An Exact Algorithm with New Upper Bounds for the Maximum k-Defective Clique Problem in Massive Sparse Graphs

Jian Gao[1], Zhenghang Xu[2], Ruizhi Li[2,3], Minghao Yin[4*]

[1]College of Information Science and Technology, Dalian Maritime University, China
[2]Key Laboratory of Symbolic Computation and Knowledge Engineering Ministry of Education, Jilin University, China
[3]School of Management Science and Information Engineering, Jilin University of Finance and Economics, China
[4]School of Computer Science and Information Technology, Northeast Normal University, China
gaojian@dlmu.edu.cn, zhenghangxu97@gmail.com, lirz111@jlufe.edu.cn, ymh@nenu.edu.cn

Abstract

The Maximum k-Defective Clique Problem (MDCP), as a clique relaxation model, has been used to solve various problems. Because it is a hard computational task, previous works can hardly solve the MDCP for massive sparse graphs derived from real-world applications. In this work, we propose a novel branch-and-bound algorithm to solve the MDCP based on several new techniques. First, we propose two new upper bounds of the MDCP as well as corresponding reduction rules to remove redundant vertices and edges. The proposed reduction rules are particularly useful for massive graphs. Second, we present another new upper bound by counting missing edges between fixed vertices and an unfixed vertex for cutting branches. We perform extensive computational experiments to evaluate our algorithm. Experimental results show that our reduction rules are very effective for removing redundant vertices and edges so that graphs are reduced greatly. Also, our algorithm can solve benchmark instances efficiently, and it has significantly better performance than state-of-the-art algorithms.

Introduction

Recently, analyzing cohesive subgraphs has received much attention as it can deal with a great number of real-world applications in network analysis. The clique is a basic concept of cohesive subgraphs, while it is difficult to model real-world problems directly due to its strong restriction on connectivity. Since identifying cliques from real-world applications is an ideal task, some relaxations of the clique, such as quasi-cliques (Brunato, Hoos, and Battiti 2007), k-plexes (Balasundaram, Butenko, and Hicks 2011), k-clubs (Almeida and de Carvalho 2014), and k-defective cliques (Yu et al. 2006), were proposed to deal with real-world applications. Among those relaxations, the k-defective clique, a relation of clique, allows at most k edges to be absent from a complete graph. Therefore, it is a more useful tool for analyzing complex networks encoded from various applications compared with the clique. The concept of k-defective clique was proposed by Yu et al. (2006) to analyze proteins and predict protein interactions in biological networks. Besides, it was also employed in many fields, such as transportation science (Sherali, Smith, and Trani 2002; Sherali and Smith 2006), cluster detection (Stozhkov et al. 2020; Bomze, Rinaldi, and Zeffiro 2021), and social network analysis (Gschwind et al. 2020; Jain and Seshadhri 2020).

Trukhanov et al. (2013) proposed the first exact algorithm for the MDCP, which was based on the Russian Doll Search (RDS) (Verfaillie, Lemaître, and Schiex 1996), and then Gschwind, Irnich, and Podlinski (2018) improved it by a new incremental verification procedure with a better worst-case runtime. Shirokikh (2013) proved some upper bounds for the maximum k-defective clique for general graphs as well as some special graphs (*e.g.* planar graphs and r-partite graphs). In recent years, there have been an increasing number of works on the MDCP. Gschwind et al. (2020) proposed a framework based on the branch-and-price technique to solve the MDCP and other relaxed cliques. They implemented their algorithm with CPLEX and tested some small instances. Stozhkov et al. (2020) proposed continuous cubic formulations for the MDCP by generalizing the Motzkin-Straus formulation, and provided some theoretical results. They compared their approach with RDS on small instances. Moreover, a nonlinear optimization approach as well as an equivalent continuous formulation has been proposed for network-based cluster detection in (Bomze, Rinaldi, and Zeffiro 2021). Besides, Chen et al. (2021) have proposed an exact algorithm based on the branch-and-bound framework with some new reduction and pruning strategies, and showed their algorithm performs best among existing algorithms for massive graphs.

In the past decade, graphs from real-world applications have revealed some new characteristics, *i.e.,* massive, sparse and the power-law distribution of vertex degrees. To find the maximum (relaxed) clique or enumerate maximal (relaxed) cliques in such graphs, a number of algorithms for cliques (Rossi et al. 2014; Cai and Lin 2016; Jiang, Li, and Manyà 2017), k-plexes (Xiao et al. 2017; Miao and Balasundaram 2017; Conte et al. 2017; Zhou et al. 2021; Jiang et al. 2021), and quasi-cliques (Sanei-Mehri et al. 2021; Marinelli, Pizzuti, and Rossi 2021) have been proposed. However, compared with the clique and its other relaxations, existing k-defective algorithms can hardly handle massive sparse graphs. Though several algorithms have been proposed, the effect of them declines greatly with k growing, and there is a lack of evaluations on the cases with $k > 5$. In fact, study on algorithms for the k-defective clique in mas-

*Corresponding author

sive graphs is in its early stage, so we require more effective algorithms for massive graphs. In this paper, we propose an exact maximum k-defective clique algorithm based on the branch-and-bound framework to solve massive sparse graphs. Our contributions are summarized as follows: (1) We focus on how to reduce graphs; Different from existing approaches that only remove vertices, we propose five reduction rules for removing both redundant vertices and edges based on our two newly proposed upper bounds. (2) With those rules, we present a preprocessing method for graph reduction, and then propose a new branch-and-bound algorithm by integrating the reduction rules into branching steps. (3) To further enhance efficiency and cut more branches, another new upper bound is defined by calculating the minimum possible increment of missing edges when adding a candidate vertex to a clique. We perform computational experiments on massive sparse graphs by varying k from 1 to 30. To the best of our knowledge, this is the first time to evaluate algorithms with $k > 5$. We show our algorithm performs best on both average CPU runtimes and the number of instances solved successfully among all comparative algorithms. Furthermore, we show our new bound still works well when k grows bigger.

Preliminaries

In this section, we provide some definitions and notations. First, we only consider simple and undirected graphs in this paper. Formally, an undirected graph is defined as $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of vertices and E is a set of edges, in which an edge e is always denoted by an unordered pair of two ended vertices (v, u) from V. Moreover, we use $E(G)$ to denote the edge set of a graph.

Furthermore, we say a vertex u is a neighbor of a vertex v if there is an edge (v, u) in E. Given a graph G, the neighbor set of a vertex v is denoted by $N_G(v)$, and the degree of a vertex v is defined as the number of its neighbors, denoted as $|N_G(v)|$. Moreover, we define $N_G[v] = N_G(v) \cup \{v\}$. The notation N_G is also used to denote common neighbor set of two vertices. Given two vertices v and u, $N_G(u, v) = N_G(u) \cap N_G(v)$; and $N_G[u, v] = (N_G(u) \cap N_G(v)) \cup \{u, v\}$.

Given a subset $S \subseteq V$, $G[S] = (V', E')$ is the induced subgraph in G by S if $V' = S$ and the edge set E' consists of all the edges in E whose both vertices are in S. We use the notation $\overline{G}$ to represent a complement graph of G. The complement of a graph G is a graph on the same vertices such that two vertices of it are adjacent if and only if they are not adjacent in G.

With the above notations, we give the definition of k-defective clique.

Definition 1 (k-defective clique)**.** *Given a graph $G = (V, E)$ and a positive integer k, an induced subgraph $G[S]$ ($S \subseteq V$) is a k-defective clique if $G[S]$ has at least $\binom{|S|}{2} - k$ edges.*

We say a k-defective clique is maximal if any other k-defective clique cannot strictly contain it. The MDCP is to find the k-defective clique with the largest number (size) of vertices in a given graph. It is easy to see the parameter k is the number of allowed missing edges at most in a k-defective clique. It is pointed out that the decision version of the MDCP is NP-complete (Chen et al. 2021).

We also have to define some notations to make it easy to specify the context of a state in the branch-and-bound algorithm. We always use S to denote a fixed vertex set whose induced graph is the current k-defective clique, and we denote a candidate vertex set as C, where $C = V \backslash S$. In our algorithm, the problem is divided through branching, so we have to solve subproblems that find the maximum k-defective clique including all vertices in S. Given a graph $G = (V, E)$ and a vertex set $S \subseteq V$, the function $\omega_k(G, S)$ returns the size of the maximum k-defective clique including all vertices in S.

The notation $=_{LB}$ defines equivalence of two graphs *w.r.t.* the maximum k-defective cliques above the level of LB, which is always denote a lower bound of the maximum k-defective cliques.

Definition 2 (equivalence)**.** *Given $G = (V, E)$ and $G' = (V, E')$, suppose $S \subseteq V$ is a fixed vertex set, and LB is a positive integer, then we say $\omega_k(G, S) =_{LB} \omega_k(G', S)$ (k is a positive integer) if it satisfies one of the following conditions:*
- $\omega_k(G, S) = \omega_k(G', S)$, and $\omega_k(G, S) > LB$;
- $\omega_k(G, S) \leq LB$, and $\omega_k(G', S) \leq LB$.

Definition 2 provides an equivalence case of two graphs, and it is helpful to reduce graphs by removing edges when we try to find a k-defective clique larger than LB.

New Upper Bounds

In this section, we present some new theoretical results of upper bounds for graph reduction and cutting branches in a branch-and-bound algorithm.

Upper Bounds for Graph Reductions

We propose two upper bounds for graph reduction in this subsection. The upper bounds are discussed in the contexts of a graph $G = (V, E)$ and a positive integer k. Given a fixed vertex set S, and a vertex v in V (either in S or in a candidate vertex set $C = V \backslash S$), we define the function $rem_{G,k}(S, v) = k - |E(\overline{G}[S \cup \{v\}])|$ to calculate the remaining number of allowed missing edges required to form a complete graph if the fixed set is $S \cup \{v\}$ (add v to S if $v \notin S$). Note that we suppose $G[S \cup \{v\}]$ is a legal k-defective clique, *i.e.,* $|E(\overline{G}[S \cup \{v\}])| \leq k$, so $rem_{G,k}(S, v)$ is always a non-negative number.

Then, we consider how many extra vertices not adjacent to v can be added to S at most. The number cannot exceed $rem_{G,k}(S, v)$ because one vertex donates at least one missing edge. Also, the number cannot exceed the total number of vertices not adjacent to v, so we define $res_{G,k}(S, v) = min(rem_{G,k}(S, v), |C \backslash N_{G[C \cup \{v\}]}[v]|)$, where $min()$ returns the minimum number of its parameters. The function $UB_{G,k}$ is to compute an upper bound as follow: $UB_{G,k}(S, v) = |S \cup \{v\}| + |N_{G[C \cup \{v\}]}(v)| + res_{G,k}(S, v)$. Hence, we have Theorem 1, which indicates our first upper bound.

Theorem 1. $UB_{G,k}(S, v) \geq \omega_k(G, S \cup \{v\})$.

The proof will be omitted since it is straightforward. We then discuss the upper bound in the case of adding two vertices. Given a fixed vertex set S, a candidate vertex set $C = V \backslash S$, and two vertices u, v in V (u (or v) either in S or in C), we define the notation $rem_{G,k}(S, u, v) = k - |E(\overline{G}[S \cup \{u, v\}])|$ (suppose $S \cup \{u, v\}$ is a k-defective clique). Also, we have $res_{G,k}(S, u, v) = min(rem_{G,k}(S, u, v), |C \backslash N_{G[C \cup \{u,v\}]}[u, v]|)$. We define $UB_{G,k}(S, u, v) = |S \cup \{u, v\}| + |N_{G[C \cup \{u,v\}]}(u, v)| + res_{G,k}(S, u, v)$, so Theorem 2 indicates $UB_{G,k}(S, u, v)$ is an upper bound of k-defective cliques including u, v and S.

Theorem 2. *$UB_{G,k}(S, u, v) \geq \omega_k(G, S \cup \{u, v\})$.*

Theorem 2 is an extension of the one-vertex case, and hence no proof will be given.

Reduction Rules with Upper Bounds

Given a lower bound (integer) LB, we shall try to find a k-defective clique larger than LB or prove no such k-defective clique exists. Therefore, we can remove some vertices and edges from the graph if they are impossible to form a k-defective clique larger than LB .

The following rules can be applied in the preprocessing step, in which S is an empty set.

Rule 1. *Remove the vertex v from $G = (V, E)$, if it satisfies $UB_{G,k}(\emptyset, v) \leq LB$.*

In fact, this rule removes the vertex if its degree is too low to form a k-defective clique larger than LB according to Theorem 1. Then, we show how to remove edges from the graph via our upper bound. We give the following theorem about equivalence between two graphs. It is useful for edge reduction.

Theorem 3. *Given a fixed vertex set S in $G = (V, E)$, a positive integer k and two vertices u and v s.t. $(u, v) \in E$, $\omega_k(G, S) =_{LB} \omega_k(G', S)$ if $UB_{G,k}(S, u, v) \leq LB$, where $G' = (V, E \backslash \{(v, u)\})$.*

Proof. It is obvious that $\omega_k(G, S) \geq \omega_k(G', S)$ since G has exactly one more edge than G'. On the one hand, if $\omega_k(G, S) \leq LB$, then $\omega_k(G', S) \leq LB$. On the other hand, if $\omega_k(G, S) > LB$, any k-defective clique in G, including all vertices in S and larger than LB, is also a k-defective clique in G'. Because $UB_{G,k}(S, u, v) \leq LB$, u and v cannot be included such a k-defective clique at the same time, and thus we have $\omega_k(G, S) = \omega_k(G', S)$. Hence, Theorem 3 is proved. □

Theorem 3 indicates that removing the edge between u and v from G does not change the maximum k-defective clique if the clique has more than LB vertices. Hence, when we try to find a k-defective clique with at least $LB + 1$ vertices, we can search on the graph excluding the edge between u and v from G if $UB_{G,k}(S, u, v) \leq LB$. Therefore, we have a rule for removing edges in the preprocessing step.

Rule 2. *Remove the edge (u, v) from $G = (V, E)$, if it satisfies $UB_{G,k}(\emptyset, u, v) \leq LB$.*

Our upper bounds also work in a branch-and-bound algorithm. We will show how to make graph reduction during search. We consider the context of selecting a vertex v and adding it to S. In that context, we can check an unfixed vertex u by calculating $UB_{G,k}(S, u, v)$. It is clear u can be removed if u and v cannot form a larger k-defective clique than LB. Rule 3 gives the formal description.

Rule 3. *Remove the vertex $u \in C$ from $G = (V, E)$, if it satisfies $UB_{G,k}(S, u, v) \leq LB$, where v is a vertex in the fixed vertex set S and $C = V \backslash S$.*

Moreover, in a search tree node, we can check whether a vertex $v \in C$ can be removed according to Theorem 1.

Rule 4. *Remove the vertex $v \in C$ from $G = (V, E)$, if it satisfies $UB_{G,k}(S, v) \leq LB$, where S is the fixed vertex set and $C = V \backslash S$.*

Another important case is that both u and v are in the candidate set C. In that case, we can check whether the edge between them can be removed.

Rule 5. *Remove the edge (u, v) ($u, v \in C$) from $G = (V, E)$, if it satisfies $UB_{G,k}(S, u, v) \leq LB$, where S is the fixed vertex set and $C = V \backslash S$.*

Note that all rules above can be done in $O(d)$ time (d is the maximum degree in G), as we can compute $UB_{G,k}$ by checking a vertex's neighbors or two vertices' common neighbors. Besides, we employ some straightforward rules, which were also employed in previous works (Chen et al. 2021). For example, if a vertex v is adjacent to all other vertices, it can be added to any k-defective clique; all vertices in the candidate set C can be added to the fixed set if after adding them the fixed set still forms a k-defective clique.

The Candidate-based Upper Bound

In this subsection, we propose a new upper bound based on counting the least increment on missing edges by trying to fix each candidate vertex, and then employ it to cut branches. Similar to upper bounds for reduction, we discuss our new upper bound in the context of a given fixed vertex set S and a candidate set $C = V \backslash S$. First, we sort vertices in C by a non-decreasing order *w.r.t.* $|N_{\overline{G}[S \cup \{v\}]}(v)|$ ($v \in C$), *i.e.,* sort by the number of missing edges between v and S. We then define an ordered set of C as $ord(C) = \{u_1, u_2, ..., u_{|C|}\}$ such that for any pair of u_i and u_j, we have $|N_{\overline{G}[S \cup \{u_i\}]}(u_i)| \leq |N_{\overline{G}[S \cup \{u_j\}]}(u_j)|$ if $i < j$.

With the ordered set $ord(C)$, we discuss the least increment on the number of missing edges after we add some vertices to S. It is easy to see when we add a candidate vertex v, the least increment is $|N_{\overline{G}[S \cup \{v\}]}(v)|$. Also, it is easy to see the first vertex in $ord(C)$ has the smallest value of the least increment. We can obtain a lower bound of the smallest increment. When we add several vertices at a time, each vertex v donates at least $|N_{\overline{G}[S \cup \{v\}]}(v)|$ missing edges, so we can easily calculate the total number of missing edges between the added vertices and fixed vertices as the lower bound. Moreover, if we add i vertices to S at a time, it is clear that selecting the first i vertices in $ord(C)$ can minimize the increment, because the first vertex in $ord(C)$, the smaller possible increment we have. We introduce the function inc to denote how many missing edges increased at least, and define $inc(ord(C), i) = \sum_{j=1}^{i} |N_{\overline{G}[S \cup \{u_j\}]}(u_j)|$,

$(u_j \in ord(C))$.

Afterwards, we define our candidate-based upper bound as:

$$candibound(G, S, k) = |S| + c,$$

where

$$c = \begin{cases} max(i) & \forall i \in I, \text{if } I \neq \emptyset; \\ 0, & \text{if } I = \emptyset, \end{cases}$$

and an integer set $I = \{ i \mid inc(ord(C), i) \leq k - |E(\overline{G}[S])| \ (1 \leq i \leq m)\}$, where $m = |ord(C)|$ and $C = V \backslash S$.

An integer i in the set I means that the smallest possible increment does not exceed the remaining number of allowed missing edges when we add the first i vertices. Therefore, the maximum number in I is the maximum possible number of vertices, denoted by c, can be added to S if $I \neq \emptyset$; otherwise the number c is 0. Hence, the function $candibound(G, S, k)$ defines the maximum possible number of vertices in a k-defective clique including S. Therefore, we have Theorem 4 as follow.

Theorem 4. $candibound(G, S, k) \geq \omega_k(G, S)$.

KDBB: An Exact Algorithm

The Preprocessing Method

Preprocessing is a necessary step to solve massive sparse graphs, since with an initial lower bound it can reduce the graph greatly. We use this technique in our algorithm, and present a preprocessing method based on our reduction rules. Algorithm 1 gives the detailed procedure. The function *preprocessing* takes a graph G, a positive integer k, and a lower bound LB as inputs. Two sub-functions are introduced here for vertex and edge reduction, respectively. With the initial bound, Rule 1 is performed iteratively in function *check_vertex* until no vertex can be removed. After that, the function *check_edge* is performed, and Rule 2 for edge reduction is performed until no redundant edge exists. Then, we obtain a reduced graph without redundant edges, but edge reduction does not remove any vertex, so we should check the graph again to remove vertices by Rule 1 (lines 3-4). Finally, the function returns the reduced graph G.

Algorithm 1: Function *preprocessing*(G,k,LB)

Input: $G = (V, E)$, a positive integer k, a lower bound LB;
Output: the reduced graph G.

```
1  G ← check_vertex(G, k, LB);
2  G ← check_edge(G, k, LB);
3  foreach v ∈ V do
4      apply Rule 1 on v with LB and k;
5  return G;
6  Function check_vertex(G, k, LB)
7  Q ← ∅; add all vertices in V to Q;
8  while Q is not empty do
9      v ← pop(Q), N_v ← N_G(v);
10     apply Rule 1 on v with LB and k;
11     if v is removed then
12         add all vertices in N_v to Q;
13 return G;
14 Function check_edge(G, k, LB)
15 Q ← ∅; add all edges in E to Q;
16 while Q is not empty do
17     (u, v) ← pop(Q);
18     apply Rule 2 on (u, v) with LB and k;
19     if (u, v) is removed then
20         add all edges of u and v to Q;
21 return G;
```

The Branch-and-Bound Algorithm

We present the framework of our algorithm (Algorithm 2) in this subsection. The algorithm starts with finding an initial lower bound by a heuristic strategy. Here we employ the method called *FastLB*. It is a classic method for the maximum clique (Rossi et al. 2014), and was used to construct a clique so as to find a lower bound of k-defective cliques (Chen et al. 2021). Then, the algorithm calls the function *preprocessing* to reduce the input graph. After that, it begins to perform branching. The function *branch-and-bound* is called as the root node (line 4). In the function, it checks whether S can form a k-defective clique, and then it applies reduction rules to further reduce graphs by calling a function *reduction* (we will introduce it later). Afterwards, it returns the size of S if the reduced graph only contains vertices in S. Otherwise, it calculates an upper bound with G and S. After selecting a vertex v, we use a binary branching strategy in the search tree node, *i.e,* either adding the vertex v to S or removing it from the graph G, each corresponding to a branch. Therefore, search tree nodes can be expanded recursively by the function *branch-and-bound* until there is no vertex left in the candidate set. Finally, the function will backtrack and return the best lower bound. There are some crucial components in the function *branch-and-bound* to be explained.

The most important component is the function *reduction*, which is shown in Algorithm 3. Three rules are considered here. On the one hand, after a new vertex v is added into S, we can check whether a candidate vertex is consistent with v, so each vertex u in the candidate set is checked by Rule 3 (line 3). On the other hand, we check vertices in the candidate set to remove redundant edges. It is desirable to remove all redundant edges, but the cost on the checking is very expensive because all edges should be considered. In fact, we have deleted redundant edges in the preprocessing step, so remaining edges probably satisfy the upper bound if their neighbors are not further removed. Therefore, to speed up reduction, we only check vertices whose neighbors have been added to S or removed in the parent search tree node (line 5). For each neighbor u of w, we apply Rule 5 to try to remove the edge (u, w). To further improve the efficiency, we check the degree of w and remove it if Rule 4 is satisfied (line 8), since vertex deletion is more efficient than deleting edges one by one. Moreover, if w is removed, we stop checking w's neighbors and continue to check the next vertex in $V \backslash S$ (line 9). Finally, the reduced graph G is returned.

Algorithm 2: *KDBB*(G, k)

Input: $G = (V, E)$, the problem parameter k;
Output: the size of the maximum k-defective clique.

$S \leftarrow \emptyset$;
initialize LB by the heuristic method ***FastLB***;
$G \leftarrow$ *preprocessing*(G, k, LB);
$LB \leftarrow$ *branch-and-bound*$(G, S, null, k, LB)$;
return LB;

Function *branch-and-bound*(G, S, v, k, LB)
if $|E(\overline{G}[S])| > k$ **then return** LB ;
if $v \neq null$ **then** $G \leftarrow reduction(G, S, v, k, LB)$;
if $V \backslash S = \emptyset$ **then return** $|S|$;
$UB \leftarrow candibound(G, S, k)$;
if $UB > LB$ **then**
    select a vertex u in $V \backslash S$;
    $size \leftarrow$ *branch-and-bound*$(G, S \cup \{u\}, u, k, LB)$;
    if $LB < size$ **then** $LB \leftarrow size$;
    remove u from G;
    $size \leftarrow$ *branch-and-bound*(G, S, u, k, LB);
    if $LB < size$ **then** $LB \leftarrow size$;
return LB;

Algorithm 3: Function *reduction*(G, S, v, k, LB)

Input: $G = (V, E)$, the fixed vertex set S, the selected vertex v , the parameter k, and the lower bound LB;
Output: the reduced graph G.

if $v \in S$ **then**
    foreach *vertex* u *in* $V \backslash S$ **do**
        apply Rule 3 on u with LB, S, k, and v;
foreach *vertex* w *in* $V \backslash S$ **do**
    if *a neigbor of* w *is removed in the parent node or* v *is a neigbor of* w **then**
        foreach *vertex* u *in* $V \backslash S$ *s.t.* $(u, w) \in E$ **do**
            apply Rule 5 on (u, w) with LB, S, and k;
            apply Rule 4 on w with LB, S, and k;
            if w *is removed* **then** break;
return G;

The upper bound for cutting branches is also important in our algorithm. After applying reduction rules on the graph, the algorithm calculates an upper bound and checks whether the bound is larger than the current lower bound. If so, it goes on to search the subtree. We use the candidate-based upper bound we propose in the previous section, and we will analyze the effectiveness and show the power of the upper bound as k grows in the experiment part.

In our candidate-based bound computation, we divide candidate vertex set $C = V \backslash S$ into $k+1$ subsets $C_0, ..., C_k$, where $C_i = \{v \in C \mid |N_{\overline{G}[S \cup \{v\}]}(v)| = i\}$, instead of sorting C. Then, we check $C_0, C_1, ..., C_k$ sequentially. Suppose r (initialized by $k - |E(\overline{G}[S])|$) is the remaining number of allowed missing edges. If $r \geq i|C_i|$, we subtract $i|C_i|$ from r, where $i|C_i|$ is the least increment of missing edges for adding all vertices in C_i, and then we add $|C_i|$ to the bound UB (initialized by 0); otherwise, we add $\lfloor r/i \rfloor$ to UB, and thus we obtain the upper bound. We then stop checking remaining sets. For each vertex in C, we should check all its neighbors, so our implementation can be achieved in $O(d|V|)$ time, where d is the maximum degree of G.

The vertex selection heuristic is another important strategy that impacts on the search tree size greatly. We use a dynamic approach to determine which one will be picked out at a decision node. In our strategy, the vertex with the maximum degree in the induced graph $G[V \backslash S]$ is selected, breaking ties randomly.

Experiments

In this section, we perform computational experiments to test our algorithm. We use massive sparse graphs as benchmarks, where several datasets are selected. The first one is Facebook social networks including 114 instances[1]. The second one is massive sparse graphs from real-world applications. Among which, 139 graphs, originally from the Network Data Repository online (Rossi and Ahmed 2015), were frequently tested in previous works (Rossi and Ahmed 2014; Cai 2015; Lin et al. 2017; Jiang et al. 2021)[2]. The third one is a set of massive graphs from DIMACS10[3] and SNAP[4] benchmarks, where 37 instances were tested the same as the instances used in (Chen et al. 2021).

Our algorithm KDBB (the maximum K-Defective clique algorithm with Branch-and-Bound) was implemented in C++ language and compiled by g++ 4.8.5 with -O3 option. Benchmarks were solved on a workstation with an Intel(R) Xeon(R) E7-4820 v2 (2.00GHz) CPU, and 256GB RAM, running CentOS Linux 7.7.1908 (Core). Each instance was solved with the cutoff time 10800s (3 hours). As we stated, MADEC$^+$ and RDS are the state-of-the-art exact algorithms for the MDCP (Chen et al. 2021; Gschwind, Irnich, and Podlinski 2018), so we mainly compare our algorithm with them. All source codes of the comparative algorithms were downloaded from GitHub[5].

Preprocessing Results

In this subsection, we analyze reduction effects of our preprocessing method. We select Facebook benchmarks as an example, where instances with more than 25000 vertices are counted (we exclude instances that cannot be solved within the cutoff time). Table 1 gives the detailed results of the number of vertices and edges before and after reduction when $k = 1$, where the column "vertex reduction" only uses *check_vertex*, "vertex&edge reduction" uses both *check_vertex* and *check_edge*, and the column "ratio" is the ratio of the number of vertices in reduced graphs to the original vertex number. It is obvious that *check_edge* can further reduce the graph greatly compared with the results of *check_vertex*. For large-scale graphs, the preprocessing method can obtain a reduced graph with the ratio below 5%.

[1] https://networkrepository.com/socfb.php
[2] http://lcs.ios.ac.cn/~caisw/Resource/realworld%20.graphs.tar.gz
[3] https://www.cc.gatech.edu/dimacs10/downloads.shtml
[4] http://snap.stanford.edu/data/
[5] https://github.com/chenxiaoyu233/k-defective

In fact, the method is still effective beyond $k = 1$, and even for $k = 10$, it can obtain a ratio of 19% on average.

	original graph		vertex reduction		vertex&edge reduction		
instance (socfb-)	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	ratio (%)
FSU53	27737	1034802	7585	464509	525	24330	1.893
Indiana69	29747	1305765	17386	1036472	1225	43354	4.118
Indiana	29732	1305757	17386	1036472	1225	43354	4.120
Michigan23	30147	1176516	16337	932524	785	29750	2.604
MSU24	32375	1118774	13592	692484	375	12444	1.158
OR	63392	816886	12942	442781	858	20832	1.353
Penn94	41536	1362220	20353	978684	487	13697	1.172
Texas80	31560	1219650	9832	624884	122	6084	0.387
Texas84	36364	1590651	18075	1142849	799	35691	2.197
UF21	35123	1465660	14613	929463	1526	70410	4.345
UF	35111	1465654	14613	929463	1526	70410	4.346
UIllinois20	30809	1264428	12067	743461	406	17969	1.318
UIllinois	30795	1264421	12067	743461	406	17969	1.318
wosn-friends	63731	817090	12942	442781	858	20832	1.346

Table 1: Comparison of graph size before and after preprocessing ($k = 1$)

Comparative Results with Existing Algorithms

We then compare our algorithm with the state-of-the-art algorithms. Though MADEC^+ and RDS are taken for comparison, it is noted that without our preprocessing method the two comparative algorithms always fail to solve most instances when k>3. Therefore, to make an intensive analysis, we combine MADEC^+ and RDS with our preprocessing method, so two enhanced versions, denoted by $\mathrm{MADEC}^+_{\mathrm{P}}$ and $\mathrm{RDS}_{\mathrm{P}}$ respectively, are also compared in our experiment. Hence, five algorithms are taken into consideration.

First, we provide summary results on three groups of datasets, *i.e.,* the number of instances that can be solved by an algorithm in limited time. Tables 2-4 indicate the results for Facebook benchmarks, 139 massive sparse graphs and 37 instances from DIMACS10 and SNAP benchmarks. The results are grouped by each value of $k = 1, 3, 5, 10, 15, 20$, respectively.

	KDBB	MADEC^+	$\mathrm{MADEC}^+_{\mathrm{P}}$	RDS	$\mathrm{RDS}_{\mathrm{P}}$
k = 1	110	31	110	16	103
k = 3	110	1	104	1	50
k = 5	108	0	78	0	14
k = 10	109	0	9	0	0
k = 15	103	0	0	0	0
k = 20	80	0	0	0	0

Table 2: Statistic results of the number of successfully solved instances for Facebook instances

From the tables, we can see our algorithm KDBB can solve far more instances than others on all datasets. Also, for each k, our algorithm solves the most number of instances. Without our preprocessing method, MADEC^+ and RDS cannot solve any Facebook instance when $k \geq 5$, and with our preprocessing method they can solve more instances,

	KDBB	MADEC^+	$\mathrm{MADEC}^+_{\mathrm{P}}$	RDS	$\mathrm{RDS}_{\mathrm{P}}$
k = 1	117	86	115	80	111
k = 3	107	64	94	56	76
k = 5	104	54	81	42	57
k = 10	85	30	36	21	21
k = 15	68	22	26	11	12
k = 20	56	17	20	7	8

Table 3: Statistic results of the number of successfully solved instances for 139 massive sparse graphs

	KDBB	MADEC^+	$\mathrm{MADEC}^+_{\mathrm{P}}$	RDS	$\mathrm{RDS}_{\mathrm{P}}$
k = 1	36	29	36	26	36
k = 3	35	24	31	22	31
k = 5	34	23	28	19	23
k = 10	30	12	15	12	12
k = 15	25	8	10	6	6
k = 20	22	5	6	3	3

Table 4: Statistic results of the number of successfully solved instances for DIMACS10 and SNAP instances

but they are not comparable with KDBB. Besides, we tested benchmark graphs by CPLEX as Chen et al. (2021) did, but it failed to solve most massive graphs, so we do not report its results.

Furthermore, we show the detailed runtime results of Facebook instances. We only show Facebook instances with more than 15000 vertices (excluding instances cannot be solved by any algorithm), and thus select out 37 instances. Table 5 indicates CPU runtimes with k=1, 3, 5, 10. Since $\mathrm{MADEC}^+_{\mathrm{P}}$ and $\mathrm{RDS}_{\mathrm{P}}$ fail to solve most instances if $k > 10$, we omit results of k=15, 20. We also omit results of MADEC^+ and RDS, and instead we list $\mathrm{MADEC}^+_{\mathrm{P}}$ and $\mathrm{RDS}_{\mathrm{P}}$. In the table, N/A means a failure run due to out of time or memory. Clearly, KDBB is the fastest one for most instances. When $k = 1$, runtimes of KDBB and $\mathrm{MADEC}^+_{\mathrm{P}}$ are close, but as k increases the difference between two algorithms is enlarged. For example, KDBB can solve a large part of instances below 1000s for $k = 5$, but $\mathrm{MADEC}^+_{\mathrm{P}}$ requires thousands of seconds for more than half of those instances. KDBB can still solve those instances when k grows to 10 whereas $\mathrm{MADEC}^+_{\mathrm{P}}$ fails to solve most of them. Moreover, $\mathrm{RDS}_{\mathrm{P}}$ is not comparable with KDBB and $\mathrm{MADEC}^+_{\mathrm{P}}$ even for $k = 1$.

We also count the search tree sizes of those algorithms for the instances in Table 5. The number of tree nodes for each run is reported in Table 6, where the tree sizes are listed in thousands and N/A means a failure run due to out of time or memory. For $k = 1$, $\mathrm{MADEC}^+_{\mathrm{P}}$ has a smaller average tree size than KDBB. For the groups of $k > 1$, however, KDBB has smaller tree sizes for most of those instances. It is clear that the gap between the two algorithms becomes wide as k grows. This result is consistent with the CPU runtimes listed in Table 5. Moreover, $\mathrm{RDS}_{\mathrm{P}}$ visited far more tree nodes to complete searching compared to $\mathrm{MADEC}^+_{\mathrm{P}}$ and KDBB.

In addition, we take 4 instances as examples to show trends of CPU runtimes by varying k from 1 to 30 at increment of 1. Figure 1 depicts the curves of three algo-

	$k=1$			$k=3$			$k=5$			$k=10$		
instance	KDBB	$\text{MADEC}_{\text{P}}^{+}$	RDS_{P}	KDBB	$\text{MADEC}_{\text{P}}^{+}$	RDS_{P}	KDBB	$\text{MADEC}_{\text{P}}^{+}$	RDS_{P}	KDBB	$\text{MADEC}_{\text{P}}^{+}$	RDS_{P}
socfb-Auburn71	**431.82**	434.32	9344.55	**535.67**	666.94	N/A	**639.28**	5254.52	N/A	**1194.54**	N/A	N/A
socfb-Berkeley13	**425.01**	430.04	1170.46	**452.45**	573.67	N/A	**505.92**	9644.20	N/A	**629.82**	N/A	N/A
socfb-BU10	**252.48**	253.05	550.18	**289.99**	295.62	N/A	**331.91**	410.58	N/A	**369.62**	N/A	N/A
socfb-Cornell5	**393.10**	431.78	N/A	**921.53**	3583.99	N/A	**1265.04**	N/A	N/A	**2636.48**	N/A	N/A
socfb-FSU53	208.65	**194.91**	4120.45	610.09	**356.71**	N/A	**827.74**	2628.48	N/A	**1400.47**	N/A	N/A
socfb-Harvard1	**346.56**	347.01	1160.67	**420.73**	504.22	N/A	**516.89**	N/A	N/A	**1354.25**	N/A	N/A
socfb-Indiana69	**1133.84**	1155.64	2406.44	**1071.99**	1186.98	N/A	**1185.71**	2673.82	N/A	**1320.53**	N/A	N/A
socfb-Indiana	**1142.24**	1163.96	2422.74	**1138.33**	1253.55	N/A	**1261.44**	2754.66	N/A	**1420.77**	N/A	N/A
socfb-Maryland58	149.80	**149.63**	235.95	161.77	**161.34**	1114.20	**185.48**	200.69	N/A	**239.30**	4785.85	N/A
socfb-Michigan23	**832.79**	837.27	1451.63	**1072.10**	1249.84	N/A	**970.61**	6060.48	N/A	**1383.89**	N/A	N/A
socfb-MSU24	**493.45**	493.69	551.56	**576.35**	583.73	9406.49	**665.95**	1085.90	N/A	**878.53**	N/A	N/A
socfb-MU78	181.78	**181.57**	193.19	**199.89**	203.42	605.46	**215.22**	397.22	N/A	**306.05**	N/A	N/A
socfb-NYU9	**348.54**	348.79	354.30	**398.52**	407.73	1492.59	**396.44**	589.08	N/A	**466.15**	N/A	N/A
socfb-Oklahoma97	383.00	**322.40**	N/A	2047.65	**1166.24**	N/A	**3938.08**	N/A	N/A	**6926.11**	N/A	N/A
socfb-OR	**356.19**	368.25	696.82	**455.70**	1683.91	N/A	**587.24**	N/A	N/A	**1485.83**	N/A	N/A
socfb-Penn94	**1138.78**	1140.64	1233.24	**1556.72**	1580.36	N/A	**1820.09**	2518.84	N/A	**1971.97**	N/A	N/A
socfb-Rutgers89	218.92	**218.84**	223.65	**276.22**	276.54	400.15	**278.99**	332.03	N/A	**386.18**	N/A	N/A
socfb-Tennessee95	246.35	**242.59**	6092.39	**361.29**	402.39	N/A	**423.89**	5583.14	N/A	**553.96**	N/A	N/A
socfb-Texas80	341.82	**339.87**	470.02	422.86	**408.56**	2254.93	**533.85**	1143.19	N/A	**753.08**	N/A	N/A
socfb-Texas84	1490.20	**1475.98**	N/A	1674.41	**1634.17**	N/A	**2769.26**	N/A	N/A	**10252.70**	N/A	N/A
socfb-UC33	**155.55**	156.44	854.40	**170.87**	176.97	N/A	**181.06**	475.71	N/A	**262.87**	N/A	N/A
socfb-UCLA26	**184.50**	184.52	185.69	**207.02**	207.22	233.95	**215.07**	219.47	N/A	**288.32**	3652.25	N/A
socfb-UCLA	**190.21**	190.23	191.41	**205.56**	205.76	232.68	**237.37**	241.77	N/A	**289.83**	3636.10	N/A
socfb-UConn91	104.97	**104.93**	108.89	**123.09**	123.34	1008.57	**172.63**	192.13	N/A	**208.27**	N/A	N/A
socfb-UConn	109.09	**109.04**	113.08	**125.95**	126.20	1016.88	**168.66**	188.16	N/A	**194.22**	N/A	N/A
socfb-UF21	786.98	**786.07**	N/A	**1297.48**	1434.09	N/A	**1542.10**	N/A	N/A	**2571.06**	N/A	N/A
socfb-UF	793.23	**791.06**	N/A	**1331.76**	1469.12	N/A	**1601.77**	N/A	N/A	**2579.11**	N/A	N/A
socfb-UGA50	723.90	**709.32**	N/A	**1467.17**	3209.00	N/A	**2458.83**	N/A	N/A	**6793.84**	N/A	N/A
socfb-UIllinois20	486.01	**481.63**	2215.92	609.77	**580.99**	N/A	**784.10**	2047.08	N/A	**1217.39**	N/A	N/A
socfb-UIllinois	486.06	**481.69**	2212.43	643.97	**616.32**	N/A	**806.13**	2073.49	N/A	**1244.94**	N/A	N/A
socfb-UMass92	**225.75**	226.59	243.97	**245.18**	268.70	N/A	**265.19**	888.08	N/A	**318.00**	N/A	N/A
socfb-UNC28	236.01	**235.62**	263.98	286.83	**283.44**	637.27	**335.61**	341.38	2237.48	**380.12**	9840.37	N/A
socfb-USC35	231.73	**229.53**	1124.37	266.70	**258.85**	N/A	**333.99**	1493.47	N/A	**408.71**	N/A	N/A
socfb-UVA16	**341.48**	341.97	547.90	**387.41**	520.16	N/A	**400.43**	7127.35	N/A	**551.93**	N/A	N/A
socfb-Virginia63	83.54	**83.32**	90.11	102.84	**100.89**	160.39	**143.49**	149.76	381.83	**214.84**	N/A	N/A
socfb-Wisconsin87	**532.00**	535.77	1355.82	**612.05**	699.23	N/A	**664.18**	6234.68	N/A	**923.55**	N/A	N/A
socfb-wosn-friends	**374.51**	386.08	713.63	**438.38**	1660.01	N/A	**533.33**	N/A	N/A	**1259.73**	N/A	N/A

Table 5: CPU runtimes of three algorithms on Facebook instances in seconds with cut-off time of 10800s

rithms. We can see that our algorithm can successfully solve instances under each value of k, even for $k = 30$, but $\text{MADEC}_{\text{P}}^{+}$ and RDS_{P} fail when $k > 11$. As k grows, runtimes of our algorithm increase slowly and for most cases it can complete search within 1000s, but we can see that there is a very sharp growth of runtimes for the other two algorithms.

Analysis of the Candidate-based Bound

In this subsection, we show the effect of our candidate-based bound. Table 7 gives the statistic results of Facebook benchmark, where the number of solved instances and average runtime are grouped by k (we set runtimes to 10800s for unsolved instances). In the previous work (Chen et al. 2021), the c-color bound is used as the main strategy to cut branches. Two versions therefore are compared: KDBB that uses our candidate-based bound, and KDBB with the c-color bound instead of our bound, denoted by $\text{KDBB}_{\text{color}}$. Both algorithms can solve the same number of instances when

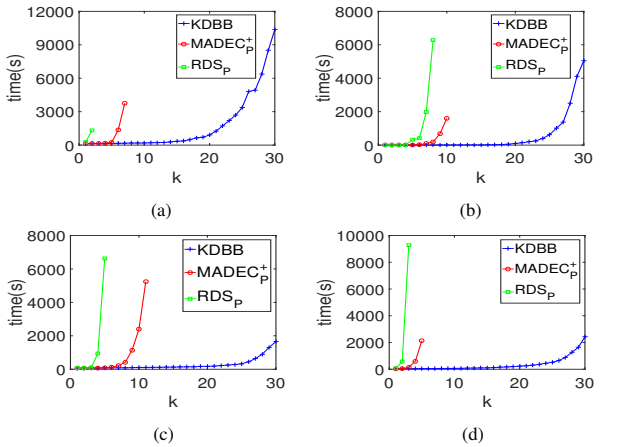

Figure 1: Runtime tendency with k growing. Four instances are: (a) Northwestern25; (b) Trinity100; (c) Bingham82; (d) USF51.

	k = 1			k = 3			k = 5			k = 10		
instance	KDBB	$\text{MADEC}^+_{\text{P}}$	RDS_{P}	KDBB	$\text{MADEC}^+_{\text{P}}$	RDS_{P}	KDBB	$\text{MADEC}^+_{\text{P}}$	RDS_{P}	KDBB	$\text{MADEC}^+_{\text{P}}$	RDS_{P}
socfb-Auburn71	29.6	**9.4**	31990.2	**212.5**	630.5	N/A	**398.7**	11578.6	N/A	**1386.5**	N/A	N/A
socfb-Berkeley13	4.9	**4.9**	2739.2	**21.3**	383.1	N/A	**30.4**	27755.0	N/A	**104.5**	N/A	N/A
socfb-BU10	2.3	**2.1**	2696.6	**12.3**	28.7	N/A	**29.0**	397.9	N/A	**88.6**	N/A	N/A
socfb-Cornell5	**21.5**	72.8	N/A	**190.2**	6961.8	N/A	**222.6**	N/A	N/A	**401.3**	N/A	N/A
socfb-FSU53	58.7	**5.9**	22337.7	467.3	**323.8**	N/A	**677.2**	8751.3	N/A	**1636.4**	N/A	N/A
socfb-Harvard1	**2.4**	6.5	4759.2	**30.5**	579.4	N/A	**74.8**	N/A	N/A	**278.1**	N/A	N/A
socfb-Indiana69	6.9	**6.6**	2250.2	**28.6**	161.3	N/A	**48.5**	2146.5	N/A	**148.0**	N/A	N/A
socfb-Indiana	6.9	**6.6**	2250.2	**28.6**	161.3	N/A	**48.5**	2146.5	N/A	**148.0**	N/A	N/A
socfb-Maryland58	3.1	**1.0**	1274.7	11.6	**10.3**	12423.2	**17.7**	125.8	N/A	**53.6**	25980.3	N/A
socfb-Michigan23	**5.4**	5.7	2144.4	**21.5**	427.1	N/A	**34.3**	9711.5	N/A	**99.5**	N/A	N/A
socfb-MSU24	6.2	**1.5**	377.5	**33.2**	56.4	17396.3	**60.2**	1588.5	N/A	**109.7**	N/A	N/A
socfb-MU78	1.9	**1.5**	209.7	**13.2**	76.0	6167.9	**29.3**	1823.6	N/A	**147.1**	N/A	N/A
socfb-NYU9	**1.5**	2.1	55.4	**3.7**	64.0	3670.7	**4.4**	1154.4	N/A	**16.6**	N/A	N/A
socfb-Oklahoma97	236.3	**23.9**	N/A	**1702.5**	2233.8	N/A	**2933.2**	N/A	N/A	**6562.4**	N/A	N/A
socfb-OR	**7.9**	22.8	708.7	**55.8**	3025.6	N/A	**99.5**	N/A	N/A	**386.7**	N/A	N/A
socfb-Penn94	**3.2**	3.3	420.6	**10.1**	56.0	N/A	**15.1**	1671.4	N/A	**26.4**	N/A	N/A
socfb-Rutgers89	1.5	**0.6**	69.5	**7.8**	15.2	1243.5	**11.3**	445.2	N/A	**39.3**	N/A	N/A
socfb-Tennessee95	62.9	**8.1**	32961.4	**379.0**	500.2	N/A	**456.3**	16405.9	N/A	**713.0**	N/A	N/A
socfb-Texas80	19.0	**2.9**	2127.1	**130.9**	189.7	26287.1	**302.9**	5040.9	N/A	**795.0**	N/A	N/A
socfb-Texas84	66.2	**14.8**	N/A	**584.5**	1319.6	N/A	**1227.8**	N/A	N/A	**6554.9**	N/A	N/A
socfb-UC33	3.8	**2.1**	4530.8	**16.8**	41.5	N/A	**24.1**	1502.6	N/A	**66.8**	N/A	N/A
socfb-UCLA26	0.4	**0.3**	15.7	**2.9**	4.7	193.8	**5.4**	43.2	N/A	**20.0**	19950.7	N/A
socfb-UCLA	0.4	**0.3**	15.7	**2.9**	4.7	193.8	**5.4**	43.2	N/A	**20.0**	19950.7	N/A
socfb-UConn91	2.7	**1.0**	58.6	**7.3**	7.9	4349.5	**7.8**	155.7	N/A	**21.7**	N/A	N/A
socfb-UConn	2.7	**1.0**	58.6	**7.3**	7.9	4349.5	**7.8**	155.7	N/A	**21.7**	N/A	N/A
socfb-UF21	101.6	**15.7**	N/A	**596.4**	1035.6	N/A	**846.9**	N/A	N/A	**2368.4**	N/A	N/A
socfb-UF	101.6	**15.7**	N/A	**596.4**	1035.6	N/A	**846.9**	N/A	N/A	**2368.4**	N/A	N/A
socfb-UGA50	87.6	**23.2**	N/A	**759.9**	5506.6	N/A	**1617.4**	N/A	N/A	**5204.3**	N/A	N/A
socfb-UIllinois20	31.6	**4.1**	12408.2	**239.4**	254.3	N/A	**389.6**	6028.7	N/A	**1067.8**	N/A	N/A
socfb-UIllinois	31.6	**4.1**	12408.2	**239.4**	254.3	N/A	**389.6**	6028.7	N/A	**1067.8**	N/A	N/A
socfb-UMass92	2.8	**2.5**	85.0	**6.9**	93.9	N/A	**12.9**	2573.9	N/A	**46.8**	N/A	N/A
socfb-UNC28	8.2	**0.9**	425.8	34.6	**12.6**	4328.5	**67.3**	189.6	18313.8	**161.5**	38954.8	N/A
socfb-USC35	30.8	**3.8**	10443.3	**143.4**	185.9	N/A	**178.8**	5857.6	N/A	**249.5**	N/A	N/A
socfb-UVA16	6.0	**3.9**	1524.7	**27.9**	498.4	N/A	**61.7**	19224.9	N/A	**360.8**	N/A	N/A
socfb-Virginia63	5.1	**0.3**	153.2	43.8	**17.4**	1295.6	**98.5**	207.9	4940.8	**275.9**	N/A	N/A
socfb-Wisconsin87	6.4	**5.7**	3368.8	**46.9**	299.9	N/A	**92.7**	14643.0	N/A	**391.8**	N/A	N/A
socfb-wosn-friends	**7.9**	22.8	708.7	**55.8**	3025.6	N/A	**99.5**	N/A	N/A	**386.7**	N/A	N/A

Table 6: Search tree sizes in thousands for three algorithms on Facebook instances with cut-off time of 10800s

$k = 1$, but KDBB is slightly better than $\text{KDBB}_{\text{color}}$ on the average runtime. As k increases, our upper bound shows its good effectiveness. It achieves far better performance than $\text{KDBB}_{\text{color}}$ when $k > 1$ because both measurements are better. The difference of the number of solved instances is enlarged with k growing, where we can see our approach can solve almost all instances when $k = 10$ whereas the algorithm with the c-color bound can only solve 19 instances, less than $1/5$ of KDBB. Therefore, it is clear that our upper bound is effective, and particularly good at solving instances with large values of k.

	KDBB		$\text{KDBB}_{\text{color}}$	
group	#solved	time(s)	#solved	time(s)
k = 1	110	561.21	110	572.81
k = 3	110	687.78	97	2337.19
k = 5	108	926.60	79	4677.43
k = 10	109	1239.92	19	9434.41

Table 7: Statistic results of KDBB with two upper bounds

Conclusion

As a generalization of the clique, the k-defective clique is a useful tool for analyzing complex networks. We propose a branch-and-bound algorithm for the maximum k-defective clique problem. The main contribution is that our algorithm adopts several newly proposed upper bounds for graph reduction. Different from existing methods that remove vertices from graphs, we try to delete both useless vertices and edges. Also, we propose an effective upper bound for cutting branches based on counting missing edges between fixed vertices and unfixed ones. We perform extensive experiments on large-scale graphs from social networks and other real-world applications. We analyze the effectiveness and efficiency of our algorithm, showing that our algorithm is far better than existing exact algorithms. It is also worth to mention that our algorithm still performs well on solving instances with $k > 5$, but other algorithms fail to solve most of them. In the future work, enumerating maximal k-defective cliques is an interesting topic.

Acknowledgments

We would like to thank anonymous reviewers for useful suggestions. The work in this paper was supported by the National Natural Science Foundation of China (Nos. 61972063, 61976050 and 61806082) and the Fundamental Research Funds for the Central Universities, JLU (No. 93K172021K07).

References

Almeida, M. T.; and de Carvalho, F. D. 2014. Two-phase heuristics for the k-club problem. *Comput. Oper. Res.*, 52: 94–104.

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

Bomze, I. M.; Rinaldi, F.; and Zeffiro, D. 2021. Fast cluster detection in networks by first-order optimization. *arXiv preprint arXiv:2103.15907*.

Brunato, M.; Hoos, H. H.; and Battiti, R. 2007. On effectively finding maximal quasi-cliques in graphs. In Maniezzo, V.; Battiti, R.; and Watson, J., eds., *Learning and Intelligent Optimization, Second International Conference, Trento, Italy, December 8-12, 2007*, volume 5313 of *Lecture Notes in Computer Science*, 41–55.

Cai, S. 2015. Balance between complexity and quality: local search for minimum vertex cover in massive graphs. In Yang, Q.; and Wooldridge, M. J., eds., *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, July 25-31, 2015*, 747–753.

Cai, S.; and Lin, J. 2016. Fast solving maximum weight clique problem in massive graphs. In Kambhampati, S., ed., *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York, NY, USA, 9-15 July 2016*, 568–574.

Chen, X.; Zhou, Y.; Hao, J.; and Xiao, M. 2021. Computing maximum k-defective cliques in massive graphs. *Comput. Oper. Res.*, 127: 105131.

Conte, A.; Firmani, D.; Mordente, C.; Patrignani, M.; and Torlone, R. 2017. Fast enumeration of large k-plexes. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, 115–124.

Gschwind, T.; Irnich, S.; Furini, F.; and Calvo, R. W. 2020. A branch-and-price framework for decomposing graphs into relaxed cliques. *INFORMS Journal on Computing*.

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

Jain, S.; and Seshadhri, C. 2020. Provably and efficiently approximating near-cliques using the Turán shadow: PEANUTS. In Huang, Y.; King, I.; Liu, T.; and van Steen, M., eds., *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, 1966–1976. ACM / IW3C2.

Jiang, H.; Li, C.; and Manyà, F. 2017. An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, 830–838.

Jiang, H.; Zhu, D.; Xie, Z.; Yao, S.; and Fu, Z. 2021. A new upper bound based on vertex partitioning for the maximum k-plex problem. In Zhou, Z., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, Virtual Event / Montreal, Canada, 19-27 August 2021*, 1689–1696.

Lin, J.; Cai, S.; Luo, C.; and Su, K. 2017. A reduction based method for coloring very large graphs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, Melbourne, Australia, August 19-25, 2017*, 517–523.

Marinelli, F.; Pizzuti, A.; and Rossi, F. 2021. LP-based dual bounds for the maximum quasi-clique problem. *Discret. Appl. Math.*, 296: 118–140.

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

Rossi, R. A.; and Ahmed, N. K. 2014. Coloring large complex networks. *Social Network Analysis and Mining*, 4(1): 228.

Rossi, R. A.; and Ahmed, N. K. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 4292–4293.

Rossi, R. A.; Gleich, D. F.; Gebremedhin, A. H.; and Patwary, M. M. A. 2014. Fast maximum clique algorithms for large graphs. In *23rd International World Wide Web Conference, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, 365–366.

Sanei-Mehri, S.; Das, A.; Hashemi, H.; and Tirthapura, S. 2021. Mining largest maximal quasi-cliques. *ACM Trans. Knowl. Discov. Data*, 15(5): 81:1–81:21.

Sherali, H. D.; and Smith, J. C. 2006. A polyhedral study of the generalized vertex packing problem. *Math. Program.*, 107(3): 367–390.

Sherali, H. D.; Smith, J. C.; and Trani, A. A. 2002. An airspace planning model for selecting flight-plans under workload, safety, and equity considerations. *Transp. Sci.*, 36(4): 378–397.

Shirokikh, O. A. 2013. *Degree-based clique relaxations: theoretical bounds, computational issues, and applications*. Ph.D. thesis, University of Florida.

Stozhkov, V.; Buchanan, A.; Butenko, S.; and Boginski, V. 2020. Continuous cubic formulations for cluster detection problems in networks. *Mathematical Programming*, 1–29.

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

Verfaillie, G.; Lemaître, M.; and Schiex, T. 1996. Russian doll search for solving constraint optimization problems. In Clancey, W. J.; and Weld, D. S., eds., *Proceedings of*

the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, Portland, Oregon, USA, August 4-8, 1996, Volume 1, 181–187.

Xiao, M.; Lin, W.; Dai, Y.; and Zeng, Y. 2017. A fast algorithm to compute maximum k-plexes in social network analysis. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, 919–925.

Yu, H.; Paccanaro, A.; Trifonov, V.; and Gerstein, M. 2006. Predicting interactions in protein networks by completing defective cliques. *Bioinform.*, 22(7): 823–829.

Zhou, Y.; Hu, S.; Xiao, M.; and Fu, Z. 2021. Improving maximum k-plex solver via second-order reduction and graph color bounding. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, Virtual Event, February 2-9, 2021*, 12453–12460.