Efficient Maximum Defective Clique Search on Real-world Networks

ABSTRACT

....

1 INTRODUCTION

.

2 PROBLEM DEFINITION

Consider an undirected and unweighted graph $G = (V, E)$, where V and E represent the sets of vertices and edges of the graph G, respectively. Let $n = |V|$ and $m = |E|$ denote the number of vertices and edges in G, respectively. For a vertex v of G, we define $N_v(G)$ as the set of neighbors of v in G, i.e., $N_v(G) = \{u \in V | (u, v) \in E\}$. The degree of v in G is the cardinality of $N_v(G)$, denoted by $d_v(G) = |N_v(G)|$. Given a vertex subset S of G, we let $G(S) = (S, E_S)$ be the subgraph of G induced by the subset S, where $E_S = \{(u, v) \in E | u \in S, v \in S\}$. In accordance with [10], the k-defective clique of G is defined as follows.

Definition 1 (k-defective clique). Given a graph G and a non-negative integer k, the subgraph $G(S)$ induced by the vertex set $S \subseteq V$ is a k-defective clique if there exists at least $\binom{|S|}{2} - k$ edges in $G(S)$.

For simplicity, in the rest of this paper, we directly refer the set S as the k-defective clique of G. A k-defective clique S of G is considered maximal if there does not exist any other k-defective clique S' of G such that $S \subset S'$. Furthermore, a k-defective clique S of G is designated as maximum if its size is largest among all maximal k-defective clique of G, where the size of k-defective clique S is defined as the number of vertices it contains. Then, two useful properties of the k-defective clique are described below, which are very helpful in designing the algorithms.

Property 1 (Hereditary [1]). Given a k-defective clique S of G, every subset of S is also a k-defective clique of G.

The hereditary property (Property 1) of a k-defective clique simplifies the maximality check process. Specifically, if a k-defective clique S of G is not the maximal, it implies the existence of a vertex in $V \setminus S$ that can form a larger k-defective clique when combined with S. Thus, this property forms the foundation for the design of our algorithms.

Property 2 (Small Diameter [2]). Given a k-defective clique S of G, the diameter of $G(S)$ is no larger than 2 if $|S| \geq k + 2$.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD '24, June 9–15, 2024, Santiago, Chile

© 2024 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Algorithm 1: GraphColoring(G, τ)

Input: The graph $G = (V, E)$
Output: The color number of each vertex in V

Compute the degeneracy ordering of vertices in G; $\omega \leftarrow 0$;
for *each* $v \in V$ *in reverse order with the degeneracy ordering* **do**
    color$(v) \leftarrow 0$;
    while $\exists u \in N_v(G)$ *with color*(v) = *color*(u) **do**
        color$(v) \leftarrow$ color$(v) + 1$;
    `/* Recoloring */`
    if *color*$(v) \geq \tau$ **then**
        for $c = 0$ *to* $\tau - 1$ **do**
            if $|\{u \in N_v^+(G) | color(u) = c\}| = 1$ **then**
                Let u be the vertex in $N_v^+(G)$ with color$(u) = c$;
                if $\exists c' \in [0, \tau - 1]$ *s.t.* $\nexists w \in N_u(G)$ *with color*$(w) = c'$ **then**
                    color$(u) \leftarrow c'$; color$(v) \leftarrow c$;
                    break;
    $\omega \leftarrow \max\{\omega, \text{color}(v)\}$
return ω;

Property 2 not only implies the internal density-connected nature (having a diameter of two with a size no less than $k + 2$) of the k-defective clique but also provides an acceleration for enumerating relatively-large maximal k-defective cliques [**?**]. However, the problem of enumerating maximal k-defective clique often suffers from excessively long computational times, as there can be an exponential number of maximal k-defective cliques compared to the number of vertices. To address this challenge, in this paper, we aim to the problem of finding a maximum k-defective clique of G, and the formal problem definition is shown below.

Problem definition. Given a graph G and a non-negative integer k, the goal of this paper is to compute the maximum k-defective clique of G, i.e., finding a k-defective clique whose size is largest among all maximal k-defective clique of G.

2.1 Existing Solutions

As shown in [7, 9], the problem of finding the maximum k-defective clique of a given graph G is NP-complete, thus there is no algorithm with polynomial-time to solve this problem. To our knowledge, there are several solutions have been developed to address the problem [1, 4, 5, 7], which mainly consist of two categories.

Russian doll search based algorithms. The first algorithm for finding the maximum k-defective clique is developed by Trukhanov et al. [7], which is based on a Russian doll search technique [8]. The key idea of such an algorithm is summarized as follows. Let $\{v_1, v_2, ..., v_n\}$ be a total ordering for the vertices in V of G. The problem of finding the maximum k-defective clique from a graph G can be divided into a series of n nested subproblems. Each subproblem represents finding the maximum k-defective clique that includes v_i in the subgraph $G(\{v_i, v_{i+1}, ..., v_n\})$, where $v_i \in V$. Then, the algorithm starts from $i = n$ and iterates down to $i = 1$, processing each subproblem along the way. The final maximum solution is obtained when $i = 1$. In addition, the algorithm employs the branch-and-bound technique, which can be accelerated using the pre-knowledge when solving the i-th subproblem. Specifically, based on the upper bound on the size of the maximum k-defective clique that has been detected by j-th subproblem (where $j > i$), it is possible to determine whether a larger k-defective clique is also likely to contain the vertex v_j. Recently, such the Russian doll search algorithm has also been improved by Gschwinda et al. [5] based on some auxiliary information prior computed.

Branch-and-bound based algorithms. In order to enhance the efficiency of finding the maximum k-defective clique in a graph G, several new enumeration algorithms have been developed [1, 4] based on a branch-and-bound technique [6]. Notably, Chen et al. [1] propose a new branching rule and employ some reduction techniques to improve the search for the maximum k-defective clique. This proposed branching rule prioritizes the vertices that have non-neighbors within the current k-defective clique S to expand S, thus ensuring a maximum of $k + 1$ subbranches in each recursive call. Furthermore, the authors establish that the worst-case time complexity of this technique for finding maximum k-defective clique is bounded by $O(P(n)\gamma_k^n)$, where $P(n)$ is a polynomial function related to n and γ_k is a real-number less than 2. To further reduce unnecessary computations, an improved branch-and-bound enumeration algorithm is presented in [4] based on some newly developed branch pruning techniques. To our knowledge, this algorithm represents the current state-of-the-art for solving the problem of finding the maximum k-defective cliques.

Nevertheless, the these existing solutions still suffer from significant computational time when applied to real-world graphs. This issue arises primarily due to the insufficient tightening of bounding techniques in these algorithms, which leads to a proliferation of unnecessary computations during the branch-and-bound procedure. Additionally, the efficiency of the employed branching rules in swiftly identifying the maximum k-defective clique is also limited. Furthermore, it is noteworthy that the worst-case time complexity of most existing solutions remains bounded by $O(P(n)2^n)$, and only one approach [1] achieves a worst-case time complexity of $O(P(n)\gamma_k^n)$, where $\gamma_k < 2$. However, even for small values of k, the value of γ_k is nearly equal to 2. For instance, as reported in [1], when $k = 2$ and 3, the corresponding values of γ_k are 1.996 and 1.999, respectively, which are unacceptably high for the problem of finding the maximum k-defective clique in graph G.

Hence, there is a pressing need to develop more efficient algorithms to finding the maximum k-defective clique of real-world graphs. In the subsequent sections, we first introduce some new bounding techniques and subsequently present our approaches that enable the efficient discovery of the maximum k-defective clique.

3 BOUNDING TECHNIQUES

Let κ and $\kappa(C)$ denote the sizes of maximum k-defective clique in graph G and the subgraph $G(C)$, respectively. In this section, we start to explore both typical and novel upper bounds for κ and $\kappa(C)$, which play crucial role in accelerating the computations related to the maximum k-defective clique.

Degree-based upper bound. The first upper bound is simply derived from the degree information of vertices in G, and the detailed result is presented the following lemma.

Lemma 1. For a given graph G, the size of the maximum k-defective clique of G containing a vertex $v \in V$ is no larger than $d_v(G) + k + 1$. Consequently, we can establish $\kappa \leq \max_{v\in V} d_v(G) + k + 1$.

Proof. This lemma is clearly established based on the definition of k-defective clique. □

Core-based upper bound. Next, we introduce a tighter upper bound for κ and $\kappa(C)$ based on a well-established concept of k-core [?]. The formal definition of k-core is provided as follows.

Definition 2. Given a graph G, the subgraph $G(C)$ of G induced by the vertex set C is a k-core of G if $d_v(C) \geq k$ for every v in C.

Let C_k represent k-core subgraph of G. The core number of a vertex v in G, denoted by $core_v(G)$, is defined as the maximum value of k such that v belongs to the k-core subgraph C_k of G, i.e., $core_v(G) = \max\{k \mid v \in C_k\}$. Based on this concept, we can derive a core number based upper bound, as shown below.

Lemma 2. For a given graph G, the size of the maximum k-defective clique in G containing a vertex $v \in V$ is bounded by $core_v(G) + k + 1$. Consequently, we can establish $\kappa \leq \max_{v\in V} core_v(G) + k + 1$.

Proof. It is evident that any k-defective clique S of G is also a $(|S| - k - 1)$-core subgraph of G, as per the definitions provided in Definition 1 and Definition 2. Consequently, if a k-defective clique S in G includes a vertex v, it follows that the size of such a k-defective clique is bounded by $core_v(G) + k + 1$. Thus, this lemma is established. □

To compute the core number for each vertex in a given graph G, the traditional peeling technique described in [?] can be employed. This technique follows an iterative process where the vertices are sequentially removed from the remaining subgraph based on their degree, in a non-decreasing order starting from $k = 0$ up to $k = n$. The core number of a vertex is assigned to the current value of k at the time of its removal. The total time-consuming for this technique is bounded by $O(m)$, indicating its high efficiency in generating the core number based upper bound.

Color-based upper bound. Furthermore, we present a method to refine the upper bound for κ and $\kappa(C)$ using a graph coloring technique. Here we begin by providing the fundamental definition of graph coloring below.

Definition 3. Given a graph G, the graph coloring is to assign a color number for each vertex v of G, denoted by $color_v(G)$, such that any two adjacent vertices have different colors. Formally, for every $(u, v) \in E$, it should hold that $color_v(G) \neq color_u(G)$.

Denote by ω and $\omega(C)$ the number of distinct colors in G and the subgraph $G(C)$ of G induced by C, respectively. Based on the concept of graph coloring, we can establish the following upper bound for κ and $\kappa(C)$.

Lemma 3. Given a coloring of the graph G and a subset C within G, the size of the maximum k-defective clique in G and $G(C)$ can be bounded by $\omega + k$ and $\omega(C) + k$, respectively.

Proof. Given a k-defective clique S in G, we can partition it into two subsets, namely S_1 and S_2, satisfying $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S$. We assume that every vertex in S_1 has a different color than the other vertices in S_1, while every vertex in S_2 shares the same color with at least one vertex in S_1. Formally, for each $v \in S_1$ (resp. $u \in S_2$), it is true that $\{w \in S_1 \mid color_v(G) = color_w(G)\} = \emptyset$ (resp. $\{w \in S_1 \mid color_u(G) = color_w(G)\} \neq \emptyset$). Based on the definition of graph coloring, it is evident that if two vertices u and v in G have the same color (i.e., $color_v(G) = color_u(G)$), there exists no edge connecting them (i.e., $(u, v) \notin E$). Consequently, we can deduce that $|S_2| \leq k$, as having more than k vertices in S_2 would violate the definition of k-defective clique. Moreover, the number of distinct colors in G is denoted as ω, it follows that $|S_1| \leq \omega$. Thus, the size of the maximum k-defective clique in G is bounded by $\omega + k$, establishing this lemma. □

Lemma 3 highlights the importance of finding a smaller value for ω in order to achieve a better upper bound. However, it is worth noting that determining the smallest value of ω for graph coloring is a computationally challenging problem, known to be NP-hard [?]. Thus, many heuristic approaches have been extensively investigated to address graph coloring [?]. In this paper, we adopt a widely used degeneracy ordering to color the graphs. Specifically, the degeneracy ordering [?] is defined as follows.

Definition 4. Given a graph G with the vertex set V, the degeneracy ordering is a permutation $(v_1, v_2, ..., v_4)$ of vertices in V such that for each vertex v_i, its degree is smallest in the subgraph of G induced by $\{v_i, v_{i+1}, ..., v_n\}$.

The degeneracy ordering, similar to the method for computing the k-core of a graph, can be acquired using the peeling technique [?]. In this method, the vertex removal ordering aligns with the degeneracy ordering, and it can be executed with a time complexity of $O(m)$. Then, we can systematically assign colors to each vertex v_i of graph G in a reverse order of the degeneracy ordering, commencing from $i = n$ and concluding at $i = 1$. Consequently, the upper bound based on graph coloring can be computed efficiently in $O(m)$ time.

Denote by δ the maximum core number of G, i.e., δ is the maximum value of k such that there exist a non-empty k-core in G. When combining with Lemma 1-3, we then have the following relations.

Lemma 4. Given a graph G, let κ be the size of the maximum k-defective clique of G. We can derive that $\kappa \leq \omega + k \leq \delta + k + 1 \leq d_{max} + k + 1$, where d_{max} is the maximum degree of vertices in G.

Proof. It is evident that $d_{\max} \geq \delta$ and $d_{\max} \geq \omega - 1$ are clearly established. Thus, here we only prove the correctness of $\omega \leq \delta + 1$. Assume that the vertex ordering $\{v_1, v_2, ..., v_n\}$ corresponds to the degeneracy ordering. Based on this ordering, we can obtain that for each vertex v_i, the degree of v_i is the smallest among all vertices in the subgraph $G(v_i, v_{i+1}, ..., v_n)$ induced by the set $v_i, v_{i+1}, ..., v_n$, where $i \in [1, n]$. Denote by $d^+_{v_i}$ the degree of v_i in $G(\{v_i, v_{i+1}, ..., v_n\})$. It can be seen that the subgraph $G(\{v_i, v_{i+1}, ..., v_n\})$ also forms a $d^+_{v_i}$-core, implying that $d^+_{v_i} \leq \delta$. Moreover, once all vertices in $\{v_{i+1}, v_{i+2}, ..., v_n\}$ have been colored, when it comes to coloring the vertex v_i, we can see that the color number assigned to v_i is at most $d^+_{v_i} + 1$. Consequently, we establish $\omega \leq \delta + 1$. Hence, the lemma is proven. □

The following example illustrates the above mentioned upper bounds.

Example 1. ...

Advanced color-based upper bound. We observe that the proposed technique for obtaining the upper bound is still not sufficiently tight. For example, let us consider a graph G depicted in Fig. ??, which has been colored using the degeneracy ordering. Given a subset of vertices $C = *******$ in G, it becomes apparent that the number of distinct colors utilized in the subgraph $G(C)$ is **. According to Lemma 3, we deduce that the upper bound for $\kappa(C)$ is *** when $k = 3$, which exceeds the size of the vertex set C. Furthermore, it is evident that the subgraph $G(C)$ itself is not a k-defective clique of G when $k = **$, implying that the upper bound for $\kappa(C)$ can be tightened to 5. To address this concern, we next develop a new color-based upper bound.

Given a vertex subset S of G, let $\kappa(S, C)$ be the size of the maximum k-defective clique in $G(S \cup C)$ that includes all vertices in S. We define $\overline{d}_v(S)$ as the number of non-neighbors of vertex v in S, given by $\overline{d}_v(S) = |S \setminus N_v(S)|$. Additionally, we define $c_v(S)$ as the number of other vertices in S that share the same color with vertex v, denoted as $c_v(S) = |\{u \in S \setminus \{v\} | color_v(G) = color_u(G)\}|$. With these definitions in place, we state the following lemma.

Lemma 5. Consider a graph G and a non-maximal k-defective clique S of G. If there exists a vertex set $D \in V \setminus S$ that can form a larger k-defective clique with S, then for each vertex $v \in D$, there are at least $\overline{d}_v(S) + c_v(D)$ non-neighbors of v in $G(S \cup D)$.

Proof. This lemma is clearly established since each vertex in $D \setminus \{v\}$ that has the same color as v is not the neighbor of v. □

Based on Lemma 5, we then have the following upper bound for $\kappa(S, C)$.

Lemma 6. Denote by $\overline{d}(S)$ the total number of missing edges in $G(S)$, i.e., $\overline{d}(S) = \frac{1}{2}\sum_{v \in S}(|S| - d_v(S) - 1)$. Given two sets S and C, let D be the largest subset of C satisfying $\sum_{v \in D}(\overline{d}_v(S) + \frac{1}{2}c_v(D))\}) \leq k - \overline{d}(S)$. When computing the maximum k-defective clique containing S in the subgraph $G(S \cup C)$, we can state that $\kappa(S, C) \leq |S| + |D|$.

Proof. Given the subset D of C, we observe that there are at least $\frac{1}{2}\sum_{v \in D} c_v(D)$ missing edges in $G(D)$, as any pair of vertices with the same color in D must not have an edge between them. Moreover, each vertex v in D has $\overline{d}_v(S)$ non-neighbors in S. Consequently, when the set D is incorporated into the current k-deficient clique S, there will be a minimum of $\sum_{v \in D}(\overline{d}_v(S) + \frac{1}{2}c_v(D))\})$ additional missing edges. Given that D is the largest subset of C satisfying $\sum_{v \in D}(\overline{d}_v(S) + \frac{1}{2}c_v(D))\}) \leq k - \overline{d}(S)$, we can conclude that the size of maximum k-defective clique containing S in the subgraph $G(S \cup C)$ is at most $|S| + |D|$. This completes the proof of the Lemma. □

Example 2. ...

Algorithm 2: Upperbound(S, C, k)

```
/* Supposing that each vertex in S∪C is colored based
   on the degeneracy ordering                          */
1 D ← ∅; s ← the missing edges in G(S);
2 while C ≠ ∅ do
3     v ← a vertex in C with minimum d̄_v(S) + c_v(D);
4     if s + d̄_v(S) + c_v(D) > k then break;
5     s ← s + d̄_v(S) + c_v(D);
6     D ← D ∪ {v}; C ← C \ {v};
7 return |S| + |D|;
```

Based on Lemma 6, we can derive an algorithm, as shown in Algorithm 2, to compute the upper bound of $\kappa(S, C)$. This algorithm employs a heuristic approach to determine the largest subset D of C. Initially, the algorithm sets D as an empty set (line 1). Then, it iteratively selects a vertex v in C with the smallest value of $\overline{d}_v(S) + c_v(D)$ and adds it to the current subset D (lines 2-6). Once v is selected, the missing edges in $G(S \cup D)$ increase by $\overline{d}_v(S) + c_v(D)$, and v is moved from C to D (lines 5-6). The algorithm terminates and outputs $|S| + |D|$ as the upper bound of $\kappa(S, C)$ when either C becomes empty or the missing edges in $G(S \cup D)$ would violate the definition of a k-defective clique (lines 2 and 4). The following Theorem presents the correctness of Algorithm 2.

Theorem 3.1. *Algorithm 2 correctly computes the upper bound of $\kappa(S, C)$.*

Proof. On the contrary, we assume that there exist a subset D' of C with $|D'| > |D|$ that satisfies $\sum_{v \in D'} (\overline{d}_v(S) + \frac{1}{2} c_v(D')\}) \leq k - \overline{d}(S)$. Denote by $D_1 = D' \setminus D$. It is easy to see that there must exist two vertices $v \in D$ and $u \in D_1$ satisfying $\overline{d}_v(S) + c_v(D \setminus \{v\}) > \overline{d}_u(S) + c_u(D \setminus \{v\})$, or D is the largest result. If $color_v(G) = color_u(G)$, we obtain that $\overline{d}_v(S) > \overline{d}_u(S)$ and the vertex u must contain in D if $v \in D$ according to our algorithm. If $color_v(G) \neq color_u(G)$, we have $\overline{d}_v(S) + c_v(D) = \overline{d}_v(S) + c_v(D \cup \{u\})$ and the vertex u will be also prior to push into D compared to u. However, we note that $u \notin D$, which is contradictory. Thus, we proved this lemma. ◻

Theorem 3.2. *The time complexity of Algorithm 2 is bounded by $O(kn + \overline{m})$, where $\overline{m}$ is the number of missing edges in $G(C)$.*

Proof. Initially, Algorithm 2 involves sorting each vertex v in set C based on the size of $\overline{d}_v(S)$. This sorting operation can be accomplished in $O(kn)$ time using a bin sort. Additionally, when a vertex v from C is added to set D, any vertex u in the set $C \setminus D$ that shares the same color as v will have its $c_u(D)$ value increased by 1. This operation takes at most $O(\overline{d}_v(C))$ time. Consequently, the total time of executing lines 2-6 in Algorithm 2 amounts to $O(\overline{m})$. Therefore, this theorem is proved. ◻

It is worth noting that through the preprocessing steps outlined in Lemma 1-3, the subgraph $G(S \cup C)$ typically exhibits a higher density in real-world graphs. This implies that the number of missing edges in $G(C)$ is relatively small. As a result, Algorithm 2

Table 1: Time complexity.

Algorithm	Time	$k =$	1	2	3	4	5
A et al. [?]	$O^*(\alpha_k^n)$	$\alpha_k =$	1.928	1.984	1.996	1.9990	1.9998
B et al. [?]	$O^*(\beta_k^n)$	$\beta_k =$	1.839	1.928	1.966	1.984	1.992
Others [?]	$O^*(2^n)$	–	–	–	–	–	–
Ours	$O^*(\gamma_k^n)$	$\gamma_k =$	1.466	1.755	1.889	1.948	1.975

demonstrates exceptional time efficiency when computing the upper bound of $\kappa(S, C)$, which is further supported by our practical experiments, as detailed in Section ??.

4 THE PROPOSED ALGORITHMS

In this section, we present a novel algorithm to detect the maximum k-defective clique in a given graph G. Our algorithm builds upon a widely used branch-and-bound technique [?], which revolves around the concept of dividing the problem into smaller sub-problems. Specifically, we define $I = (G, S, C, k)$ as an instance aimed at computing the maximum k-defective clique that contains set S within the subgraph $G(S \cup C)$ of G, where S and C are the current partial k-defective clique and the candidate set used to enlarge S, respectively. By selecting a branching vertex v from C, we split the instance I into two sub-instances: $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ and $I_2 = (G, S, C \setminus \{v\}, k)$. It can be seen that the optimal solution for instance I is precisely the larger solution obtained from either I_1 or I_2. Thus, to obtain the final solution for I, each sub-instance is recursively divided until the candidate set C becomes empty, and the overall outcome is then determined as the maximum solution obtained across all sub-instances.

However, the total number of sub-instances for the instance $I = (G, S, C, k)$ initialized with $S = \emptyset$ and $C = V$ is at most $O(2^n)$. Thus, it is much inefficient for detecting all possible sub-instances of I. To achieve a better performance, it is important to determine the unnecessary sub-instances that definitely cannot retrieve the maximum k-defective clique of G as much as possible. Below, we first introduce some new branching techniques and then present our enumeration algorithm.

4.1 Branch Reduction Rules

In this subsection, we further discuss some special cases to reduce the unnecessary branches during the detecting the maximum k-defective clique of G. Given by $I = (G, S, C, k)$ an instance to compute the maximum k-defective clique containing S in the subgraph $G(S \cup C)$, we note that the sub-instances of I can be further reduced if there exists a vertex in C that has at most three non-neighbors in $S \cup C$. Let v be the vertex in C with $\overline{d}_v(S \cup C) \leq 3$. We then have the following three cases.

(1) One non-neighbor reduction: $\overline{d}_v(S \cup C) = 1$. In this case, all other vertices in $S \cup C$ are the neighbors of v. Then, the maximum k-defective clique in $G(S \cup C)$ must contain v, and the following lemma can be derived.

Lemma 7. Given an instance $I = (G, S, C, k)$, if there exists a vertex v in C with $\overline{d}_v(S \cup C) = 1$, then the maximum k-defective clique for instance I must include in the sub-instance $I' = (G, S \cup \{v\}, C \setminus \{v\}, k)$.

(2) Two non-neighbors reduction: $\overline{d}_v(S \cup C) = 2$. Let u be the non-neighbor of v in $S \cup C$. It is easy to verify that the maximum k-defective clique in $G(S \cup C)$ contains at least one vertex among v and u. Let S_1^* be maximum k-defective clique that contains u. Then, we note that there always exists a k-defective clique S_2^* that satisfies $|S_2^*| \geq |S_1^*|$ and $v \in S_2^*$. Thus, there is no necessary to detect the maximum k-defective clique that excludes u for instance I if $\overline{d}_v(S \cup C) = 2$, and the following lemma is obtained.

Lemma 8. Given an instance $I = (G, S, C, k)$, if there exists a vertex v in C with $\overline{d}_v(S \cup C) = 2$, then the maximum k-defective clique for instance I must include in the sub-instance $I' = (G, S \cup \{v\}, C \setminus \{v\}, k)$.

Proof. Let u be the non-neighbor of v in $S \cup C$. Assuming S^* is a maximum k-defective clique of instance I, we can easily verify that S^* must include v if $u \notin S^*$. This is based on a fact that u is the only non-neighbor of v in $S \cup C$. Next, we establish the correctness of this lemma when $u \in S^*$. If $u \in C$, we observe that $S^* \setminus \{u\} \cup \{v\}$ also constitutes a maximum k-defective clique of instance I. On the other hand, if $u \in S$, we have $(S^* \setminus S) \subseteq N_v(G)$. Let w be a vertex in $S^* \setminus S$ with the minimum value of $d_w(S^*)$. If $d_w(S^*) = |S^*| - 1$, it follows that S^* represents a $(k-1)$-defective clique in $G(S \cup C)$, as each vertex in C can be used to expand S. Hence, S^* must contain v if $d_w(S^*) = |S^*| - 1$. If $d_w(S^*) \leq |S^*| - 2$, then $S^* \setminus \{w\} \cup \{v\}$ forms a maximum k-defective clique of instance I. By combining the aforementioned analysis, we conclude that the sub-instance $I' = (G, S \cup \{v\}, C \setminus \{v\}, k)$ is capable of retrieving a maximum k-defective clique with a size no less than $|S^*|$. This completes the proof of this lemma. □

Based on Lemma 7 and Lemma 8, we derive that for an instance $I = (G, S, C, k)$, if there exist a vertex v in C with $\overline{d}_v(S \cup C) \leq 2$, it only needs to consider the case of the maximum k-defective clique of I that contains v. Thus, it significantly reduces the unnecessary sub-branches that detect the maximum k-defective clique excluding v. The following example further demonstrates these results.

Example 3. ...

(3) Three non-neighbors reduction: $\overline{d}_v(S \cup C) = 3$. Let u and w be the two non-neighbors of v in $S \cup C$. The maximum k-defective clique in the instance $I = (G, S, C, k)$ must contain at least one vertex in set $\{v, u, w\}$. Based on Lemma 8, we can further obtain that if such a maximum k-defective clique contains only the vertex u or w, there must also exists maximum k-defective clique contains v. Thus, we can conclude that if the maximum k-defective clique exclude the vertex v, it must contains both u and w. Here further improves this result with the following lemma.

Lemma 9. Given an instance $I = (G, S, C, k)$, if there exists a vertex v in C satisfying $\overline{d}_v(S \cup C) = 3$ and $\overline{d}_v(S) \leq 1$, with u and w denoting the two non-neighbors of v in $S \cup C$, the following results hold.

- *For the case of $\overline{d}_v(S) = 0$, we derive that: (1) if $(u, w) \notin E$ or $\overline{d}_u(S) + \overline{d}_w(S) \geq 1$, the maximum k-defective clique for instance I must include in the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$; (2) otherwise, it either includes in the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ or the sub-instance $I_2 = (G, S \cup \{u, w\}, C \cap N_u(G) \cap N_w(G), k)$.*
- *For the case of $\overline{d}_v(S) = 1$, we derive that: (1) if $\overline{d}_u(S) \geq 1$ with $u \in C$, the maximum k-defective clique for instance I must include in the sub-instance $I_1 = (G, S \cup \{u\}, C \setminus \{u\}, k)$; (2) otherwise, it either includes in the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ or the sub-instance $I_3 = (G, S \cup \{u\}, C \cap N_u(G), k)$.*

Proof. Let S^* be the maximum k-defective clique of I. We first prove the correctness for the case of $\overline{d}_v(S) = 0$. Based on Lemma 8, if S^* contains only one vertex in $\{u, w\}$, then a k-defective clique with the size no less than $|S^*|$ can also be generated by the sub-instance I_1. Next, we only consider the case of $\{u, w\} \subset S^*$ when $\overline{d}_v(S) = 0$. It can be seen that if $\overline{d}_u(S) \geq 1$ (or $\overline{d}_w(S) \geq 1$) or $(u, w) \in E$, $S^* \setminus \{u\}$ (or $S^* \setminus \{w\}$) will be a $(k-1)$-defective clique of G, which can be expanded by the vertex v. This meas that the sub-instance I_1 can obtain a maximum k-defective clique with the size no less than $|S^*|$ if $(u, w) \notin E$ or $\overline{d}_u(S) + \overline{d}_w(S) \geq 1$ for the case of $\overline{d}_v(S) = 0$. Moreover, if S^* is obtained by the sub-instance $I' = (G, S \cup \{u, w\}, C \setminus \{u, w\}, k)$, only the common neighbors of u and w in C can be used to expand $S \cup \{u, w\}$. The reason is that if there exist a vertex v' in $C \setminus N_u(G)$ (or $C \setminus N_w(G)$) that belongs to S^*, it is easy to verify that $S^* \setminus \{u\}$ (or $S^* \setminus \{w\}$) is a $(k-1)$-defective clique of G, resulting in that $S^* \setminus \{u\} \cup \{v\}$ (or $S^* \setminus \{w\} \cup \{v\}$) is also a maximum k-defective clique, which can be obtained by the sub-instance I_1. Thus, we established the correctness for the case of $\overline{d}_v(S) = 0$. For the case of $\overline{d}_v(S) = 0$, we can make use of a similar method to prove that if $\overline{d}_u(S) \geq 1$ with $u \in C$, the maximum k-defective clique for instance I is included in the sub-instance $I_1 = (G, S \cup \{u\}, C \setminus \{u\}, k)$; (2) otherwise, it is either included in the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ or the sub-instance $I_3 = (G, S \cup \{u\}, C \cap N_u(G), k)$. That is the complete proof of this lemma. □

The following example demonstrates the idea of Lemma 9.

Example 4. ...

4.2 New Pivot-Based Techniques

Before presenting our techniques, here we first introduce a pivot-based solution originally developed for enumerating maximal k-defective clique of G [**?**]. The idea of this pivot-based technique is that given an instance $I = (G, S, C, k)$ to enumerate all maximal k-defective clique that contains S in G, if there exists a vertex with $S \subseteq N_v(G)$, then all maximal k-defective clique that contains S in G either contains the vertex v or a non-neighbor vertex of v in C. Clearly, this is also suitable for solving the problem of detecting the maximum k-defective clique. However, we note that the restriction for the pivot vertex is so restrictive such that there still exist so much unnecessary computations when using this pivot-based technique to detect the maximum k-defective clique of G. For instance, consider a graph G shown in Fig. **??**, we assume that the instance $I = (G, S, C, k)$ is initialized with $S = \{....\}$ and $C = \{....\}$. It is easy to see that there is no vertex v in C satisfying that $S \subseteq N_v(G)$. Thus, this instance I cannot be pruned with the pivot-based technique developed in [**?**], which leads to a large number of redundant computations. To fix this issue, we present a new pivot-base technique to detect the maximum k-defective clique of G, which is present below.

Theorem 4.1 (New pivoting). *Given an instance $I = (G, S, C, k)$ to find the maximum k-defective clique that contains S in $G(S \cup C)$, let v, denoted by the pivot vertex, be a vertex in C with $\overline{d}_v(S) \leq 1$, then the maximum k-defective clique for instance I either contains v or a vertex in $C \setminus \{v\} \setminus N_v(G)$.*

Proof. When $\overline{d}_v(S) = 0$, this theorem is clearly established based on the result shown in [2]. Thus, we only focus on the case of $\overline{d}_v(S) = 1$. Since $S \cup \{v\}$ is a k-defective clique, we can obtain that the number of missing edges in $G(S)$ is less than k. Denote by $C_1 = C \cap N_v(G)$. It is easy to verify that this theorem ignores the maximum k-defective clique contained in $G(S \cup C_1)$. Assume that $S \cup D$ is the maximum k-defective clique in $G(S \cup C_1)$, where $D \subseteq C_1$. We next show that the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ of I can also generate a k-defective clique with the size no less than $|S \cup D|$. If each vertex u in D satisfies that $\overline{d}_v(S \cup D) = 1$, then the number of missing edges in $G(S \cup D)$ is also less than k. This means that a larger k-defective clique $S \cup D \cup \{v\}$ in $G(S \cup C)$ can be obtained by the sub-instance I_1. Otherwise, if there exists a vertex u in D with $\overline{d}_u(S \cup D) \geq 2$, based on the condition of $\overline{d}_v(S \cup D) = 2$, we note that $S \cup D \setminus \{u\} \cup \{v\}$ is also a k-defective clique of $G(S \cup C)$, which can be detected by I_1. Thus, the maximum k-defective clique returned by the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ of I is no less than the maximum k-defective clique in $G(S \cup C_1)$. This is the complete proof of the theorem. □

Although Theorem 4.1 is efficient in reducing redundant sub-branches that definitely cannot generate the maximum k-defective clique, we note that in instance $I = (G, S, C, k)$, when expanding S with vertices in $C \setminus N_v(G)$, there still may exist unnecessary computations, where v is a pivot vertex selected from C. For example, suppose that S^* is the maximum k-defective clique of the instance I. If $|S^* \setminus \{v\} \setminus N_v(G)| \geq 2$, it can be seen that such a maximum k-defective clique S^* can be detected by either $I' = (G, S \cup \{u\}, C \setminus \{u\}, k)$ or $I'' = (G, S \cup \{w\}, C \setminus \{w\}, k)$, where u and w are the two vertices in $S^* \setminus \{v\} \setminus N_v(G)$ that will be used to expand S based on the pivot-based technique shown in Theorem 4.1. Thus, this results in the redundant computations. To remedy this issue, we further improve this pivot-base technique, which is shown below.

Theorem 4.2. *Given an instance $I = (G, S, C, k)$, let v be the pivot vertex in C with $\overline{d}_v(S) \leq 1$. Denote by $P = C \setminus \{v\} \setminus N_v(G)$. We then have the following two cases:*

(1) If $\overline{d}_v(S) = 0$, the maximum k-defective clique for instance I either contains v or an edge in $G(P)$.
(2) If $\overline{d}_v(S) = 1$, the maximum k-defective clique for instance I either contains a vertex in $\{v\} \cup P_1$ or an edge in $G(P_2)$, where $P_1 = \{u \in P | \overline{d}_u(S) = 0\}$ and $P_2 = P \setminus P_1$.

Proof. Denote by S^* the maximum k-defective clique that contains S in $G(S \cup C)$, and let $D = S^* \cap P$ be the subset of vertices in $S^* \cap C$ that are also non-neighbors of v. We then show the correctness of case (1). Suppose that, on the contrary, the set D is the independent set if $v \notin S^*$. We can easily verify that $D \neq \emptyset$, or $S^* \cup \{v\}$ will be a larger k-defective clique of G based on a fact that $\overline{d}(S) = 0$. Given a random vertex u in D, we can obtain that $S^* \setminus \{u\}$ is a $(k + 1 - |D|)$-defective clique in $G(S \cup C)$, since D is an independent set. Moreover, based on condition of $(S^* \setminus D) \subseteq N_v(G)$, we can derive that $S^* \setminus \{u\} \cup \{v\}$ must be also a k-defective clique in $G(S \cup C)$, which can be obtained by the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$. Thus, if $v \notin S^*$, there must exist at least one edge in $G(D)$ for case (1). For case (2), we have already known that S^* must contains a vertex in $\{v\} \cup P$ according to Theorem 4.1. Then, it only needs to prove that there must exist at least one edge in $G(D)$ when $S^* \cap (\{v\} \cup P_1) = \emptyset$. As an analytical method similar to the previous case, we can note that $S^* \setminus \{u\} \cup \{v\}$ is also a k-defective clique in $G(S \cup C)$, where u is a random vertex in D. As a result, this theorem is established. □

The following example further illustrates the idea of the proposed Theorem 4.1 and Theorem 4.1.

Example 5. ...

4.3 The Algorithm Implementation

Equipping with the proposed branch reduction techniques and the pivoting techniques, we then can develop a new branching rule for detecting the maximum k-defective clique of a given graph G.

Branching rule. Consider an instance $I = (G, S, C, k)$ to detect the maximum k-defective clique containing S in $G(S \cup C)$, where C is the set of vertices used to expand the current k-defective clique S. The maximum solution for instance I can be detected by the following branching rule.

- If there has a vertex $v \in C$ with $\overline{d}_v(S \cup C) \leq 3$ and $\overline{d}_v(S) \leq 1$, we make use of the proposed branch reduction rules (in Sec. 4.1) to find the maximum solution for the instance I.
- Else if $|C \setminus N(S)| + \overline{d}(S) \geq k \geq 1$, we select a vertex $v \in C$ with the maximum $\overline{d}_v(S)$ to split the instance I into two sub-instances $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ and $I_2 = (G, S, C \setminus \{v\}, k)$.
- Otherwise, we make use of the proposed pivoting technique in Theorem 4.2 to branch the instance I.

Implementation details. Armed with the proposed branching rule, we develop a new algorithm to detect the maximum k-defective clique of G, which is shown in Algorithm 3.

Algorithm 3 begins by invoking the *Branch*(S, C) procedure (line 1), where the parameters S and C are denoted by the current k-defective clique the candidate set used to expand S, respectively. This *Branch* procedure makes use of the proposed branching rules to find the maximum k-defective clique of G (lines 9-29). Specifically, if there exist such a vertex u in C that satisfies $\overline{d}_u(S \cup C) \leq 3$ and $\overline{d}_u(S) \leq 1$, it adopts the branch reduction rules (proposed in Sec. 4.1) to find the maximum k-defective clique that contains S (lines 9-10). If the branch reduction rule cannot be used, the procedure then finds whether the number of common neighbors of S in C is no larger than $|C| - k + \overline{d}(S)$ (line 11). If the condition holds, this procedure calls the branch-and-bound technique to detect the maximum k-defective clique that contains v and excludes v (line 14), respectively, where v is a vertex in C with the maximum value of $\overline{d}_v(S)$ among all vertices in $C \setminus N(S)$ (line 13). Otherwise, the maximum k-defective clique either contains a vertex in P_1 or an edge in $G(P_2)$ based on Theorem 4.2, where $P_1 = \{v\}$ (or $P_1 = \{v\} \cup \{u \in \overline{N}_v(C) | \overline{d}_v(S) = 0\}$ if $\overline{d}_v(S) = 1$) and $P_2 = \overline{N}_v(C) \setminus P_1$ when given a pivot vertex v in C. To make the size of $P_1 \cup P_2$ as

Algorithm 3: The branch and bound algorithm

Input: The graph $G = (V, E)$ and a parameter k
Output: The maximum k-defective clique S^* of G
1 $S^* \leftarrow \emptyset$;
2 $Branch(\emptyset, V)$;
3 **return** S^*;
4 **Function:** $Branch(S, C)$
5 **if** $C = \emptyset$ **then**
6   **if** $|S| > |S^*|$ **then** $S^* \leftarrow S$;
7   **return**;
8 **if** $\kappa(S, C) \leq |S^*|$ **then return**;
9 **if** $\exists u \in C$ *with* $\overline{d}_u(S) \leq 1$ *and* $\overline{d}_u(S \cup C) \leq 3$ **then**
10   Applying the branch reduction rules shown in Lemma 7-9;
11 **else if** $|C \setminus N(S)| + \overline{d}(S) \geq k \geq 1$ **then**
12   $v \leftarrow$ a vertex in $C \setminus N(S)$ with largest $\overline{d}_v(S)$;
13   $C' \leftarrow Update(S, C, v)$;
14   $Branch(S \cup \{v\}, C')$; $Branch(S, C \setminus \{v\})$;
15 **else**
16   $v \leftarrow$ a vertex in $\{u \in C | \overline{d}_u(S) \leq 1\}$ with largest $d_v(C)$;
17   $P_1 \leftarrow \{v\}$; $P_2 \leftarrow \overline{N}_u(C) \setminus P_1$;
18   **if** $\overline{d}_v(S) = 1$ **then**
19    $P_1 \leftarrow \{u \in \overline{N}_u(C) | \overline{d}_u(S) = 0\} \cup P_1$;
20    $P_2 \leftarrow \overline{N}_u(C) \setminus P_1$;
21   **foreach** $u \in P_1$ **do**
22    $C' \leftarrow Update(S, C, u)$;
23    $Branch(S \cup \{u\}, C')$; $C \leftarrow C \setminus \{u\}$;
24   **foreach** $u \in P_2$ **do**
25    $C' \leftarrow Update(S, C, u)$; $P'_2 \leftarrow C' \cap P_2$;
26   **foreach** $w \in P'_2$ *s.t.* $(u, w) \in E$ **do**
27     $C'' \leftarrow Update(S \cup \{u\}, C', w)$;
28     $Branch(S \cup \{u, w\}, C'')$; $C' \leftarrow C' \setminus \{w\}$;
29    $C \leftarrow C \setminus \{u\}$;

Algorithm 4: $Update(S, C, v)$

// Retrieve the vertices in C that can also be used to expand $S \cup \{v\}$
1 $C' \leftarrow \emptyset$; $s \leftarrow$ the number of missing edges in $G(S)$;
2 **for** $u \in C$, *s.t.* $u \neq v$ **do**
3 $\overline{d} \leftarrow s + \overline{d}_v(S) + \overline{d}_u(S)$;
4 **if** $\overline{d} \leq k$ **then**
5   **if** $u \in N_v(G)$ **then** $C' \leftarrow C' \cup \{u\}$;
6   **else if** $\overline{d} < k$ **then** $C' \leftarrow C' \cup \{u\}$;
7 **return** C';

small as possible, this procedure selects a vertex v in C with the maximum $\overline{d}_v(C)$ as the pivot vertex (line 16). Subsequently, this procedure iteratively selects the vertices in P_1 (lines 21-23) and the edges in $G(P_2)$ (lines 24-29) to expand the current k-defective clique. Note that if the current k-defective clique S is larger than all the k-defective cliques detected so far, S is refereed as the current maximum one (line 6). Finally, this procedure terminates whenever C is empty (lines 5-7) or there is no maximum k-defective clique of G in the subgraph $G(S \cup C)$ (line 8).

Note that when a vertex $v \in C$ is added into S, the current candidate set also needs to be updated so that all the remaining vertices in the newly candidate set can be used to expand $S \cup \{v\}$. To achieve this, we develop a procedure shown in Algorithm 4. Suppose that the vertex $v \in C$ is added into S. This procedure simply removes each vertex u in $C \setminus \{v\}$ that has more than $k - s - \overline{d}_v(S)$ non-neighbors in $S \cup \{v\}$ (lines 2-6), where s is the total number of missing edges in $G(S)$ (line 1). Then each remaining vertex can be used to expand the $S \cup \{v\}$ (line 7). It is easy to verify that the time complexity of this update algorithm is bounded by $O(n)$.

4.4 Complexity Analysis

We next show the time and space complexity of the proposed algorithm, which are presented below.

Theorem 4.3. *The time complexity of Algorithm 3 is bounded by $O(m\gamma_k^n)$, where γ_k is the maximum real root of $x^{k+3} - 2x^{k+2} + x^2 - x + 1 = 0$ if $k \geq 1$. Specifically, when $k = 1, 2$ and 3, the values of γ_k are 1.466, 1.755, and 1.889, respectively.*

Proof. Let $T(n)$ be the total number of leaves generated by the *branch*(S, C) procedure of Algorithm 3. Then, we can derive that the time complexity of Algorithm 3 is bounded by $O(mT(n))$, as each recursive call of *branch* consumes at most $O(m)$ time. Next, we analyze the size of $T(n)$ with the following three cases.

(1) If $\exists v \in C$ with $\overline{d}_v(S \cup C) \leq 3$ and $\overline{d}_v(S) \leq 1$, the algorithm makes use of the branch reduction rules shown in Sec. 4.1 to find the maximum k-defective clique of G. If $\overline{d}_v(S \cup C) \leq 2$, it is easily to obtain the following recurrence.
$$T(n) \leq T(n-1). \tag{1}$$
Otherwise, the branch reduction rule presented in Lemma 9 will be used. Let u and w be the two non-neighbors of v in $S \cup C$. If $\overline{d}_v(S) = 0$, the algorithm would invoke two sub-branches $Branch(S \cup \{v\}, C \setminus \{v\})$ and $Branch(S \cup \{u, w\}, C \cap N_u(G) \cap N_w(G))$ to find the maximum k-defective clique in the worst case. Based on the conditions of $\overline{d}_u(S) + \overline{d}_w(S) = 0$, $\overline{d}_u(S \cup C) \geq 3$ and $\overline{d}_w(S \cup C) \geq 3$, we can derive that $|C \cap N_u(G) \cap N_w(G)| \leq |C| - 4$, thus we have:
$$T(n) \leq T(n-1) + T(n-4). \tag{2}$$
If $\overline{d}_v(S) = 1$, the algorithm would invoke two sub-branches $Branch(S \cup \{v\}, C \setminus \{v\})$ and $Branch(S \cup \{u\}, C \cap N_u(G))$ to find the maximum k-defective clique in the worst case. Since $|C \cap N_u(G)| \leq |C| - 3$, we then have:
$$T(n) \leq T(n-1) + T(n-3). \tag{3}$$
Thus, Eq. (3) is the worst case recurrence for the branch reduction rules.

(2) If $|C \setminus N(S)| + \overline{d}(S) \geq k$, the algorithm selects a vertex v in $C \setminus N(S)$ to perform the branch-and-bound procedure. We then have:
$$T(n) \leq T(n-1) + T(n-1). \tag{4}$$
We note that if every vertex in $C \setminus \{v\}$ can also be used to expand $S \cup \{v\}$, then another vertex in $C \setminus N(S)$ is selected to perform

the branch-and-bound procedure in $branch(S \cup \{v\}, C \setminus \{v\})$. Similarly, this process is performed recursively until there are k missing edges in the current k-defective clique. If $\overline{d}_v(S) \geq 2$, we can obtain that at most $k - \overline{d}(S) - 1$ vertices in $C \setminus N(S)$ can be added into S. Initially, we have $\overline{d}(S) = 0$ and $|C \setminus N(S)| \geq k$, then the following recurrence can be obtained.

$$T(n) \leq \sum_{i=1}^{k} T(n-i) \quad \text{if } k \geq 2. \tag{5}$$

If $\overline{d}_v(S) \leq 1$, we can obtain that $\overline{d}_v(S \cup C) \geq 4$ based on the branch reduction technique. If there exists a non-neighbor of v in $C \setminus N(S)$, then the recurrence of Eq. (5) can be obtained accordingly. Otherwise, we can derive that $|C \setminus N(S \cup \{v\})| \geq k - \overline{d}(S) + 2$. Since at most $k - \overline{d}(S) - 1$ vertices in $C \setminus N(S \cup \{v\})$ can be added into $S \cup \{v\}$, thus the following recurrence can be obtained.

$$T(n) \leq \sum_{i=1}^{k} T(n-i) + T(n-k-2). \tag{6}$$

With above the analysis, Eq. (6) is the worst case recurrence of $T(n)$ if $|C \setminus N(S)| + \overline{d}(S) \geq k$.

(3) If $|C \setminus N(S)| + \overline{d}(S) < k$, the pivot-based branching rule is invoked. The following recurrence can be easily obtained.

$$T(n) \leq \sum_{i=1}^{|P_1|} T(n-i) + \sum_{i=1}^{|P_2|} \sum_{j=1}^{|P_2|-i} T(n - |P_1| - i - j). \tag{7}$$

Based on Eq. (4), we can derive that $T(n) \leq \sum_{i=1}^{n} T(n-i)$. Then, the recurrence for Eq. (7) can be improved with:

$$T(n) \leq \sum_{i=1}^{|P_1|} T(n-i) + \sum_{i=1}^{|P_2|} T(n - |P_1| - i). \tag{8}$$

To analyze the bound of Eq. (8), we assume that $\overline{d}(S) = 0$. Let $\overline{d} = |P_1| + |P_2|$. It is easy to verify that $T(n) \leq \sum_{i=1}^{k} T(n-i)$ if $\overline{d} \leq k$. Next, we only focus on the case of $\overline{d} > k$. If there exists a vertex $u \in \overline{N}_v(C)$ with $\overline{d}_u(S) \geq 2$, $Branch(S \cup \{v\}, C \setminus \{v\})$ would perform the branching rule for case (2). Based on the definition of the k-defective clique and the condition of $\overline{d}_u(S \cup \{v\}) \geq 3$, we can note that at most $k-2$ vertices in $\overline{N}_v(C) \setminus \{v\}$ can be added into $S \cup \{v\}$. Thus, we have:

$$\begin{aligned} T(n) &\leq \sum_{i=2}^{\overline{d}} T(n-i) + \sum_{i=1}^{k-2} T(n-1-i) + T(n-\overline{d}) \\ &\leq \sum_{i=1}^{k-1} T(n-i) + T(n-\overline{d}) \leq \sum_{i=1}^{k} T(n-i). \end{aligned} \tag{9}$$

If there is no vertex $u \in \overline{N}_v(C)$ with $\overline{d}_u(S) \geq 2$, we derive that $\overline{d}_u(C) \leq \overline{d}$ for each $u \in \overline{N}_v(G)$. Then the first $\overline{d} - k$ sub-recursive calls of $Branch(S, C)$ would perform the branching rule for case (2). When combining with Eq. (6) and Eq. (10), we have:

$$\begin{aligned} T(n) \leq &\sum_{i=1}^{\overline{d}-k} \sum_{j=1}^{k} T(n-i-j) + \sum_{i=1}^{\overline{d}-k} T(n-\overline{d}-2) \\ &+ \sum_{i=1}^{k} T(n-(\overline{d}-k)-i) \end{aligned} \tag{10}$$

Based on the conditions of $T(n) \leq 2T(n-1)$ and $\sum_{i=1}^{\overline{d}-k} T(n-\overline{d}-2) \leq T(n-k-2)$ if $\overline{d} > k$, the Eq. (10) can be improved with:

$$T(n) \leq \sum_{i=1}^{k} T(n-i) + T(n-k-2). \tag{11}$$

In summary, we obtain that the maximum size of $T(n)$ is bounded by Eq. (6) or Eq. (11). When applying the theoretical result in [3], we can derive that the maximum size of $T(n)$ is bounded by $O(\gamma_k^n)$, where γ_k is the maximum real-root of function $x^{k+3} - 2x^{k+2} + x^2 - x + 1 = 0$ if $k \geq 1$. Thus, this proof is established. □

Theorem 4.4. *For the case of $k = 0$, the time complexity of Algorithm 3 is bounded by $O(m1.414^n)$.*

Proof. When $k = 0$, the algorithm either uses the branch reduction rules or the pivot-based branching rule to detect the maximum k-defective clique. If using the branch reduction rules, the recurrence in Eq. (2) is obtained, as there does not exist any vertex $v \in C$ satisfying $\overline{d}_v(S) \geq 1$. If using the pivot-based branching rule, a recurrence of $T(n) \leq \overline{d}T(n-\overline{d})$ can be obtained. Since $\overline{d} \geq 4$, we derive that $T(n) \leq 4T(n-4)$. Thus, when $k = 0$, the size of $T(n)$ is bounded by $O(\sqrt{2}^n)$, and this theorem is proved. □

Theorem 4.5. *The space complexity of Algorithm 3 is bounded by $O(|S^*|n + m)$, where $|S^*|$ is the size of the maximum k-defective clique of G.*

Proof. Based on the depth-first branching strategy, it is easy to verify that the $Branch(S, C)$ procedure consumes at most $(|S^*|n)$ spaces. Since the algorithm also requires storing the entire graph in the main memory, the overall space usage of Algorithm 3 is bounded by $O(|S^*|n + m)$. □

5 THE HEURISTIC-BASED OPTIMIZATIONS

In this section, we develop a heuristic-based algorithm to further improve the efficiency for detecting the maximum k-defective clique of G. The algorithm follows a three-step approach. Firstly, it identifies a near-maximum k-defective clique of G based on a novel heuristic approach. Then, it removes all vertices whose upper bound for the k-defective clique are no larger than the size of the near-maximum k-defective clique. Finally, the algorithm adapts enumeration algorithm in Sec. 4 to obtain the maximum k-defective clique within the reduced subgraph of G. In the following, we first introduce the heuristic approach for finding the near-maximum k-defective clique and subsequently present the heuristic-based enumeration algorithm.

5.1 The Heuristic Approach

The basic idea of this heuristic approach is to iteratively expand the current k-defective clique S using a candidate set C, where S (C) is initialized with $\emptyset$ (V). Whenever there no vertex in C (the set C is empty) can be used to expand S, a near-maximum maximal k-defective clique S^* is obtained. To make the size of S^* as large as possible, below we propose an ordering-based heuristic approach.

Let $O = \{v_1, v_2, ..., v_n\}$ be an ordering of vertices in G. Assume that $V^+_{v_i}$ is the set of vertices in G that rank higher than v_i according to the ordering O. We denote by $G^+_{v_i}$ the subgraph of G induced by $V^+_{v_i}$. Our heuristic approach is first to compute the near-maximum k-defective clique containing v_i in each $G^+_{v_i}$, and then take the largest among all obtained k-defective clique as the near-maximum k-defective clique of G. Moreover, during the process, we also develop some pruning techniques to remove unnecessary vertices in candidate set C that is used to expand the current k-defective clique S.

Distance-based pruning. Based on Property 2, the diameter of a k-defective clique whose size is no less than $k + 2$ is at most 2. Thus, the distance from each vertex in C to v_i must be no larger than 2 when making use of the vertices in C to expand the initial k-defective clique $S = \{v_i\}$ in subgraph $G^+_{v_i}$.

Non-neighbor-based pruning. When using the set C to expand the current k-defective clique S, then each vertex in C must has at most $k - \overline{d}(S)$ non-neighbors in S.

Degree-based pruning. Let S^* be a near-maximum k-defective clique obtained so far. When using the set C to expand the current k-defective clique S, then each vertex in C must has a degree no less than $|S^*| - k + \overline{d}_u(S)$ in $G(S \cup C)$.

The implementation details. Armed with above pruning techniques, our ordering-based heuristic approach is presented in Algorithm 5.

In Algorithm 5, it first computes the degeneracy ordering of vertices in G, which is achieved by an algorithm proposed in [?] within $O(n + m)$ time (line 1). Then, the algorithm iteratively computes the near-maximum k-defective clique containing v_i in each $G^+_{v_i}$ with the degeneracy ordering O. Specifically,

Theorem 5.1. *Let n' (m') be the maximum number of vertices (edges) in $G(S \cup C_1 \cup C_2)$ obtained in Algorithm 5. Then, the time complexity of Algorithm 5 is $O(n(|S^*|n' + m'))$.*

Proof. ... □

Algorithm 5: The heuristic algorithm

Input: The graph $G = (V, E)$ and a parameter $k \geq 0$
Output: A near maximum k-defective clique S^* in G

1 $V \leftarrow$ a degeneracy ordering of vertices in G;
2 **for** $i = n$ *to* 1 *s.t.* $core(v_i) \geq |S^*| - k$ **do**
3     $S \leftarrow \{v_i\}$; $C_1 \leftarrow N^+_{v_i}(G)$; $C_2 \leftarrow \emptyset$;
4   **while** $\exists u \in C_1$ *with* $d_u(S \cup C_1) < |S^*| - k$ **do**
5         $C_1 \leftarrow C_1 \setminus \{u\}$;
6   **foreach** $v_j \in N^{=2}_{v_i}(G)$ *s.t.* $j > i$ **do**
7   **if** $d_{v_j}(C_1) \geq |S^*| - k$ **then** $C_2 \leftarrow C_2 \cup \{v_j\}$;
8   **while** $S \cup C_1 \cup C_2$ *is not a k-defective clique* **do**
9         $v \leftarrow$ a vertex in $C_1 \cup C_2$ with maximum degree (or degeneracy ordering);
10         $S \leftarrow S \cup \{v\}$ and remove all vertices in $C_1 \cup C_2$ that cannot form a larger k-defective clique with $S \cup \{v\}$;
11   **while** $\exists u \in C_1 \cup C_2$ *s.t.* $d_u(S \cup C_1) < |S^*| - k + \overline{d}_u(S)$ **do**
12   Remove u from $C_1 \cup C_2$;
13   **if** $\exists u \in S$ *with* $d_u(S \cup C_1) < |S^*| - k + \overline{d}_u(S)$ **then**
14             $C_1 \leftarrow \emptyset$; $C_2 \leftarrow \emptyset$; **break**;
15   **if** $|S^*| < |S \cup C_1 \cup C_2|$ **then** $S^* \leftarrow S \cup C_1 \cup C_2$;
16 **return** S^*;

Algorithm 6: The improved algorithm

Input: The graph $G = (V, E)$ and a parameter k
Output: The maximum k-defective clique S^* of G

S^* returned by Algorithm 5;
$G \leftarrow (|S^*| - k)$-core of G;
while $|V| > |S^*|$ **do**
    $v \leftarrow$ a vertex in V with the minimum degree;
    if $\kappa(v, N_v(G)) \leq |S^*| - k$ **then**
        Remove v from V and G; **continue**;
    if $|S^*| < k + 1$ **then** $Branch(\{v\}, V \setminus \{v\})$;
    else
        $S \leftarrow \{v_i\}$; $C_1 \leftarrow N_v(G)$; $C_2 \leftarrow \emptyset$;
        while $\exists u \in C_1$ *with* $d_u(S \cup C_1) < |S^*| - k$ **do**
            $C_1 \leftarrow C_1 \setminus \{u\}$;
        for *each* $u \in N^{=2}_v(G)$ **do**
            if $d_u(C_1) \geq |S^*| - k$ **then** $C_2 \leftarrow C_2 \cup \{u\}$;
        GraphColoring($G(S \cup C_1 \cup C_2), |S^*| - k + 1$);
        $Branch(S, C)$;
        Remove v from V and G;

5.2 The Improved Algorithm

.

.

.

.

.

.

.

.

6 EXPERIMENTS

.

Table 2: Real-world graph datasets.

Datasets	n	m	$d_{\max}$	δ

6.1 Experimental Setup

.

6.2 Experimental Results

.

7 RELATED WORKS

.

8 CONCLUSION

.

REFERENCES

[1] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. 2021. Computing maximum k-defective cliques in massive graphs. *Comput. Oper. Res.* 127 (2021), 105131.

[2] Qiangqiang Dai, Rong-Hua Li, Meihao Liao, and Guoren Wang. 2023. Maximal Defective Clique Enumeration. *Proc. ACM Manag. Data* 1, 1 (2023), 77:1–77:26.

[3] Fedor V. Fomin and Dieter Kratsch. 2010. *Exact Exponential Algorithms.* Springer.

[4] Jian Gao, Zhenghang Xu, Ruizhi Li, and Minghao Yin. 2022. An Exact Algorithm with New Upper Bounds for the Maximum k-Defective Clique Problem in Massive Sparse Graphs. In *AAAI.* 10174–10183.

[5] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. 2018. Maximum weight relaxed cliques and Russian Doll Search revisited. *Discret. Appl. Math.* 234 (2018), 131–138.

[6] Ailsa H. Land and Alison G. Doig. 2010. An Automatic Method for Solving Discrete Programming Problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art.* 105–132.

[7] Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. 2013. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Comput. Optim. Appl.* 56, 1 (2013), 113–130.

[8] Gérard Verfaillie, Michel Lemaître, and Thomas Schiex. 1996. Russian Doll Search for Solving Constraint Optimization Problems. In *AAAI.* 181–187.

[9] Mihalis Yannakakis. 1978. Node- and Edge-Deletion NP-Complete Problems. In *STOC.* 253–264.

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