on the ILP formulation 1 in Section 1) as our reference algorithms. Our source codes (MADEC, MADEC$^+$ and ILP) are publicly available at `https://github.com/chenxiaoyu233/k-defective`. For RDS, we used one efficient implementation in C++ available from `https://github.com/zhelih/rds-serial`. For the ILP approach, we used version 12.9 of the IBM CPLEX solver. For all the algorithms, we turned off the parallelization mode and used only one thread to solve each input instance. Because all these algorithms are exponential time algorithms, we set a uniform cut-off time of 18000 s (5 h). All the experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8 GHz and 2 GB RAM) running CentOS 6.5.

These algorithms were tested on three groups of benchmark graphs, i.e., random graphs, DIMACS2 graphs, and massive real-life graphs.

5.1 Computational results on random graphs

We generated 18 random graphs based on the procedure described in [7]. The graphs can be categorized into two groups. In the first group, the number of vertices of a graph n is 100, whereas in the second group, $n = 200$. Each group comprised nine graphs, and each graph had an edge density ρ in $\{0.1, 0.2, \cdots, 0.9\}$, where ρ is defined as $\frac{2|E|}{|V|(|V|-1)}$. We investigated the performance of the aforementioned algorithms on these 18 graphs with k varying in $\{1, 2, \cdots, 6\}$.

The computational results are summarized in Figure 4. Each subfigure shows the run time of the four algorithms for a specific pair of vertex number n and k, which are indicated on the top of the subfigures. In each subfigure, the horizontal and vertical axes represent the density of the graph and the run time in seconds, respectively.

For the first group of graphs where $n = 100$, only MADEC$^+$ and CPLEX can solve all the graphs for each k value. Furthermore, we observed that MADEC$^+$ always performed faster than or at least as fast as the other algorithms for sparse and mildly dense graphs ($\rho \leq 0.6$). When the graphs became extremely dense and k increased, i.e., $\rho = 0.9$ and $k = 6$, MADEC$^+$ was on par with CPLEX in terms of performance. In fact, the number of constraints in the ILP formulation (1) decreased significantly when the density approached 1, rendering the problem easy to solve for CPLEX. For the second group of graphs where $n = 200$, no algorithm can solve all the instances within 5 h. When $k = 1, 2, 3$, MADEC$^+$ can solve more graphs than the competitors in 5 h. When $k = 4$, RDS became the most competitive approach. All the algorithms can barely obtain an optimal solution for dense graphs within 5 h.

5.2 Computational results on DIMACS2 graphs

The 2nd DIMACS Implementation Challenge Benchmark (DIMACS2) [1] is the de facto standard benchmark set for testing clique or clique-related algorithms. The DIMACS2 benchmark set includes 80 graphs, whose sizes range from small graphs of 50 vertices and 1,000 edges to large graphs of 4,000 vertices and 5,000,000 edges. These graphs are either from real-world problems (e.g., coding theory, fault diagnosis, and the Steiner triple problem) or random graphs. We tested all these graphs with $k = 1, \ldots, 4$. For larger k values, most graphs cannot be solved within the time limit of 5 h. Furthermore, in studies regarding k-defective cliques such as [6,18,20], the experiments also involved k values from 1 to 4.

Tables 1 and 2 show the computation times of the four competing algorithms (MADEC, MADEC$^+$, RDS, CPLEX) for the 80 DIMACS2 graphs. The first column indicates the basic information of each graph (name, number of vertices and edges). Column "*LB*" provides the lower bound from the preprocessing procedure in Algorithm 2, whereas column "*opt*" provides the optimal k-defective clique size obtained by the algorithm. "–" indicates that the corresponding algorithm fails to reach the optimal solution in 5 h. We omitted rows where neither of the four algorithms obtained the optimal solution, and we denoted the best run time in **bold**.

In Tables 1 and 2 on the DIMACS2 graphs, we observed that MADEC$^+$ dominated MADEC, indicating that the additional reduction strategies were important for solving these instances. For a better understanding of the performances of MADEC$^+$, RDS, and CPLEX, we summarize the numbers of instances solved by these algorithms within 5 h in Table 3. Clearly, MADEC$^+$ outperformed RDS and performed better than CPLEX when $k \leq 4$ in terms of the number of solved instances. In fact, the three algorithms competed with each other in different families of graphs. For example, for the *c-fat* family graph, MADEC$^+$ and RDS performed 3–4 orders of magnitude faster than CPLEX. However, for most *san* family graphs, only MADEC$^+$ and CPLEX obtained the optimal solutions. Additionally, our preprocessing procedure did not reduce any vertex for these DIMACS2 graphs owing to their high densities.

[1] http://www.cs.hbg.psu.edu/txn131/clique.html.

Table 1
Computation time of the four algorithms for all DIAMCS2 graphs (part 1).

graph name $(\lvert V\rvert,\lvert E\rvert)$	k	LB	opt	Running time in seconds			
				MADEC^{+}	MADEC	RDS	CPLEX
C125.9.clq (125,6963)	1	31	35	**37.41**	7801.85	–	1490.21
	2	31	36	**378.32**	–	–	1546.12
	3	31	37	2072.35	–	–	**1886.73**
	4	31	38	8308.56	–	–	**4662.59**
DSJC500_5.clq (500,62624)	1	12	14	2675.77	9585.33	**1029.00**	–
MANN_a27.clq (378,70551)	1	125	127	–	–	–	**1162.95**
	2	125	128	–	–	–	**110.65**
	3	125	129	–	–	–	**125.94**
	4	125	130	–	–	–	**133.57**
MANN_a45.clq (1035,533115)	1	342	346	–	–	–	**640.66**
	2	342	347	–	–	–	**2246.70**
	3	342	348	–	–	–	**2288.89**
	4	342	349	–	–	–	**1073.24**
MANN_a9.clq (45,918)	1	16	17	0.03	8.28	0.08	557.41
	2	16	18	0.12	66.81	1.01	641.37
	3	16	19	**0.56**	266.42	6.15	587.96
	4	16	20	**1.79**	696.46	20.48	621.51
brock200_1.clq (200,14834)	1	18	21	**539.53**	5519.06	7816.60	–
	2	18	22	**6910.64**	–	–	–
brock200_2.clq (200,9876)	1	10	12	8.86	22.79	**1.33**	4376.88
	2	10	12	146.03	370.54	**23.91**	4962.37
	3	10	13	1039.20	3101.30	**108.74**	4936.51
	4	10	13	7533.36	–	**830.65**	–
brock200_3.clq (200,12048)	1	12	15	33.85	137.35	**26.04**	6586.26
	2	12	16	**385.36**	2100.47	429.40	10826.17
	3	12	16	4055.42	–	**2728.20**	15911.75
	4	12	17	–	–	**13749.00**	–
brock200_4.clq (200,13089)	1	14	17	**55.22**	427.69	107.35	13484.38
	2	14	18	**803.82**	7182.26	1452.80	–
	3	14	18	**6603.53**	–	15034.00	–
c−fat200−1.clq (200,1534)	1	12	12	0.05	0.54	0.00	1998.17
	2	12	12	0.07	2.95	0.01	2464.29
	3	12	12	0.10	22.62	0.06	2390.07
	4	12	12	0.14	73.71	0.09	2709.05
c−fat200−2.clq (200,3235)	1	24	24	0.05	0.63	0.00	1395.73
	2	24	24	0.07	4.39	0.00	1311.65
	3	24	24	0.16	23.10	0.01	1610.97
	4	24	24	0.39	92.31	0.03	1639.97
c−fat200−5.clq (200,8473)	1	58	58	0.12	0.73	0.00	423.42
	2	58	58	0.54	9.64	0.00	516.23
	3	58	58	1.59	45.83	**0.01**	721.52
	4	58	58	8.86	317.77	**0.01**	742.89
c−fat500−1.clq (500,4459)	1	14	14	0.77	7.92	0.04	–
	2	14	14	0.86	59.65	0.08	–
	3	14	14	0.86	666.02	1.21	–
	4	14	14	1.09	5082.27	1.72	–
c−fat500−10.clq (500,46627)	1	126	126	3.23	21.31	**0.01**	–
	2	126	126	30.17	330.07	**0.02**	–
	3	126	126	60.29	3109.01	**0.04**	17852.12
	4	126	126	316.12	–	**0.09**	–
c−fat500−2.clq (500,9139)	1	26	26	0.86	10.68	0.02	–
	2	26	26	1.04	78.16	0.05	–
	3	26	26	1.43	704.55	**0.26**	–
	4	26	26	2.46	5450.48	**0.40**	–
c−fat500−5.clq (500,23191)	1	64	64	1.02	15.74	**0.01**	–
	2	64	64	2.23	146.42	**0.03**	–
	3	64	64	6.65	1087.42	**0.06**	–
	4	64	64	18.56	8031.17	**0.14**	–
gen200_p0.9_44.clq (200,17910)	1	35	45	**696.27**	–	–	1818.41
	2	35	46	9048.11	–	–	**152.48**
	3	35	46	–	–	–	**278.21**
	4	35	47	–	–	–	**4790.88**
gen200_p0.9_55.clq (200,17910)	1	39	56	**157.80**	–	–	219.20
	2	39	57	**2526.87**	–	–	3015.03
	3	39	57	–	–	–	**166.55**
	4	39	58	–	–	–	**173.29**
hamming10−2.clq (1024,518656)	1	512	512	5.45	–	**1.61**	2.89
	2	512	512	28.03	–	3.83	2.87
	3	512	512	108.33	–	16.60	**2.73**
	4	512	512	506.15	–	63.51	**3.58**
hamming6−2.clq (64,1824)	1	32	32	0.01	0.64	0.00	1.80
	2	32	32	0.03	7.22	0.00	1.62
	3	32	32	0.20	49.71	0.00	0.88
	4	32	32	0.87	268.50	0.00	0.95
hamming6−4.clq (64,704)	1	4	4	0.09	0.07	0.00	2929.78
	2	4	5	0.45	0.50	0.01	2375.61
	3	4	6	1.14	1.74	**0.01**	119.81
	4	4	6	4.46	5.27	**0.04**	882.85
hamming8−2.clq (256,31616)	1	128	128	0.17	–	0.02	1.36
	2	128	128	0.74	–	0.05	1.05
	3	128	128	2.03	–	0.18	0.86
	4	128	128	10.97	–	0.53	0.96
hamming8−4.clq (256,20864)	1	16	16	40.60	1080.91	**0.12**	3475.41
	2	16	16	1236.09	–	**2.01**	2662.11
	3	16	16	16520.67	–	**63.07**	3192.75
	4	16	17	–	–	**528.46**	2918.38

Table 2
Computation time of the four algorithms for all DIAMCS2 graphs (part 2).

graph name (|V|, |E|)	k	LB	opt	Running time in seconds			
				MADEC$^+$	MADEC	RDS	CPLEX
johnson16−2−4.clq (120,5460)	1	8	8	1159.73	969.41	**20.13**	107.89
	2	8	9	15176.22	12705.94	**243.20**	1715.38
	3	8	9	–	–	4068.50	**288.32**
	4	8	10	–	–	1800.00	**344.08**
johnson32−2−4.clq (496,107880)	1	16	16	–	–	–	**1813.32**
johnson8−2−4.clq (28,210)	1	4	4	0.02	0.01	0.00	38.45
	2	4	5	0.06	0.05	0.00	48.23
	3	4	5	0.20	0.16	0.00	74.83
	4	4	6	0.35	0.28	0.00	40.89
johnson8−4−4.clq (70,1855)	1	14	14	0.17	1.93	0.00	53.07
	2	14	14	2.03	26.71	**0.01**	96.95
	3	14	14	21.20	210.12	**0.12**	1177.80
	4	14	15	54.44	804.41	**0.61**	152.86
keller4.clq (171,9435)	1	11	12	14.26	139.52	**2.73**	6092.35
	2	11	13	198.53	2725.41	**46.65**	580.62
	3	11	14	1191.99	–	**438.38**	3065.87
	4	11	15	7080.61	–	778.83	**636.33**
p_hat1000−1.clq (1000,122253)	1	10	11	1693.89	3193.77	**127.20**	–
	2	10	12	–	–	**2007.70**	–
p_hat1500−1.clq (1500,284923)	1	11	12	–	–	**2034.50**	–
p_hat300−1.clq (300,10933)	1	8	9	7.97	8.69	**0.25**	10052.29
	2	8	9	98.53	140.32	**3.78**	10691.14
	3	8	10	759.02	1204.25	**20.62**	–
	4	8	10	3633.09	6786.02	**135.64**	–
p_hat300−2.clq (300,21928)	1	24	26	**17.28**	285.69	268.74	–
	2	24	27	**323.60**	6194.20	4587.90	–
	3	24	28	**3764.77**	–	–	–
p_hat300−3.clq (300,33390)	1	32	37	**1798.92**	–	–	–
p_hat500−1.clq (500,31569)	1	9	10	63.88	83.54	**3.35**	–
	2	9	11	1091.01	1889.86	**45.08**	–
	3	9	11	11031.05	–	**494.89**	–
	4	9	12	–	–	**2737.30**	–
p_hat500−2.clq (500,62946)	1	34	37	**456.76**	–	–	–
	2	34	38	**10171.30**	–	–	–
p_hat700−1.clq (700,60999)	1	9	12	251.14	490.74	**9.46**	–
	2	9	12	5175.18	11067.12	**188.59**	–
	3	9	13	–	–	**1320.50**	–
	4	9	13	–	–	**12478.00**	–
p_hat700−2.clq (700,121728)	1	41	45	**4723.56**	–	–	–
san1000.clq (1000,250500)	1	9	15	**1900.63**	–	–	–
san200_0.7_1.clq (200,13930)	1	16	30	**4.02**	–	–	3619.30
	2	16	30	**70.71**	–	–	3545.48
	3	16	30	**728.64**	–	–	3443.48
	4	16	30	**5226.59**	–	–	7059.52
san200_0.7_2.clq (200,13930)	1	13	19	**4.88**	–	–	16314.63
	2	13	19	**98.28**	–	–	–
	3	13	20	**1001.86**	–	–	–
	4	13	20	**8620.52**	–	–	–
san200_0.9_1.clq (200,17910)	1	46	70	**11.65**	–	–	1840.70
	2	46	70	155.32	–	–	**72.42**
	3	46	71	1498.42	–	–	**1295.63**
	4	46	71	10264.93	–	–	**2453.86**
san200_0.9_2.clq (200,17910)	1	40	60	**95.12**	–	–	2299.30
	2	40	61	**1344.95**	–	–	2993.94
	3	40	61	10050.17	–	–	**2295.25**
	4	40	61	–	–	–	**1692.01**
san200_0.9_3.clq (200,17910)	1	34	44	1935.90	–	–	**191.16**
	2	34	45	–	–	–	**379.45**
	3	34	45	–	–	–	**4826.02**
	4	34	46	–	–	–	**2891.77**
san400_0.5_1.clq (400,39900)	1	8	13	**28.83**	–	–	–
	2	8	13	**1114.88**	–	–	–
	3	8	13	**13965.69**	–	–	–
san400_0.7_1.clq (400,55860)	1	22	40	**395.05**	–	–	–
	2	22	40	**11189.74**	–	–	–
san400_0.7_2.clq (400,55860)	1	18	30	**553.66**	–	–	–
san400_0.7_3.clq (400,55860)	1	15	22	**2889.04**	–	–	–
san400_0.9_1.clq (400,71820)	1	54	100	–	–	–	**3181.29**
	2	54	100	–	–	–	**2625.88**
	3	54	100	–	–	–	**5321.40**
	4	54	100	–	–	–	**3644.30**
sanr200_0.7.clq (200,13868)	1	16	19	**151.67**	1086.99	444.60	16260.89
	2	16	19	**2328.21**	–	11195.00	–
sanr200_0.9.clq (200,17863)	1	37	43	13402.63	–	–	**5045.97**
	2	37	44	–	–	–	**6682.93**
	3	37	45	–	–	–	**12501.08**
sanr400_0.5.clq (400,39984)	1	12	14	576.75	1793.34	**144.60**	–
	2	12	14	9700.05	–	**3394.20**	–

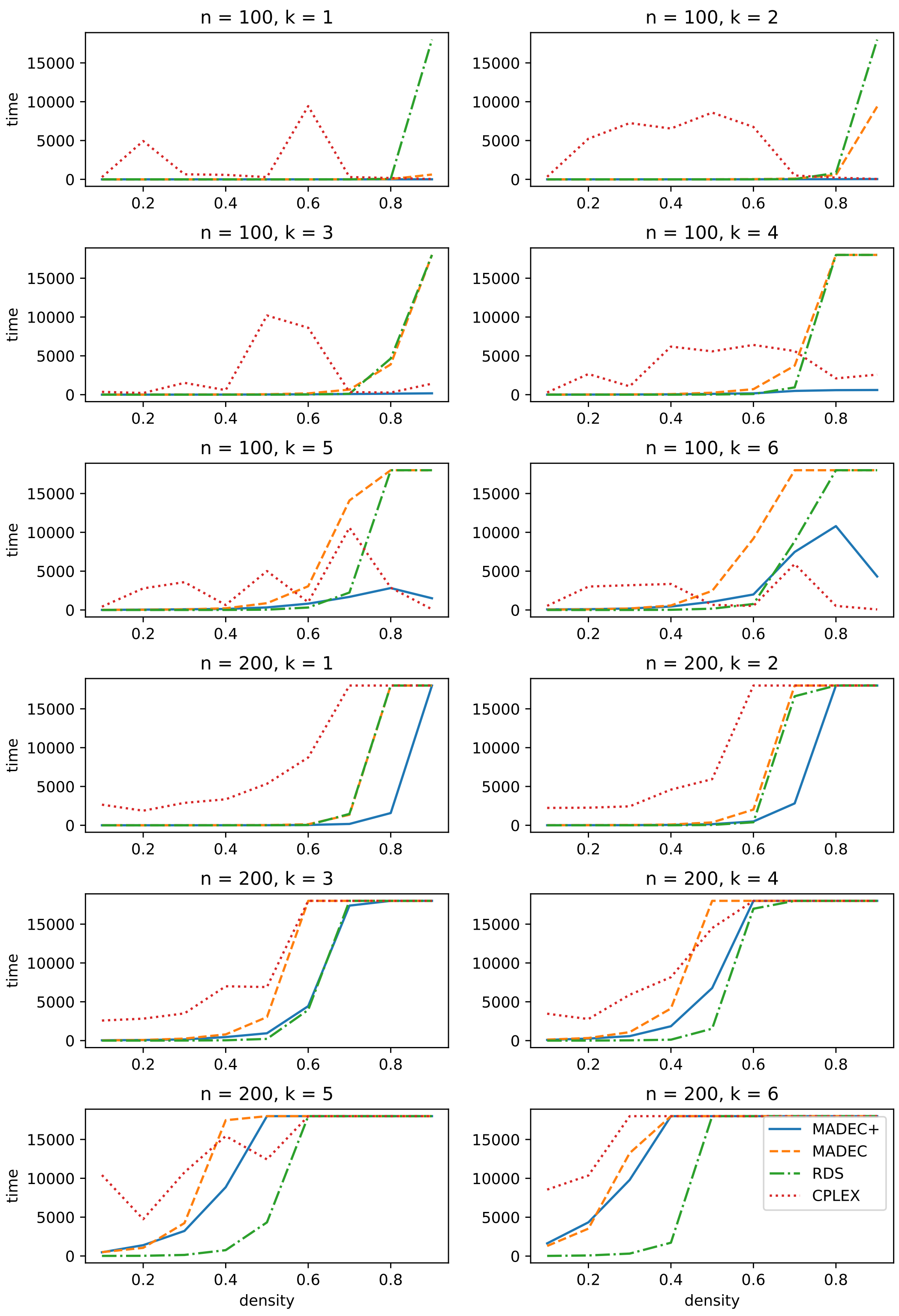

Fig. 4. Computational results of random graphs

5.3 Computational results on massive real-life graphs

We used two sets of well-known real-life benchmarks: the Stanford large network dataset collection (SNAP) and the 10th DIMACS implementation chal-

Table 3
Number of DIMACS2 graphs solved by MADEC$^+$, RDS, and CPLEX in 5 h.

Alg.	$k = 1$	$k = 2$	$k = 3$	$k = 4$
MADEC$^+$	46	37	28	21
RDS	30	27	23	21
CPLEX	31	27	27	23

lenge benchmark (DIMACS10). SNAP is a comprehensive collection of large real-world networks [9], including graphs retrieved from social networks, communication networks, citation networks, etc. DIMACS10 contains artificial and real-world graphs from various applications [2]. Among the 300 graphs in the two sets, we adopted the 43 graphs tested in [6]. It is noteworthy that these 43 graphs included all the graphs tested in [20]. Nonetheless, we tested all these graphs with $k = 1, \ldots, 4$. For a fair comparison of the competing algorithms, we first applied the preprocessing procedure of Section 4.2 (Algorithm 2) to reduce each of these 43 massive graphs and then performed each algorithm to solve the reduced graph.

Tables 4 and 5 show the computational results for these massive real-life graphs. The same information as in Tables 1 and 2 was provided. Instances not solvable by any algorithm were omitted. The results indicated that MADEC$^+$ outperformed the other algorithms. For instances such as web-Google with k=4, a speedup of five orders of magnitude was observed. However, for a few instances such as caidaRouterLevel with $k = 1$ and wiki-Vote with $k = 1, 2$, RDS was the only algorithm that obtained an optimal solution. For almost all the instances, MADEC$^+$ dominated MADEC, indicating the effectiveness of the proposed reduction rules. In Section 5.4.2, we compare the different reduction strategies. CPLEX did not exhibit any advantage over the other algorithms under the current setting.

5.4 Additional studies on preprocessing and pruning rules

We investigated the effect of the key strategies used exclusively in MADEC$^+$: the preprocessing procedure and the pruning rules.

5.4.1 Effectiveness of preprocessing procedure

The aforementioned experimental results indicate that for several large instances, the algorithms stopped within 0.0 s. We observed that for these instances, the preprocessing process procedure *Peel* (Section 4.2, Algorithm 2) removed all or nearly all the vertices for these tested instances. As the preprocessing procedure *Peel* is independent of the exact search process, we can study the variation of the sizes of each graph before and after the preprocessing in this section.

Table 4
Computation time of the four algorithms on SNAP and DIMACS10 graphs (part 1).

graph name ($\|V\|,\|E\|$)	k	LB	opt	Running time in seconds			
				MADEC$^+$	MADEC	RDS	CPLEX
adjnoun.graph (112,425)	1	5	6	0.03	0.04	0.07	66.58
	2	5	6	0.08	0.32	0.08	119.36
	3	5	7	**0.75**	2.18	10.38	1737.46
	4	5	7	**1.01**	5.68	9.44	631.74
as−22july06.graph (22963,48436)	1	17	18	0.37	0.27	75.86	1970.51
	2	17	18	**1.09**	4.34	121.05	197.78
	3	17	19	35.33	**31.85**	–	342.86
	4	17	19	**75.80**	178.82	–	527.92
astro−ph.graph (16706,121251)	1	57	57	0.00	0.00	14.88	0.01
	2	57	57	0.00	0.47	149.68	2.83
	3	57	57	**0.01**	1.09	–	2.76
	4	57	57	**0.32**	32.65	–	304.65
caidaRouterLevel.graph (192244,609066)	1	17	18	–	**3313.56**	–	–
celegans_metabolic.graph (453,2025)	1	9	10	0.01	0.01	0.08	1011.00
	2	9	10	0.17	0.34	0.42	37.53
	3	9	11	7.90	**6.76**	291.50	1694.49
	4	9	11	**55.63**	158.58	2691.50	4879.75
celegansneural.graph (297,2148)	1	8	8	0.77	0.87	24.96	5568.69
	2	8	9	**1.37**	7.97	30.08	4947.01
	3	8	10	**38.45**	69.00	10643.00	5965.59
	4	8	10	**39.85**	330.40	10581.00	6428.07
chesapeake.graph (39,170)	1	5	6	0.00	0.01	0.01	645.90
	2	5	6	0.01	0.03	0.01	1145.84
	3	5	7	0.07	0.09	0.23	709.18
	4	5	7	0.09	0.12	0.26	1308.45
cnr−2000.graph (325557,2738969)	1	84	85	0.00	0.00	199.40	0.01
	2	84	85	0.00	0.00	215.65	0.02
	3	84	86	0.00	0.00	–	0.91
	4	84	86	**0.02**	49.28	–	2.57
coAuthorsCiteseer.graph (227320,769418)	1	87	87	0.00	0.00	284.70	0.01
	2	87	87	0.00	0.00	283.39	0.01
	3	87	87	0.00	0.00	–	0.01
	4	87	87	0.00	0.00	–	0.01
coAuthorsDBLP.graph (299067,917405)	1	115	115	0.00	0.00	316.83	0.01
	2	115	115	0.00	0.00	1529.50	0.01
	3	115	115	0.00	0.00	–	0.01
	4	115	115	0.00	0.00	–	0.01
cond−mat−2003.graph (31163,120029)	1	25	25	0.00	0.00	0.03	0.01
	2	25	25	0.00	0.00	0.05	0.18
	3	25	26	0.00	0.04	45.70	1.30
	4	25	26	**0.00**	1.10	264.44	748.12
cond−mat−2005.graph (40421,175691)	1	30	30	0.00	0.00	0.12	0.01
	2	30	30	0.00	0.00	0.12	0.01
	3	30	30	0.00	0.00	11.22	0.00
	4	30	30	0.00	0.33	193.88	1.22
cond−mat.graph (16726,47594)	1	18	18	0.00	0.00	0.01	0.01
	2	18	18	0.00	0.00	0.01	0.01
	3	18	18	0.00	0.09	9.74	466.41
	4	18	18	**0.01**	2.55	104.20	356.19
dolphins.graph (62,159)	1	5	6	0.00	0.01	0.00	81.16
	2	5	6	0.00	0.03	0.00	1842.66
	3	5	6	0.01	0.21	0.14	1596.54
	4	5	7	0.03	0.38	0.20	113.67
email−EuAll.txt (265214,364481)	1	16	17	–	**442.34**	–	–
	2	16	17	–	**12316.13**	–	–
email.graph (1133,5451)	1	12	12	0.00	0.00	0.00	0.01
	2	12	12	0.03	0.55	1.13	1279.51
	3	12	12	**0.55**	42.61	4192.50	3934.93
	4	12	13	**6.43**	931.71	–	16928.75
football.graph (115,613)	1	9	9	0.02	0.11	0.32	1380.87
	2	9	9	0.04	0.46	0.34	1283.88
	3	9	9	**0.29**	3.00	42.88	1430.37
	4	9	9	**0.48**	5.83	48.06	1384.03
hep−th.graph (8361,15751)	1	24	24	0.00	0.00	0.03	0.01
	2	24	24	0.00	0.00	0.03	0.02
	3	24	24	0.00	0.00	1.52	0.01
	4	24	24	0.00	0.00	1.74	0.01
jazz.graph (198,2742)	1	30	30	0.00	0.00	0.10	0.01
	2	30	30	0.00	0.00	0.10	0.01
	3	30	30	0.00	0.00	9.80	0.01
	4	30	30	0.00	0.00	11.40	0.01
karate.graph (34,78)	1	5	6	0.00	0.00	0.00	0.28
	2	5	6	0.00	0.00	0.00	421.82
	3	5	6	0.01	0.04	0.01	684.04
	4	5	6	0.01	0.04	0.01	1153.63
lesmis.graph (77,254)	1	10	10	0.00	0.00	0.00	0.23
	2	10	11	0.00	0.00	0.00	1.53
	3	10	11	0.00	0.02	0.18	402.35
	4	10	12	0.02	0.07	0.35	569.02

Table 5
Computation time of the four algorithms on SNAP and DIMACS10 graphs (part 2).

graph name ($\|V\|,\|E\|$)	k	LB	opt	Running time in seconds			
				MADEC$^+$	MADEC	RDS	CPLEX
memplus.graph (17758,46894)	1	97	97	0.00	0.00	220.77	0.01
	2	97	97	0.00	0.00	541.71	0.01
	3	97	97	0.00	0.00	–	0.01
	4	97	97	0.00	0.00	–	0.01
netscience.graph (1589,2742)	1	20	20	0.00	0.00	0.01	0.01
	2	20	20	0.00	0.00	0.01	0.01
	3	20	20	0.00	0.00	0.41	0.01
	4	20	20	0.00	0.00	0.41	0.01
p2p−Gnutella04.txt (10876,39994)	1	4	4	–	**15230.06**	–	–
PGPgiantcompo.graph (10680,24316)	1	25	26	0.01	0.14	42.66	988.39
	2	25	27	**0.07**	1.29	42.13	811.92
	3	25	28	**0.79**	6.85	–	1003.85
	4	25	28	**4.28**	52.65	–	1252.95
polblogs.graph (1490,16715)	1	20	21	**6.34**	7.99	3382.30	–
	2	20	22	**15.75**	127.94	3491.00	–
	3	20	22	**531.68**	1147.03	–	–
	4	20	23	**3006.17**	9290.49	–	–
polbooks.graph (105,441)	1	6	7	0.01	0.03	0.03	1856.00
	2	6	7	0.05	0.31	0.10	1642.40
	3	6	8	**0.24**	2.21	10.61	1398.93
	4	6	8	**0.41**	4.03	11.43	1049.76
power.graph (4941,6594)	1	6	6	0.00	0.00	0.00	0.47
	2	6	6	0.00	0.01	0.00	86.63
	3	6	7	**0.10**	37.28	15.71	4768.15
rgg_n_2_17_s0.graph (131072,728472)	1	15	15	0.00	0.00	0.02	104.33
	2	15	16	0.01	0.49	2.11	465.80
	3	15	16	**1.20**	1689.45	–	–
rgg_n_2_19_s0.graph (524288,3269202)	1	18	19	0.00	0.00	0.01	0.01
	2	18	19	0.00	0.05	0.17	301.08
	3	18	19	**0.03**	24.27	4149.00	1506.07
	4	18	20	**0.68**	6968.47	–	–
rgg_n_2_20_s0.graph (1048576,6890893)	1	17	18	0.00	0.35	8.41	1495.40
	2	17	18	**0.50**	41.12	129.92	–
Slashdot0811.txt (77360,469180)	1	26	27	–	**15694.18**	–	–
Slashdot0902.txt (82168,504230)	1	27	28	–	**15943.48**	–	–
soc−Epinions1.txt (75879,405740)	1	23	24	–	**16029.11**	–	–
web−BerkStan.txt (685230,6649470)	1	201	202	**0.09**	3.51	–	17.87
	2	201	202	**1.27**	49.85	–	129.43
	3	201	202	**6.28**	288.53	–	132.42
	4	201	202	**32.44**	2399.58	–	147.32
web−Google.txt (875713,4322051)	1	44	45	0.00	0.04	48.22	12.25
	2	44	46	**0.04**	5.34	1009.80	1269.64
	3	44	46	**0.05**	36.10	–	1479.04
	4	44	47	**0.05**	155.23	–	1741.27
web−NotreDame.txt (325729,1090108)	1	155	155	586.52	**518.28**	–	–
	2	155	155	**1359.30**	3135.32	–	–
web−Stanford.txt (281903,1992636)	1	61	62	**48.14**	254.98	–	–
	2	61	63	**211.48**	5545.57	–	–
	3	61	64	**8283.56**	–	–	–
wiki−Vote.txt (7115,100762)	1	17	18	–	**1351.37**	–	–

For simplicity, we show the numbers of vertices (in log scale) of these graphs before and after the preprocessing for $k = 1$ and 4 in Figures 5 and 6, respectively. For each k, we arbitrarily selected 15 graphs from the group of real-life graphs. As shown, for many large graphs, the number of vertices reduced significantly after the preprocessing. The reduction effect was less significant only for several smaller graphs, such as football.graph and dolphins.graph.

5.4.2 Effectiveness of novel pruning strategies

In Section 4, we introduced novel strategies (degree-based pruning, two-hop constraint, and color-bound) to reduce the search space in large graphs. In Section 5.4.1, we verified that with a high-quality lower bound, degree-based pruning can substantially reduce the graph during preprocessing. To assess the effectiveness of the two-hop constraint and color-bound strategies, we com-

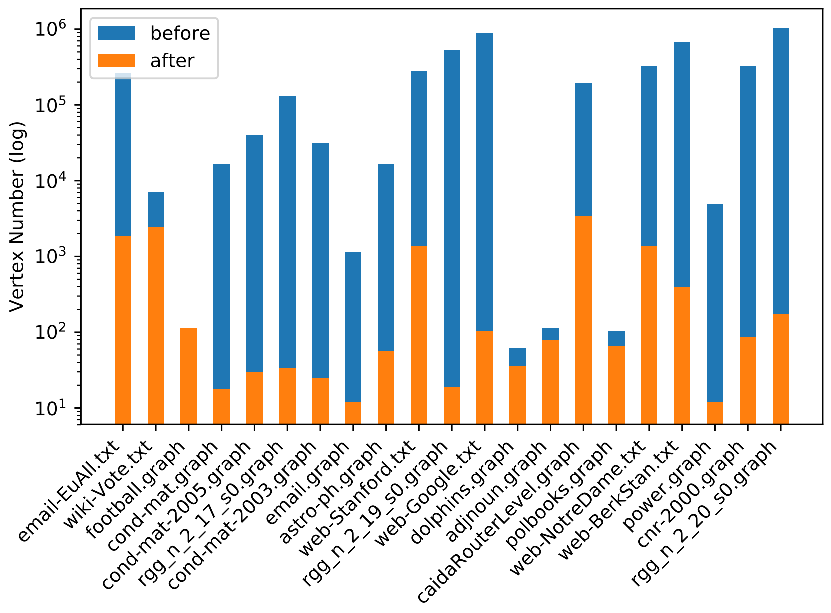

Fig. 5. Number of vertices (in log scale) before and after preprocessing when $k = 1$.

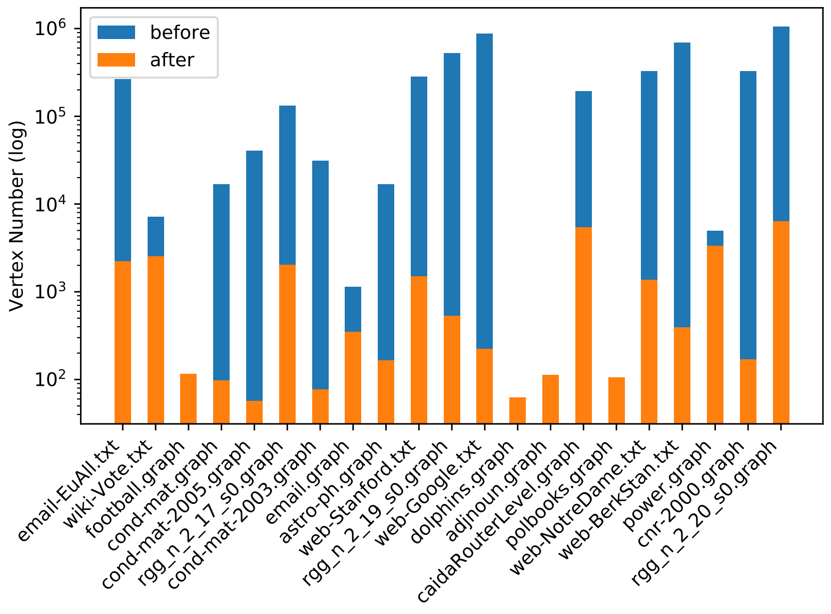

Fig. 6. Number of vertices (in log scale) before and after preprocessing when $k = 4$.

pared MADEC and MADEC$^+$ with two MADEC variants (which can also be considered as two weakened versions of MADEC$^+$).

- **MADEC+diam**, MADEC with degree-based reduction rule (Reduction Rule 5) and two-hop constrained reduction rule (Reduction Rule 7).
- **MADEC+color**, MADEC with degree-based reduction and color-bound (Reduction Rule 8).

As the pruning rules are mainly used for large and massive graphs, we selected four such graphs with various sizes from SNAP and DIMACS10 for this study (see Table 6).

Table 6
General information of selected real-life graphs.

Graph	$(\lvert V\rvert, \lvert E\rvert)$	Source	Description
web-Google.txt	(875713, 5105039)	SNAP	The data was released by Google as a part of the Google Programming Contest.
rgg_n_2_19_s0.graph	(524288, 3269202)	DIMACS10	The data are a random geometric graph with 2^{19} vertices [8].
astro-ph.graph	(16706, 121251)	DIMACS10	The data is a collaboration network of preprints in astrophysics archive
email.graph	(1133, 5451)	DIMACS10	The data are a network of e-mail interchanges between members of the University Rovira i Virgili

In Figures 7 and 8, we demonstrate the time consumption and the number of nodes in the search tree for each algorithm. For each of these graphs and each k, MADEC$^+$ reached the optimal solution with the minimum computational time and the minimum number of tree nodes. As shown in Figure 8, both the two-hop constraint and color-bound pruning techniques effectively reduced the number of search nodes. However, in some cases, these techniques might increase the search time of the branch-and-bound process. For example, MADEC+color generated fewer tree nodes for email.graph for $k = 4$, but the run time was longer than that of MADEC, as shown in Figure 7. This experiment indicates that the use of a single pruning rule may worsen the performance of the pure branch-and-bound algorithm, whereas using both two-hop constraint and color-bound pruning rules jointly can improve the search performance.

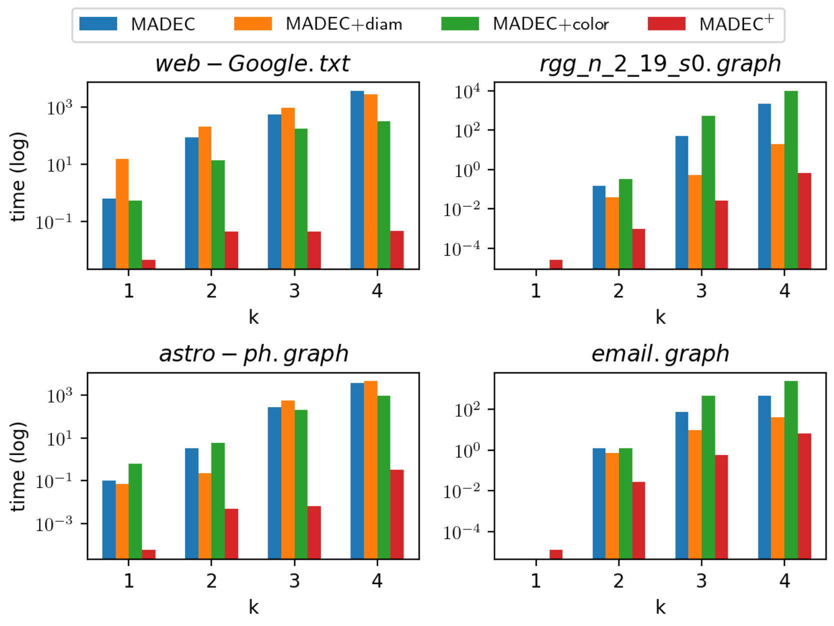

Fig. 7. Computation times in seconds (in log scale) for four real-life instances.

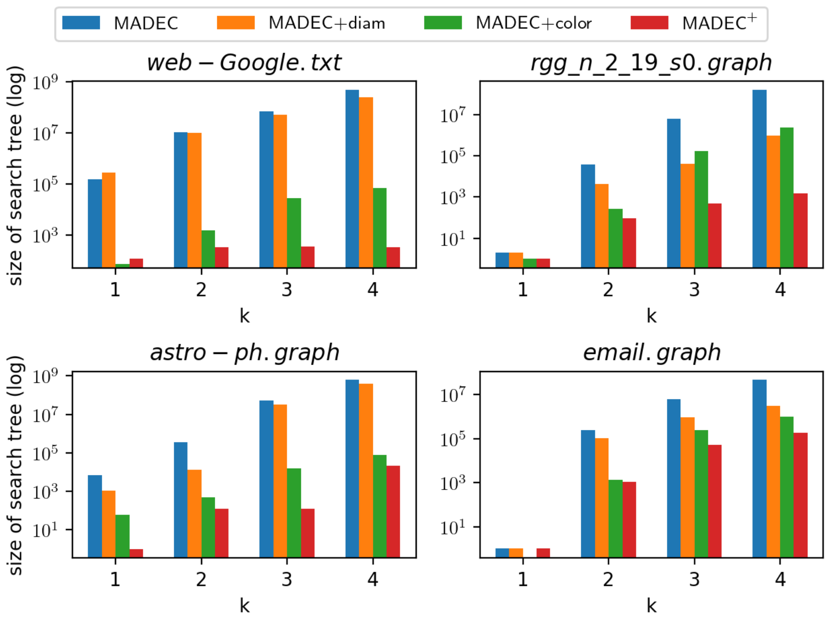

Fig. 8. Number of tree nodes (in log scale) for four large instances.

6 Conclusion

We studied the problem of obtaining the maximum k-defective clique in a graph. We presented MADEC, a branch-and-bound algorithm with a provable worst-case run time. To handle massive real-life graphs, we introduced novel reduction and branching rules, including graph preprocessing, two-hop reduction, and color-bound, and integrated them with an enhanced algorithm MADEC$^+$. This is the first time that such problem-specific rules are applied within a branch-and-bound algorithm for the MDCP.

We proposed two algorithms and assessed their performances on various graphs; furthermore, we compared them with two state-of-the-art methods. Our computational results demonstrated the superiority of MADEC$^+$ over the reference methods on most of the test graphs.

For future studies, several directions can be considered. First, the reduction rules (e.g., two-hop reduction) and bounding strategies (e.g., color-bound) introduced herein can be used to enhance other MDCP approaches, including RDS algorithms. Next, from the algorithmic perspective, similar ideas can be investigate to design effective algorithms for other relaxed clique problems. Finally, from the practical perspective, the proposed algorithms can be used to solve related applications in social networks, bio-informatics, etc.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments. This study was partially supported by the National Natural Science Foundation of China (Nos. 61802049 and 61972070) and UESTC Research Fund (No. ZYGX2018KYQD210).

References

[1] James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *Latin American Symposium on Theoretical Informatics*, pages 598–612. Springer, 2002.

[2] David A Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. Graph partitioning and graph clustering. In *10th DIMACS Implementation Challenge Workshop*, pages 13–14, 2012.

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

[4] Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research*, 138(1):21–28, 2002.

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

[6] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. Maximum weight relaxed cliques and russian doll search revisited. *Discrete Applied Mathematics*, 234:131–138, 2018.

[7] Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical Review E*, 65(2):026107, 2002.

[8] Manuel Holtgrewe, Peter Sanders, and Christian Schulz. Engineering a scalable high quality graph partitioner. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2010.

[9] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[10] Chu-Min Li, Zhiwen Fang, and Ke Xu. Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 939–946. IEEE, 2013.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] Oleg A Shirokikh. *Degree-based Clique Relaxations: Theoretical Bounds, Computational Issues, and Applications.* PhD thesis, University of Florida, 2013.

[20] 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.

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

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

[23] Qinghua Wu and Jin-Kao Hao. A clique-based exact method for optimal winner determination in combinatorial auctions. *Information Sciences*, 334:103–121, 2016.

[24] Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 253–264. ACM, 1978.

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

[26] Yi Zhou, André Rossi, and Jin-Kao Hao. Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs. *European Journal of Operational Research*, 269(3):834–843, 2018.

A Proof of Theorem 1

Proof. The general idea of the worst-case run time analysis of the proposed algorithms is based on upper bounding the size of the *search tree* generated by an algorithm. Assume that 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 on the instance with the maximum size of n. To bound $T(n)$, for a branching rule that decomposes I into l subinstances such that the size of I decreases by a_i at the least in the ith subinstance, we obtain the following 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 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 the instance with size n is $O^*(\gamma^n)$. For more details on this issue, we refer the reader to [5].

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

$$T(n) \leq T(n-1) + \ldots + T(n-(p+1)), \tag{A.1}$$

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

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

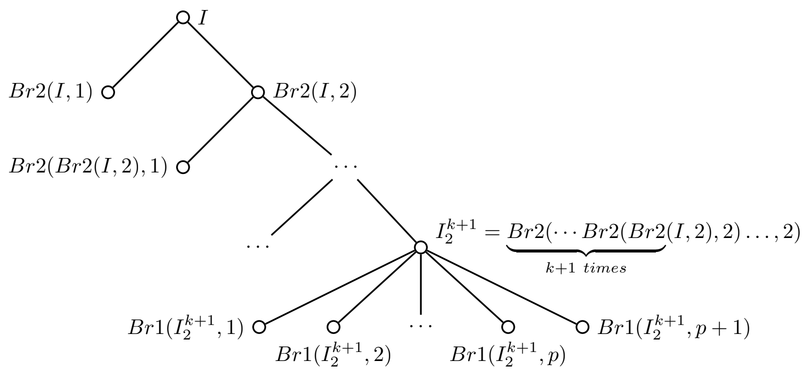

Fig. A.1. Search tree for applying Branching Rule 2 in the worst case

Meanwhile, when only Branching Rule 2 is applied in the algorithm, we obtain the following recurrence relation.

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

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

Assume an instance $I = (G = (V, E), P, k)$; and C^+ and C^- are defective and nondefective candidates of I, respectively. Suppose I satisfies $|\overline{E}(P \cup C^+)| \leq k$. Let us simulate 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 in two instances $Br2(I,1)$ and $Br2(I,2)$. Next, we denote $Br2(I,2)$ as $(G' = (V', E'), P', k)$; C'^+ and C'^- as the defective and nondefective candidate sets 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 $V \setminus P$. Consequently, $C^+ \subset C'^+$ 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, based on the observation above, we obtain

$$I_2^q = \underbrace{Br2(...Br2(Br2}_{q\ \ times}(I,2),2)\ldots,2),$$

where $q \leq k+1$ does not satisfy the condition of applying Branching Rule 2. In the worst case, q can be as large as $k+1$, and Branching Rule 1 must be used to branch I_2^{k+1} if I_2^{k+1} is not reduced. Figure A.1 shows the entire procedure.

According to the abovementioned procedure for expanding the search tree, it is safe to rewrite the recurrence relation of Branching Rule 2 as follows:

$$\begin{aligned}T(n) &\leq T(n-1) + \ldots + T(n-(k+1)) + \\ &\ldots + T(n-(k+1)-(p+1)),\end{aligned}$$

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

$$x^{2k+3} - 2x^{2k+2} + 1 = 0. \tag{A.2}$$

Λ