# Theoretically and Practically Efficient Maximum Defective Clique Search

QIANGQIANG DAI, Beijing Institute of Technology, China
RONG-HUA LI, Beijing Institute of Technology, China
DONGHANG CUI, Beijing Institute of Technology, China
GUOREN WANG, Beijing Institute of Technology, China

The study of $k$-defective cliques, defined as induced subgraphs that differ from cliques by at most $k$ missing edges, has attracted much attention in graph analysis due to their relevance in various applications, including social network analysis and implicit interaction predictions. However, determining the maximum $k$-defective clique in graphs has been proven to be an NP-hard problem, presenting significant challenges in finding an efficient solution. To address this problem, we develop a theoretically and practically efficient algorithm that leverages newly-designed branch reduction rules and a pivot-based branching technique. Our analysis establishes that the time complexity of the proposed algorithm is bounded by $O(m\gamma_k^n)$, where $\gamma_k$ is a real value strictly less than 2 (e.g., when $k = 1$, 2, and 3, $\gamma_k = 1.466$, 1.755, and 1.889, respectively). To our knowledge, this algorithm achieves the best worst-case time complexity to date compared to state-of-the-art solutions. Moreover, to further reduce unnecessary branches, we propose a time-efficient upper bound-based pruning technique, which is obtained by manipulating information such as the number of distinct colors assigned to vertices and the presence of non-neighbors among them. Additionally, we employ an ordering-based heuristic approach as a preprocessing step to improve computational efficiency. Finally, we conduct extensive experiments on a diverse set of over 300 graphs to evaluate the efficiency of the proposed solutions. The results demonstrate that our algorithm achieves a speedup of 3 orders of magnitude over state-of-the-art solutions in processing most of real-world graphs.

CCS Concepts: • **Theory of computation → Branch-and-bound**.

Additional Key Words and Phrases: Cohesive subgraph search, $k$-Defective clique, Branch-and-bound

**ACM Reference Format:**
Qiangqiang Dai, Rong-Hua Li, Donghang Cui, and Guoren Wang. 2024. Theoretically and Practically Efficient Maximum Defective Clique Search. *Proc. ACM Manag. Data* 2, 4 (SIGMOD), Article 206 (September 2024), 27 pages. https://doi.org/10.1145/3677142

## 1 Introduction

Graph has emerged as a versatile model for representing diverse real-world networks, including social networks [2], web networks [24], biological networks [46], and others. The task of identifying cohesive subgraphs from these networks is a fundamental problem in graph analysis, with broad applications in various domains. For instance, community detection in social networks [5, 15], identification of protein complexes in protein-protein interaction (PPI) networks [60, 63], and

Authors' Contact Information: Qiangqiang Dai, qiangd66@gmail.com, Beijing Institute of Technology, Beijing, China; Rong-Hua Li, lironghuabit@126.com, Beijing Institute of Technology, Beijing, China; Donghang Cui, cuidonghang@bit.edu.cn, Beijing Institute of Technology, Beijing, China; Guoren Wang, wanggrbit@126.com, Beijing Institute of Technology, Beijing, China.

ACM 2836-6573/2024/9-ART206
https://doi.org/10.1145/3677142

statistical analysis in financial networks [6, 7] all can be formulated as cohesive subgraph mining problems. Perhaps, the classical clique [33], which requires every pair of vertices associated with an edge, is a commonly-used cohesive subgraph model, as extensively advanced solutions have been proposed in the literature [9, 16, 39, 56].

In real-world applications, it is often too restrictive to mandate the presence of all possible relationships within a community. This is due to the fact that a subgraph missing certain edges can still effectively represent a community [45]. Moreover, real-world networks often involve noise or faults during data collection through experiments or automated sensors [1]. To address this issue, several relaxed clique models have also been extensively studied [45], including the $k$-plex [52], quasi-clique [31], $r$-clique [32], $k$-club [37], $k$-defective clique [63], and others.

In this paper, we primarily focus on the concept of the $k$-defective clique, which is defined as a subgraph $G(S)$ induced by the subset $S$ of the graph $G = (V, E)$, such that it contains at least $\binom{|S|}{2} - k$ edges. This notion was originally introduced in [63] and has proven to be valuable in predicting implicit interactions among proteins in biological networks. The rationale behind this concept is that the missing edges within the $k$-defective clique can be seen as indicative of implicit interactions between proteins. Due to the practical relevance to various real-world applications, including community detection in social networks [21, 44] and statistical analysis in financial networks [14], as well as its close relationship with other cohesive subgraph models, such as the clique [33] and $k$-plex [52], the maximum $k$-defective clique problem, which involves identifying a $k$-defective clique with the largest cardinality among all $k$-defective cliques in a given graph $G$, has recently received significant interest [11, 13, 19, 20, 57].

As shown in [57, 62], the problem of identifying the maximum $k$-defective clique of a given graph $G$ is NP-hard, thereby establishing the infeasibility of a polynomial-time algorithm unless NP=P. To our knowledge, there are a number of solutions that address this challenging problem [11, 13, 19, 20, 57]. Specifically, Trukhanov et al. [57] pioneer an exact algorithm for the maximum $k$-defective clique problem based on the Russian doll search technique [59]. Building upon their work, Gschwinda et al. [20] improve this algorithm by employing new preprocessing methods and an optimized implementation. Furthermore, several new algorithms [11, 13, 19], based on a branch-and-bound search technique [25], have also emerged to improve efficiency. Notably, Chen et al. [13] introduce a new branching rule that prioritizes vertices with non-neighbors in the current $k$-defective clique during the branch-and-bound process. The authors establish that the worst-case time complexity of this technique, used for identifying the maximum $k$-defective clique, is bounded by $O(P(n)\alpha_k^n)$, where $n$ is the number of vertices in the graph $G$, $P(n)$ is a polynomial function dependent on $n$, and $\alpha_k$ is a real-number less than 2. To achieve a better practical efficiency, Gao et al. [19] present an improved branch-and-bound algorithm based on several new branch pruning techniques. More recently, a faster algorithm based on a *non-fully-adjacent-first* branching rule and several reduction rules is developed in [11]. The author shows that the time complexity of this algorithm can be tightened to $O(P(n)\beta_k^n)$, where $\beta_k$ is a real-number smaller than $\alpha_k$ for every $k \geq 1$. To our knowledge, the algorithm developed in [11] represents currently the most advanced solution to the problem of identifying the maximum $k$-defective cliques, both in theoretical and practical terms.

However, these existing solutions still exhibit several noteworthy issues. Firstly, the practical performance remains prohibitively expensive when processing real-world graphs. This issue primarily stems from the insufficient tightening of the upper bound-based pruning techniques and inefficient of the branching rules employed in these existing solutions. Specifically, the presence of overly loose upper bounds often hinders the timely termination of the branch-and-bound process. Moreover, the inefficient branching rules lead to a proliferation of duplicate results, resulting in

Table 1. Summary of different algorithms

| Algorithms | Times | $k =$ | 1 | 2 | 3 | 4 | ... |
|---|---|---|---|---|---|---|---|
| RDS [57], KDBB [19] | $O(P(n)2^n)$ | – | – | – | – | – | ... |
| MADEC [13] | $O(P(n)\alpha_k^n)$ | $\alpha_k =$ | 1.928 | 1.984 | 1.996 | 1.999 | ... |
| kDC [11] | $O(P(n)\beta_k^n)$ | $\beta_k =$ | 1.839 | 1.928 | 1.966 | 1.984 | ... |
| MDC (Ours) | $O(m\gamma_k^n)$ | $\gamma_k =$ | **1.466** | **1.755** | **1.889** | **1.948** | ... |

numerous unnecessary computations within these existing algorithms. Secondly, the theoretical time complexity has not been sufficiently optimized. It is noteworthy that many existing solutions [19, 20, 57] still have a worst-case time complexity of $O(P(n)2^n)$, with only approaches [11, 13] achieving a worst-case time complexity of $O(P(n)\alpha_k^n)$, where $\alpha_k < 2$. However, the value of $\alpha_k$ is of close to 2 for these solutions to find the maximum $k$-defective clique in graph $G$, even for relatively small values of $k$. For instance, as reported in [11], when $k = 1$ and 2, the respective values of $\alpha_k$ are 1.839 and 1.928, respectively.

It is important to emphasize that efficient algorithms for identifying the maximum $k$-defective clique within a given graph $G$ are very beneficial for graph data mining and management tasks. For instance, in domains like social network analysis, the utilization of efficient algorithms not only leads to significant reductions in computation time but also minimizes system cost requirements. This enables swift decision-making and optimization for tasks like personalized recommendations, social advertising, and user relationship management. Consequently, there is an urgent demand for developing more efficient algorithms that can effectively identify the maximum $k$-defective clique in real-world graphs. Addressing this demand is essential for overcoming graph data mining and management challenges across diverse fields, including social network analysis [21, 44] and protein complex discovery [63].

**Contribution.** To address aforementioned issues, in this paper we extensively investigate the problem of finding maximum $k$-defective clique of a given graph $G$, and develop a novel algorithm that combines both theoretical advancements and practical efficiency. The main contributions are summarized below.

*A novel search framework.* To identify the maximum $k$-defective clique of $G$, we develop an elegant search framework, which mainly combines two newly-developed branching rules. These rules ensure the framework's effectiveness in reducing search space. Firstly, if there exists a vertex with at most three non-neighbors within the search space, we employ the newly-proposed branch reduction rules. Secondly, for cases where no such vertex exists, we further utilize a newly pivot-based branching rule to significantly reduce redundant branches. It is worth mentioning that we prove that the time complexity of our framework is bounded by $O(m\gamma_k^n)$, where $\gamma_k$ takes the value of 1.414 if $k = 0$, or the maximum real-root of $x^{k+3} - 2x^{k+2} + x^2 - x + 1 = 0$ if $k \geq 1$. For example, when $k = 1$, 2, and 3, the corresponding values of $\gamma_k$ are 1.466, 1.755, and 1.889, respectively (details in Table 1). To the best of our knowledge, our framework represents the most advanced solution in terms of worst-case time complexity.

*New optimization techniques.* To further improve the efficiency of the proposed framework, we develop a set of optimization techniques. These include upper bound-based pruning and an ordering-based heuristic approach. We show that the size of the maximum $k$-defective clique in a graph $G$ can be effectively bounded by considering several essential factors, such as the vertex degree, core number [51], and number of distinct colors present in $G$. Moreover, we observe that the proposed color-based upper bound can be tightened by further considering the presence of non-neighbors among vertices. Leveraging these observations, we develop a highly efficient pruning technique with a time complexity of $O(kn + \overline{m})$, where $n$ and $\overline{m}$ are the number of vertices and missing edges

Table 2. Frequently-used notations

| Notations | Descriptions |
|---|---|
| $G = (V, E)$ | The undirected and unweighted graph. |
| $N_v(G)$, $\overline{N}_v(G)$ | The set of neighbors, non-neighbors of the vertex $v$ in $G$. |
| $d_v(G)$, $\overline{d}_v(G)$ | The degree of the vertex $v$ in $G$, and the cardinality of $\overline{N}_v(G)$. |
| $G(S) = (S, E_S)$ | The subgraph of $G$ induced by set $S$. |
| $core_v(G)$ | The core number of $v$ in $G$. |
| $\kappa$, $\kappa(C)$ | The size of the maximum $k$-defective clique in $G$ and $G(C)$ |
| $\omega$, $\omega(C)$ | The number of distinct colors in $G$ and $G(C)$ |
| $\delta$ | The maximum core number in $G$. |
| $col_v(G)$ | The color number assigned to $v$ in $G$. |
| $c_v(G)$ | The count of other vertices in $G$ with the same color as $v$. |

in $G$, respectively. In addition to the pruning technique, we also present a novel ordering-based heuristic algorithm. This algorithm functions as a preprocessing step, allowing us to identify a near-maximum $k$-defective clique and greatly reduce unnecessary vertices in $G$.

*Extensive experiments.* We construct extensive experiments to evaluate the efficiency of the proposed algorithms on three distinct sets of datasets with a total of 337 graphs. The experimental results demonstrate that our algorithms substantially outperform the state-of-the-art algorithms in identifying the maximum $k$-defective clique by up to 3 orders of magnitude on most of real-world graphs. For example, on the sc-ldoor dataset (with 21 million edges), our algorithm takes less than 10 seconds to identify the maximum $k$-defective clique when $k = 1$. In contrast, all existing state-of-the-art algorithms failed to terminate within 3 hours under identical conditions. To ensure reproducibility, we make the source code of this work available at https://github.com/dawhc/MaximumDefectiveClique.

## 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 denoted as $d_v(G) = |N_v(G)|$. Similarity, we refer to the set of non-neighbors of $v$ in $G$ as $\overline{N}_v(G) = V \setminus N_v(G)$, and the cardinality of $\overline{N}_v(G)$ as $\overline{d}_v(G) = |\overline{N}_v(G)|$. 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 $E_S = \{(u, v) \in E | u \in S, v \in S\}$. To simplify notation, $N_v(G(S))$ ($\overline{N}_v(G(S))$) and $d_v(G(S))$ ($\overline{d}_v(G(S))$) are abbreviated as $N_v(S)$ ($\overline{N}_v(S)$) and $d_v(S)$ ($\overline{d}_v(S)$), respectively. Below, we present the formal definition of the $k$-defective clique.

*Definition 1 ($k$-defective clique [63]). Given a graph $G$ and a non-negative integer $k$, the subgraph $G(S)$ induced by $S \subseteq V$ is a $k$-defective clique if there exist 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 it contains the largest number of vertices among all maximal $k$-defective clique of $G$. Before formulate our problem, we first introduce two useful properties of the $k$-defective clique, which are very helpful for designing our algorithms.

*Property 1 (Hereditary [57]). 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 the $k$-defective clique simplifies the maximality check process. Specifically, if there is no vertex in $V \setminus S$ that can be used to expand $S$, then $S$ is maximal.

*Property 2 (Small diameter [14]). Given a $k$-defective clique $S$ of $G$, the diameter of $G(S)$ is no larger than 2 if $|S| \geq k + 2$.*

Property 2 not only implies the internal density-connected nature of the $k$-defective clique (diameter 2 for the size no less than $k + 2$) but also provides an acceleration for enumerating relatively-large maximal $k$-defective cliques [14]. Since $k$ is often small (e.g., $k \leq 10$), a relatively-large $k$-defective clique $S$ often satisfies the condition $|S| \geq k + 2$. Consequently, we can use such a diameter constraint for pruning unnecessary search space while finding the maximum $k$-defective clique. Below, we formulate our problem.

**Problem definition.** Given a graph $G$ and a positive integer $k$, the goal of this paper is to compute the maximum $k$-defective clique of $G$, i.e., to find a $k$-defective clique with the largest size among all maximal $k$-defective cliques of $G$. It is noteworthy that the maximum $k$-defective clique search problem we studied pertains to identifying the one among all maximum $k$-defective cliques in a given graph $G$.

As analyzed in Sec. 1, existing solutions [11, 13, 19, 20, 57] are still inefficient in finding the maximum $k$-defective clique. To address this issue, we will propose a theoretically and practically efficient solution in the following sections.

## 3 A Novel Search Framework

In this section, we present a novel framework for efficiently identifying the maximum $k$-defective clique within a given graph $G$. Our framework builds upon a classic branch-and-bound technique [25], which centers around dividing the current problem into smaller sub-problems. Specifically, we define an instance $I = (G, S, C, k)$ that aims to compute the maximum $k$-defective clique containing the set $S$ within the subgraph $G(S \cup C)$. Here, $S$ represents the current partial $k$-defective clique, while $C$ denotes the candidate set used to expand $S$. By selecting a vertex $v$ from $C$, the instance $I$ can be split into two sub-instances: $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ and $I_2 = (G, S, C \setminus \{v\}, k)$, utilizing the branch-and-bound technique. The solution for instance $I$ corresponds precisely to the larger result obtained from either $I_1$ or $I_2$. Consequently, in order to obtain the final solution for $I$, each sub-instance of $I$ can be recursively divided until the candidate set $C$ becomes empty. The overall outcome is determined by selecting the maximum solution among all sub-instances.

However, the total number of sub-instances for the instance $I = (G, S, C, k)$ can be exponentially large, specifically $O(2^n)$, when $S = \emptyset$ and $C = V$. Enumerating all possible sub-instances would be highly inefficient. In order to enhance the performance of such a procedure, it becomes crucial to identify and eliminate unnecessary sub-instances that cannot yield the maximum $k$-defective clique of $G$. Below, we first develop several new branch reduction rules and then present our search framework.

### 3.1 New Branch Reduction Rules

Given an instance $I = (G, S, C, k)$ that focuses on computing the maximum $k$-defective clique containing set $S$ in the subgraph $G(S \cup C)$, we observe that the sub-instances of $I$ can be further reduced under specific conditions. Particularly, if there is a vertex $v$ in $C$ that has at most three non-neighbors within $S \cup C$ (i.e., $\overline{d}_v(S \cup C) \leq 3$), the following three reduction rules apply.

**(1) One non-neighbor reduction: $\overline{d}_v(S \cup C) = 1$.** In this scenario, all other vertices within $S \cup C$ are the neighbors of $v$. Consequently, the maximum $k$-defective clique in $G(S \cup C)$ must necessarily include the vertex $v$, which leads us to obtain the following lemma.

*Lemma 1. Given an instance $I = (G, S, C, k)$, if there is a vertex $v$ in $C$ with $\overline{d}_v(S \cup C) = 1$, then the maximum $k$-defective clique for instance $I$ also exists 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 can be easily verified that the maximum $k$-defective clique in $G(S \cup C)$ must contain at least one vertex in $\{v, u\}$. Let $S_1^*$ be the maximum $k$-defective clique that contains $v$. We observe that for any $k$-defective clique $S_2^*$ where $u \in S_2^*$, it always holds that $|S_1^*| \geq |S_2^*|$. Therefore, in the case where $\overline{d}_v(S \cup C) = 2$, it is unnecessary to find the maximum $k$-defective clique that excludes $v$ in instance $I$. This leads us to the following lemma.

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

*Proof sketch.* Assume that $S^*$ is a maximum $k$-defective clique for instance $I$. It can be easily verified that $|S^* \cap \{v, u\}| \geq 1$, where $u$ is the sole non-neighbor of $v$ in $S \cup C$. When $v \notin S^*$, we can establish that $S^* \setminus \{u\} \cup \{v\}$ also forms a maximum $k$-defective clique if $u \in C$. Moreover, for the case where $u \in S$, we let $w$ be a vertex in $S^* \setminus S$ with the minimum value of $d_w(S^*)$. It follows that $S^*$ can be expanded by $v$, if $d_w(S^*) = |S^*| - 1$. Otherwise, $S^* \setminus \{w\} \cup \{v\}$ will form a maximum $k$-defective clique for instance $I$. Thus, there always exist a maximum $k$-defective clique for instance $I$ that contains $v$. □

Based on Lemma 1 and Lemma 2, we can derive that for a given instance $I = (G, S, C, k)$, if there exists a vertex $v$ in $C$ such that $\overline{d}_v(S \cup C) \leq 2$, then it is sufficient to consider only the scenario where the maximum $k$-defective clique of $I$ includes vertex $v$.

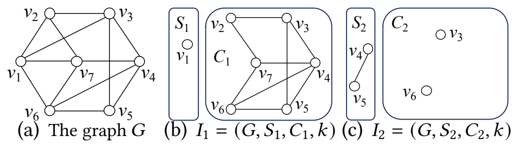
**(3) Three non-neighbors reduction:** $\overline{d}_v(S \cup C) = 3$. Let $u$ and $w$ be the two non-neighbors of vertex $v$ in $S \cup C$. In the instance $I = (G, S, C, k)$, the maximum $k$-defective clique must include at least one vertex from the set $\{v, u, w\}$. Moreover, based on Lemma 2, we obtain that if a maximum $k$-defective clique contains only $u$ or only $w$ among the vertices in $\{v, u, w\}$, there must also exist a maximum $k$-defective clique that includes vertex $v$. Hence, if the maximum $k$-defective clique excludes vertex $v$, it necessarily contains both vertices $u$ and $w$. To further enhance such a result, we introduce the following lemma.

*Lemma 3. Let $S^*$ be a maximum $k$-defective clique for the 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.*

- *Case $\overline{d}_v(S) = 0$: If $(u, w) \notin E$ or $\overline{d}_u(S) + \overline{d}_w(S) \geq 1$, $S^*$ is included in the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$; otherwise, $S^*$ is either included in the sub-instance $I_1$ or the sub-instance $I_2 = (G, S \cup \{u, w\}, C \cap N_u(G) \cap N_w(G), k)$.*
- *Case $\overline{d}_v(S) = 1$: If $\overline{d}_u(S) \geq 1$ with $u \in C$, $S^*$ is included in the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$; otherwise, $S^*$ is either included in the sub-instance $I_1$ or the sub-instance $I_3 = (G, S \cup \{u\}, C \cap N_u(G), k)$.*

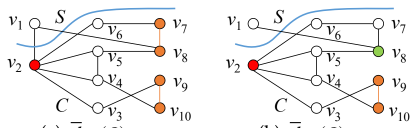
*Proof sketch.* When $\overline{d}_v(S) = 0$, it is easy to verify that $S^*$ contains either the vertex $v$, or both vertices $u$ and $w$ based on Lemma 2. We now consider the case where $\{u, w\} \subseteq S^*$. If $\overline{d}_u(S) \geq 1$ (or $\overline{d}_w(S) \geq 1$) or $(u, w) \notin E$, $S^* \setminus \{u\}$ (or $S^* \setminus \{w\}$) forms a $(k-1)$-defective clique in $G$, which can be expanded by $v$. If $\overline{d}_u(S) + \overline{d}_w(S) = 0$ and $(u, w) \in E$, another sub-instance $I_2 = (G, S \cup \{u, w\}, C \setminus \{u, w\}, k)$ is invoked. It should be noting that only the common neighbors of $u$ and $w$ in $C$ can be used to expand $S \cup \{u, w\}$ for sub-instance $I_2$. Since if there exists a vertex $v'$ in $C \setminus N_u(G)$ (or $C \setminus N_w(G)$) included in $S^*$, $S^* \setminus \{u\}$ (or $S^* \setminus \{w\}$) also forms a $(k-1)$-defective clique in $G$, expandable by $v$. Hence, this theorem holds for the case $\overline{d}_v(S) = 0$. The case $\overline{d}_v(S) = 1$ can be proved similarly. □

The following example illustrates the idea of Lemma 3.



(a) The graph $G$ (b) $I_1 = (G, S_1, C_1, k)$ (c) $I_2 = (G, S_2, C_2, k)$

Fig. 1. Illustrations of three non-neighbor reduction, where the maximum $k$-defective clique $S^*$ of $G$ is either included in $I_1$ or in $I_2$ ($k \geq 2$)



(a) $\overline{d}_{v_2}(S) = 0$ (b) $\overline{d}_{v_2}(S) = 1$

Fig. 2. Illustrations of the pivot-based technique in Theorem 3.2, where red vertices are the pivot vertices. The maximum $k$-defective clique either contains a red or a green vertex or an orange edge ($k \geq 1$)

*Example 1. Consider a graph $G = (V, E)$ depicted in Fig. 1(a) with $k \geq 2$. Since $\overline{d}_{v_1}(G) = 3$, we can utilize the branch reduction rule in Lemma 3. Then, the maximum $k$-defective clique $S^*$ of $G$ is either included in $I_1 = (G, S_1 = \{v_1\}, C_1, k)$ or in $I_2 = (G, S_2 = \{v_4, v_5\}, C_2, k)$, as shown in Fig. 1(b) and Fig. 1(c), respectively. Furthermore, in the sub-instance $I_2$ for computing $S^*$ containing $\{v_4, v_5\}$, the candidate set $C_2$ must be included in common neighbors of $v_4$ and $v_5$. Consequently, $C_2$ contains only the vertices $v_3$ and $v_6$. In addition, in the sub-instance $I_1$, we note that $\overline{d}_{v_4}(S_1 \cup C_1) = 3$. This observation allows us to further apply the three non-neighbor reduction rule to prune $I_1$. Thus, this example serves to illustrate the remarkable pruning capabilities of the proposed reduction rules.*

## 3.2 New Pivot-Based Techniques

Before unveiling our techniques, we first introduce a pivot-based solution initially developed for maximal $k$-defective clique enumerations [14]. The core concept behind this pivot-based technique is as follows: when given an instance $I = (G, S, C, k)$ aimed at enumerating all maximal $k$-defective cliques containing $S$ in $G(S \cup C)$, if there exists a vertex $v \in C$ such that $S \subseteq N_v(G)$, then any maximal $k$-defective clique containing $S$ in $G(S \cup C)$ either includes the vertex $v$ or a non-neighbor vertex of $v$ in $C$. This pivot-based technique is evidently applicable to solving the problem of finding the maximum $k$-defective clique.

However, we note that the approach in [14] is overly restrictive on pivot vertices, resulting in numerous unnecessary computations when adapting this method to identify the maximum $k$-defective clique of $G$. To illustrate this point, let us consider an instance $I = (G, S, C, k)$. If there exist two vertices $u \in S$ and $v \in S$ such that $N_v(C) \cap N_u(C) = \emptyset$, it becomes apparent that there is no vertex $w$ in $C$ that satisfies $S \subseteq N_w(G)$. As a result, the pivot-based technique established in [14] cannot be used to prune sub-instances of $I$, leading to a substantial number of redundant computations. To address this issue, we develop a novel pivot-based technique for finding the maximum $k$-defective clique of $G$, which is presented below.

Theorem 3.1 (New pivoting rule). *Given an instance $I = (G, S, C, k)$ aimed at finding 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 sketch.* If $\overline{d}_v(S) = 0$, this theorem is clearly established [14]. When $\overline{d}_v(S) = 1$, we obtain that $S$ is a $(k-1)$-defective clique, as $v$ can be used to expand $S$. Denote by $C_1 = C \cap N_v(G)$. We note that this theorem disregards the maximum $k$-defective clique contained in $G(S \cup C_1)$. Let $S \cup D$ be the maximum $k$-defective clique in $G(S \cup C_1)$. We now demonstrate that there exists a $k$-defective clique $S^*$ with $|S^*| \geq |S \cup D|$, where $v \in S^*$. Given a vertex $u \in D$ with the minimum $\overline{d}_u(S \cup D)$, we observe that $S \cup D \cup \{v\}$ is a larger $k$-defective clique if $\overline{d}_u(S \cup D) = 1$. Moreover, if $\overline{d}_u(S \cup D) \geq 2$, we obtain that $S \cup D \setminus \{u\} \cup \{v\}$ is also a $k$-defective clique. Thus, there exists a $k$-defective clique $S^*$ containing $v$ with $|S^*| \geq |S \cup D|$. □

Although Theorem 3.1 effectively reduces redundant sub-branches that cannot generate the maximum $k$-defective clique, we find that in instance $I = (G, S, C, k)$, there might still be unnecessary computations when expanding $S$ with vertices in $C \setminus N_v(G)$, where $v$ is the selected pivot vertex from $C$. For instance, let $S^*$ be a maximum $k$-defective clique of instance $I$. If $|S^* \setminus \{v\} \setminus N_v(G)| \geq 2$, it is easy to see that $S^*$ can be identified 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 are used to expand $S$ based on the pivot-based technique described in Theorem 3.1. Consequently, this leads to redundant computations. To overcome this problem, we propose an improved pivot-based technique, which is outlined below.

Theorem 3.2 (Improved pivoting rule). *Consider an instance $I = (G, S, C, k)$, where $v$ is 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 results.*

- *If $\overline{d}_v(S) = 0$, the maximum $k$-defective clique for instance $I$ either contains $v$ or an edge in $G(P)$.*
- *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 sketch.* Let $S^*$ be the maximum $k$-defective clique containing $S$ in $G(S \cup C)$. For the case where $\overline{d}_v(S) = 0$, suppose, on the contrary, that $D = S^* \cap P$ forms an independent set when $v \notin S^*$. Clearly, $D \neq \emptyset$, as $\overline{d}_v(S) = 0$. Given any vertex $u$ in $D$, we obtain that $S^* \setminus \{u\}$ forms a $(k + 1 - |D|)$-defective clique. Moreover, since $\overline{d}_v(S^*) = |D|$, we conclude that $S^* \setminus \{u\} \cup \{v\}$ is also a $k$-defective clique in $G(S \cup C)$. Therefore, if $v \notin S^*$, there exists at least one edge in $G(D)$ for the case where $\overline{d}_v(S) = 0$. A similar analysis can be employed to prove the case where $\overline{d}_v(S) = 1$. □

The example shown in Fig. 2 further illustrates the results described in Theorem 3.2.

*Example 2. Consider the graph $G$ shown in Fig. 2(a) with $k \geq 2$. Let $S = \{v_1\}$ and $C = \{v_2, v_3, ..., v_{10}\}$ be the current $k$-defective clique and the candidate set of $C$, respectively. By selecting $v_2$ as the pivot vertex, we can derive that the maximum $k$-defective clique $S^*$ of $G$ either contains $v_2$ or an edge from $G(\{v_7, v_8, v_9, v_{10}\})$, as stated in Theorem 3.2. However, if $(v_1, v_2) \notin E$ and we still choose $v_2$ as the pivot vertex, then $S^*$ either contains a vertex from $\{v_2, v_8\}$ or an edge from $G(\{v_7, v_9, v_{10}\})$. Notably, since $G(\{v_7, v_9, v_{10}\})$ does not contain an edge involving $v_7$, it is unnecessary to consider the scenario where $S^*$ includes $v_7$, as depicted in Fig. 2(b).*

## 3.3 Implementation of the Search Framework

Utilizing the proposed branch reduction rules and pivoting rules, we develop a novel branching rule to efficiently find the maximum $k$-defective clique in graph $G$, as outlined below.

**Branching rule.** Let $N_S(C) = \{v \in C | S \subseteq N_v(G)\}$ be the set of common neighbors of $S$ in $C$. We obtain the following branching rule for instance $I = (G, S, C, k)$.

- If there exists a vertex $v \in C$ satisfying $\overline{d}_v(S \cup C) \leq 3$ and $\overline{d}_v(S) \leq 1$, the branch reduction rules proposed in Sec. 3.1 are applied to determine the maximum solution for the instance $I$.
- If $|C \setminus N_S(C)| + \overline{d}(S) \geq k \geq 1$ or there is no vertex $v$ in $C$ with $\overline{d}_v(S) \leq 1$, a vertex $v \in C$ with the highest $\overline{d}_v(S)$ is selected 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, the proposed pivoting rule described in Theorem 3.2 is employed to branch the instance $I$.

**Implementation details.** Armed with the proposed branching rule, we devise a new algorithm to identify the maximum $k$-defective clique of $G$, which is shown in Algorithm 1.

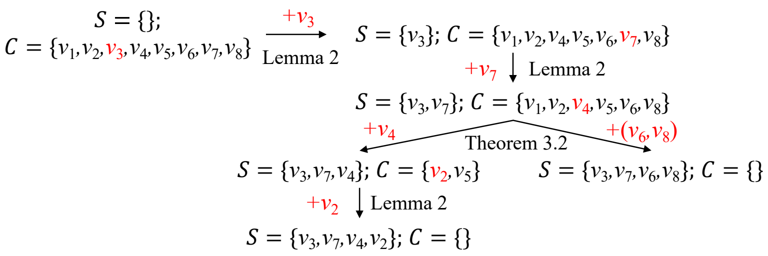
**Algorithm 1:** A New Search Framework

```
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      $C_1 \leftarrow \{u \in C | \overline{d}_u(S) \leq 1\}$; $C_2 \leftarrow C \setminus C_1$;
9      if $\exists u \in C_1$ such that $\overline{d}_u(S \cup C) \leq 3$ then
10         Apply the branch reduction rules in Lemma 1-3;
11     else if $|C \setminus N_S(C)| + \overline{d}(S) \geq k \geq 1$ or $C_1 = \emptyset$ then
12         $v \leftarrow$ a vertex in $C \setminus N_S(C)$ 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 $C_1$ with the largest value of $d_v(C)$;
17         $P_1 \leftarrow \{v\}$; $P_2 \leftarrow \overline{N}_v(C) \setminus P_1$;
18         if $\overline{d}_v(S) = 1$ then
19             $P_1 \leftarrow \{u \in \overline{N}_v(C) | \overline{d}_u(S) = 0\} \cup P_1$;
20             $P_2 \leftarrow \overline{N}_v(C) \setminus P_1$;
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             $C \leftarrow C \setminus \{u\}$;
```

First, Algorithm 1 initializes the current maximum $k$-defective clique $S^*$ as an empty set. Then, it invokes the $Branch(S, C)$ procedure (line 2), which follows our proposed branching rules (lines 9-29). Here, the parameters $S$ and $C$ are denoted by the current $k$-defective clique and the candidate set used to expand $S$, respectively. If there exists a vertex $u \in C$ that satisfies both $\overline{d}_u(S \cup C) \leq 3$ and $\overline{d}_u(S) \leq 1$, the branch reduction rules (proposed in Sec. 3.1) are applied to find the maximum $k$-defective clique that contains $S$ (lines 9-10). If branch reduction rules cannot be used, the procedure determines whether the size of $C \setminus N_S(C)$ is no less than $k - \overline{d}(S)$ or there is no vertex $v$ in $C$ with $\overline{d}_v(S) \leq 1$ (line 11). If this condition holds, the procedure directly identifies the maximum $k$-defective clique that includes or excludes vertex $v$ (line 14), where $v$ is a vertex in $C$ with the maximum value of $\overline{d}_v(S)$ (line 12). If the above conditions are not satisfied, the maximum $k$-defective clique either contains a vertex in $P_1$ or an edge in $G(P_2)$, based on Theorem 3.2. Here, $P_1$ is defined as $\{v\}$ (or $\{v\} \cup \{u \in \overline{N}_v(C) | \overline{d}_v(S) = 0\}$ if $\overline{d}_v(S) = 1$), and $P_2$ as $\overline{N}_v(C) \setminus P_1$, where $v$ is a pivot vertex selected from $C$ to minimize the size of $P_1 \cup P_2$ (line 16). Subsequently, this procedure iteratively

**Algorithm 2:** $Update(S, C, v)$

```
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 $C'$;
```



$S = \{\}$; $C = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$
$+v_3$
Lemma 2
$S = \{v_3\}$; $C = \{v_1, v_2, v_4, v_5, v_6, v_7, v_8\}$
$+v_7$
Lemma 2
$S = \{v_3, v_7\}$; $C = \{v_1, v_2, v_4, v_5, v_6, v_8\}$
$+v_4$
Theorem 3.2
$+(v_6, v_8)$
$S = \{v_3, v_7, v_4\}$; $C = \{v_2, v_5\}$
$S = \{v_3, v_7, v_6, v_8\}$; $C = \{\}$
$+v_2$
Lemma 2
$S = \{v_3, v_7, v_4, v_2\}$; $C = \{\}$

Fig. 3. The branching process of Algorithm 1 for finding the maximum $k$-defective clique in the graph $G$, where $k = 1$ and $G$ is shown in Fig. 4(a)

expands the current $k$-defective clique $S$ by selecting vertices in $P_1$ (lines 21-23) and edges in $G(P_2)$ (lines 24-29). Finally, this recursion terminates when $C$ becomes empty (lines 5-7) and updates $S^*$ if a larger $k$-defective clique is discovered (line 6).

In addition, when a vertex $v \in C$ is added to $S$, it needs to remove the vertices from the candidate set that cannot be used to expand $S \cup \{v\}$. To meet this requirement, we develop a procedure outlined in Algorithm 2, which involves the straightforward removal of each vertex $u$ from $C \setminus \{v\}$ that possesses more than $k - s - \overline{d}_v(S)$ non-neighbors within $S \cup \{v\}$ (lines 2-6). Here, $s$ represents the total number of missing edges in $G(S)$ (line 1). It can be easily verified that the time complexity of Algorithm 2 is bounded by $O(n)$. The following example illustrates the idea of Algorithm 1.

*Example 3. Consider the graph $G$ depicted in Fig.4(a), with $k = 1$. Algorithm 1 initializes the current $k$-defective clique $S$ and its candidate set as $\emptyset$ and $\{v_1, v_2, ..., v_8\}$, respectively. Upon recognizing that there are 2 non-neighbors for both $v_3$ and $v_7$ in $G(S \cup C)$, the algorithm first applies the branching reduction rule, focusing solely on the scenario where the maximum $k$-defective clique contains $S = S \cup \{v_3, v_7\}$. In the subsequent recursive call with $S = \{v_3, v_7\}$, it is observed that $C \subseteq N_S(C)$ and no vertex $v$ in $C$ satisfies $\overline{d}_v(S \cup C) \leq 3$. Consequently, the pivot-based branching rule is invoked. If $v_4$ is selected as the pivot vertex, then $S^*$ either includes vertex $v_4$ or edge $(v_6, v_8)$ based on Theorem 3.2. Opting to expand $\{v_3, v_7\}$ by selecting $v_4$, the algorithm identifies the current maximum $k$-defective clique $S^* = \{v_3, v_7, v_4, v_2\}$. Additionally, upon considering the edge $(v_6, v_8)$ for expansion, it is evident that there exists no vertex in the candidate set of $\{v_3, v_7, v_6, v_8\}$. Thus, the recursive process is finished, yielding the final maximum $k$-defective clique $S^* = \{v_3, v_7, v_4, v_2\}$. The complete branching tree of Algorithm 1 is illustrated in Fig. 3.*

## 3.4 Complexity Analysis

We proceed to analyze the time and space complexity of the proposed algorithm, as outlined below.

THEOREM 3.3. *The time complexity of Algorithm 1 is bounded by $O(m\gamma_k^n)$, where $\gamma_k$ 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 corresponding values of $\gamma_k$ are 1.466, 1.755, and 1.889, respectively.*

*Proof sketch.* Let $T(n)$ be the total number of leaves of $branch(S, C)$ outlined in Algorithm 1. Then, the time complexity of Algorithm 1 is bounded by $O(mT(n))$, as each recursive call of *branch* requires at most $O(m)$ time. We now analyze the size of $T(n)$.

(1) If $\exists v \in C$ with $\overline{d}_v(S \cup C) \leq 3$ and $\overline{d}_v(S) \leq 1$, the branch reduction rules outlined in Sec. 3.1 is used to identify the maximum $k$-defective clique. Let $D = \{u, w\}$ be the set of two non-neighbors of $v$ in $S \cup C$. We then have the following recurrence relations:
$$\begin{cases} T(n) \leq T(n-1), & \text{if } \overline{d}_v(S \cup C) \leq 2; \\ T(n) \leq T(n-1) + T(|N_D(C)|), & \text{if } \overline{d}_v(C) = 3 \text{ and } \overline{d}_v(S) = 0; \\ T(n) \leq T(n-1) + T(|N_u(C)|), & \text{if } \overline{d}_v(C) = 2 \text{ and } \overline{d}_v(S) = 1. \end{cases} \tag{1}$$
It is easy to verify that $T(n) \leq T(n-1) + T(n-3)$ is the worst-case recurrence for the branch reduction rules.

(2) If $|C \setminus N_S(C)| + \overline{d}(S) \geq k$, a base recurrence of $T(n) \leq T(n-1) + T(n-1)$ is obtained. We note that this recurrence can be tightened, as in $branch(S \cup \{v\}, C \setminus \{v\})$, another vertex from $C \setminus N_S(C)$ will be selected to expand $S \cup \{v\}$. This means that at most $k - \overline{d}(S)$ vertices in $C \setminus N_S(C)$ are in preference to be added to $S$. If $\overline{d}_v(S) \geq 2$, it easy to obtain a recurrence of $T(n) \leq \sum_{i=1}^{k} T(n-i)$. If $\overline{d}_v(S) \leq 1$, we have $\overline{d}_v(S \cup C) \geq 4$. Consequently, we derive that the size of $C \setminus N_{S \cup \{v\}}(C)$ is at least $k - \overline{d}(S) + 2$. Hence, the following recurrence relation can be obtained:
$$T(n) \leq \sum_{i=1}^{k} T(n-i) + T(n-k-2). \tag{2}$$

(3) If $C_1 = \emptyset$ (line 8 of Algorithm 1), we obtain that at most $k/2$ vertices in $C$ are possible to be added to $S$. This leads to the following recurrence: $T(n) \leq \sum_{i=1}^{k/2} T(n-i)$, where $k \geq 2$.

(4) If $|C \setminus N_S(C)| + \overline{d}(S) < k$, the pivot-base branching technique is executed. In this case, all vertices in $P_2$ are pairwise connected to produce the worst case recursion. Thus, the pivot-based branching rule produces a recurrence of $T(n) \leq \sum_{i=1}^{|P_1|} T(n-i) + \sum_{i=1}^{|P_2|} \sum_{j=i+1}^{|P_2|} T(n - |P_1| - j)$. Such a recurrence can be improved as:
$$T(n) \leq \sum_{i=1}^{|P_1|} T(n-i) + \sum_{i=1}^{|P_2|-1} T(n - |P_1| - i) = \sum_{i=1}^{|P_1|+|P_2|-1} T(n-i), \tag{3}$$
since $T(n) \leq \sum_{i=1}^{|P_1|} T(n-i) + \sum_{i=1}^{|P_2|} \sum_{j=i+1}^{|P_2|} T(n-|P_1|-j) \leq \sum_{i=1}^{|P_1|} T(n-i) + \sum_{i=1}^{|P_2|} \sum_{j=1}^{|P_1|+|P_2|-1} T(n-|P_1|-i-j) \leq \sum_{i=1}^{|P_1|+|P_2|-1} T(n-i)$. It is easy to verify that $T(n) \leq \sum_{i=1}^{k} T(n-i)$ if $\overline{d} = |P_1| + |P_2| \leq k+1$. For the case of $\overline{d} > k+1$, we note that each sub-recursive call of $Branch(S, C)$ will, in the worst-case scenario, employ the branching rule outlined in case (2). When combining Eq. (2) and Eq. (3), we derive that $T(n)$ is no larger than a constant multiple of the result presented in Eq. (2). Thus, we deduce the following recurrence:
$$T(n) \leq \sum_{i=1}^{k} T(n-i) + T(n-k-2). \tag{4}$$

To summary, we establish that the maximum size of $T(n)$ is bounded by Eq. (4). By utilizing the theoretical result in [17], it can be derived that the maximum size of $T(n)$ can be bounded by $O(\gamma_k^n)$, where $\gamma_k$ is the maximum real-root of function $x^{k+3} - 2x^{k+2} + x^2 - x + 1 = 0$ if $k \geq 1$. Thus, Theorem 3.3 is established. □

THEOREM 3.4. *For the case where $k = 0$, the time complexity of Algorithm 1 is bounded by $O(m1.414^n)$.*

*Proof sketch.* When $k = 0$, Algorithm 1 employs either the branch reduction rules or the pivot-based branching rule. If the branch reduction rules are utilized, we obtain a recurrence of $T(n) \leq T(n-1) + T(n-4)$, as there is no vertex $v \in C$ satisfying $\overline{d}_v(S) \geq 1$. Moreover, if the pivot-based branching rule is employed, we can derive a recurrence of $T(n) \leq \overline{d}T(n-\overline{d})$. Since $\overline{d} \geq 4$, we obtain that $T(n) \leq 4T(n-4) \leq \sqrt{2}^n$. Thus, this theorem is established. □

THEOREM 3.5. *The space complexity of Algorithm 1 is $O(\kappa n+m)$, where $\kappa$ 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 $(\kappa n)$ spaces. Since the algorithm also requires storing the entire graph in the main memory, then the overall space usage of Algorithm 1 is bounded by $O(\kappa n + m)$. □

**Remark.** It is worth highlighting that our branching method offers significant advantages in reducing the unnecessary sub-branches when compared to the state-of-the-art algorithm kDC [11]. This distinction arises from the fact that the branching rule developed in [11] may still attempt to expand the current $k$-defective clique $S$ using all vertices in the candidate set $C$ during recursive calls. In contrast, our proposed branching rule selectively utilizes only a small subset of vertices (or edges) in $C$ to expand $S$ under similar circumstances. To elaborate, when the vertices in $S$ are the neighbors of every vertex in $C$, kDC arbitrarily selects a vertex $v$ in $C$ to find the maximum $k$-defective clique that includes $v$ and excludes $v$, respectively. However, based on the pivot-based branching technique proposed in Theorem 3.2, our algorithm expands $S$ solely using the vertex $v$ and vertices in $\overline{N}_v(C)$ (or edges in $G(\overline{N}_v(C))$), where $v \in C$ is the selected pivot vertex, thus significantly reducing the unnecessary sub-branches. Furthermore, as demonstrated in [11], the worst-case time complexity of kDC is bounded by $O(P(n)\beta_k^n)$, whereas as analyzed in Theorem 3.3, the time complexity of our proposed framework is bounded by $O(m\gamma_k^n)$, which is notably lower than that of kDC (detailed in Table 1). This finding further confirms the superiority of our algorithm over the state-of-the-art algorithm.

## 4 The Proposed Search Algorithm

In this section, we propose several novel optimization techniques to further enhance the efficiency of our framework. Next, we first develop new upper bound-based pruning techniques and then present our algorithms.

### 4.1 The Proposed Upper Bounds

Let $\kappa$ (or $\kappa(C)$) be the size of the maximum $k$-defective clique in graph $G$ (or subgraph $G(C)$). In this subsection, we explore both existing and our proposed upper bounds for $\kappa$ and $\kappa(C)$, which play a crucial role in accelerating the computations of our algorithm.

**Degree-based upper bound.** The first upper bound is derived straightforwardly from the degree information of vertices in $G$, and it is widely used in various maximum $k$-defective clique search algorithms [11, 13, 19]. The details of this upper bound are given in the following lemma.

*Lemma 4. For a given graph $G$, the size of the maximum $k$-defective clique in $G$ that contains a vertex $v \in V$ is at most $d_v(G) + k + 1$. As a consequence, we have $\kappa \leq \max_{v \in V} d_v(G) + k + 1$.*

**Core-based upper bound.** Here we introduce a refined upper bound for both $\kappa$ and $\kappa(C)$, drawing upon the well-established concept of $k$-core [51]. The formal definition of $k$-core is as follows.

*Definition 2 ([51]). Given a graph $G$, the subgraph $G(C)$ of $G$ induced by the set $C$ is a $k$-core of $G$ if $d_v(C) \geq k$ for every $v$ in $C$.*

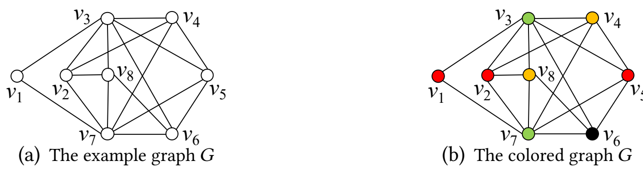


Fig. 4. An illustrative example for graph coloring.

Let $C_k$ be a $k$-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, a tighter upper bound is derived as follows.

*Lemma 5 ([11, 14]). Given graph G, the size of the maximum k-defective clique containing a vertex $v \in V$ in G is bounded by $core_v(G) + k + 1$. Consequently, we have $\kappa \leq \max_{v \in V} core_v(G) + k + 1$.*

Lemma 5 clearly holds, as any $k$-defective clique $S$ is also a $(|S| - k - 1)$-core of $G$. Let $\delta$ be the maximum core number of $G$, representing the highest value of $k$ for which a non-empty $k$-core exists in $G$. We also obtain that $\kappa \leq \delta + k + 1$. To determine the core number for each vertex in a given graph $G$, an algorithm with $O(m + n)$ time developed in [4] can be employed, indicating its remarkable efficiency in generating the core-based upper bound.

**Color-based upper bound.** We observe that the upper bound for $\kappa$ and $\kappa(C)$ can be improved by a graph coloring technique. Below, we begin by providing the formal definition of graph coloring.

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

Denote by $\omega$ 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 derive the following upper bound for $\kappa$ and $\kappa(C)$.

*Lemma 6. Given a coloring of the graph G and a subgraph $G(C)$ of 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 sketch.* Given a $k$-defective clique $S$, we partition it into two disjoint subsets, $S_1$ and $S_2$, while ensuring the constraints: for each $v \in S_1$ (resp. $u \in S_2$), it holds that $\nexists w \in S_1 \setminus \{v\}$ with $col_v(G) = col_w(G)$ (resp. $\exists w \in S_1$ with $col_u(G) = col_w(G)$). It can be verified that $|S_1| \leq \omega$ and $|S_2| \leq k$. Thus, this lemma is established. □

Lemma 6 demonstrates that finding a smaller value for $\omega$ ($\omega(C)$) can achieve a better upper bound. However, it is worth noting that determining the smallest value of $\omega$ ($\omega(C)$) for graph coloring poses a computational challenge and is known to be NP-hard [23]. Thus, numerous heuristic approaches have been explored to address graph coloring problem [26, 55]. In this paper, we adopt a widely used degeneracy ordering heuristic for graph coloring. The definition of degeneracy ordering [30] is presented below.

*Definition 4. Given a graph $G = (V, E)$, 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 $G(\{v_i, v_{i+1}, ..., v_n\})$.*

The degeneracy ordering, akin to the technique employed for computing the $k$-core of a graph, can be obtained by the classic peeling algorithm [4]. Specifically, the vertex removal ordering aligns with the degeneracy ordering, which can be accomplished in a time complexity of at most $O(n + m)$ [4]. Subsequently, we can assign colors to each vertex $v$ of the graph $G$ in a reverse order of the degeneracy ordering. As a result, the upper bound, derived from graph coloring, can be efficiently computed in $O(n + m)$ time.

**Algorithm 3:** *Upperbound*$(S, C, k)$

```
1 $D \leftarrow \emptyset$; $s \leftarrow$ the missing edges in $G(S)$;
2 while $C \neq \emptyset$ do
3     $v \leftarrow$ a vertex in $C$ with minimum $\overline{d}_v(S) + c_v(D)$;
4     if $s + \overline{d}_v(S) + c_v(D) > k$ then break;
5     $s \leftarrow s + \overline{d}_v(S) + c_v(D)$;
6     $D \leftarrow D \cup \{v\}$; $C \leftarrow C \setminus \{v\}$;
7 return $|S| + |D|$;
```

The following example illustrates the proposed upper bounds.

*Example 4. Consider the graph $G$ depicted in Fig. 4(a). It can be seen that the maximum degree $d_{max}$ and maximum core number $\delta$ of vertices in $G$ are $5$ and $4$, respectively. By Lemma 4-5, we obtain that $\kappa$ is limited to $7$ or $6$ when $k = 1$. However, when employing the color-based upper bound technique (the colored graph shown in Fig. 4(b)), we derive that $\kappa \leq 5$ when $k = 1$, as there exist $4$ distinct colors assigned to vertices in $G$, which confirms the efficiency of the proposed color-based upper bound.*

**Advanced color-based upper bound.** Let $S$ be a vertex subset of $G$, and $\kappa(S, C)$ represent the size of the maximum $k$-defective clique in $G(S \cup C)$ that includes all vertices in $S$. We define $c_v(C)$ as the count of other vertices in $C$ that share the same color as vertex $v$, denoted as $c_v(C) = |\{u \in C \setminus \{v\} | col_v(G) = col_u(G)\}|$. With these definitions, we present the following lemma.

*Lemma 7. Consider a graph $G$ and a non-maximal $k$-defective clique $S$ of $G$. If there exists a vertex set $D \subseteq V \setminus S$ 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 $G(S \cup D)$.*

It is easy to derive that Lemma 7 establishes. Denote by $\overline{d}(S)$ the total number of missing edges in $G(S)$, i.e., $\overline{d}(S) = \frac{1}{2}\sum_{v \in S}(|S| - d_v(S) - 1)$. A lemma states the following.

*Lemma 8. Given 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 finding the maximum $k$-defective clique containing $S$ in the subgraph $G(S \cup C)$, we have $\kappa(S, C) \leq |S| + |D|$.*

*Proof sketch.* Given a subset $D$ of $C$, we observe that the number of missing edges in $G(D)$ is at least $\frac{1}{2}\sum_{v \in D} c_v(D)$. Moreover, since each vertex $v$ in $D$ has $\overline{d}_v(S)$ non-neighbors in $S$, we can derive that the number of missing edges in $G(S \cup D)$ will increase by at least $\sum_{v \in D}(\overline{d}_v(S) + \frac{1}{2}c_v(D))$ if we add $D$ to $S$. Thus, if $D$ is 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 obtain that $\kappa(S, C) \leq |S| + |D|$. □

The following example illustrates the superiority of the proposed Lemma 8.

*Example 5. Reconsider the colored graph shown in Fig. 4(b). Assume that $S = \{v_2\}$ and $C = \{v_1, v_3, ..., v_8\}$, with $k = 1$. By Lemma 8, we can obtain that $\kappa(S, C) \leq |S| + |D|$, where $D \subseteq C$. If $v_6 \in D$, we have $|D| \leq 3$ since $\overline{d}_{v_6}(S) = 1$ and there are only two colors in $C \setminus \{v_6\}$ that different from the color of $v_2$; otherwise, if $v_6 \notin D$, we still have $|D| \leq 3$. Thus, we obtain that $\kappa(S, C) \leq 4$. However, when utilizing Lemma 6, it yields $\kappa(S, C) \leq 5$, which is worse than the result obtained by Lemma 8.*

Based on Lemma 8, we proposed an algorithm, as shown in Algorithm 3, to compute the upper bound of $\kappa(S, C)$. Initially, the algorithm initializes $D$ as an empty set (line 1). Then, the algorithm iteratively selects a vertex $v$ from $C$ that has the smallest value of $\overline{d}_v(S) + c_v(D)$ and adds it to the current subset $D$ (lines 2-6). Note that whenever $v$ is selected to move from $C$ to $D$, the missing edges in $G(S \cup D)$ increase by at least $\overline{d}_v(S) + c_v(D)$ (lines 5-6). Finally, when $C$ becomes empty or the number of missing edges in $G(S \cup D)$ violates the definition of a $k$-defective clique, the algorithm terminates and outputs $|S|+|D|$ as the upper bound of $\kappa(S, C)$ (lines 2 and 4). The following theorem establishes the correctness of Algorithm 3.

Theorem 4.1. *Algorithm 3 correctly computes the upper bound of $\kappa(S, C)$.*

*Proof sketch.* On the contrary, assume that there exist a subset $D'$ of $C$ with $|D'| > |D|$ that satisfies $\sum_{v\in D'}(\overline{d}_v(S) + \frac{1}{2}c_v(D')\}) \leq k - \overline{d}(S)$. It can be seen that there exist two vertices $v \in D$ and $u \in D' \setminus D$ with $\overline{d}_v(S) + c_v(D \setminus \{v\}) > \overline{d}_u(S) + c_u(D \setminus \{v\})$. However, whether we consider the condition $col_v(G) = col_u(G)$ or $col_v(G) \neq col_u(G)$, we always conclude that the vertex $u$ will be pushed into $D$ in preference to $v$ in our algorithm. This results in a contradiction. □

Theorem 4.2. *The time complexity of Algorithm 3 is bounded by $O(kn+\overline{m})$, where $\overline{m}$ is the number of missing edges in $G(C)$.*

Proof. Algorithm 3 first sorts each vertex $v$ in set $C$ by the size of $\overline{d}_v(S)$, which can be done efficiently in $O(kn)$ time using a bin sort. Then, 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 particular operation takes at most $O(\overline{d}_v(C))$ time. Consequently, the overall time complexity of executing lines 2-6 in Algorithm 3 amounts to $O(\overline{m})$. □

**Relations of our proposed bounds with related approaches.** Within the existing literature [11, 13], several color-based upper bounds have been developed to enhance the efficiency of finding the maximum $k$-defective clique. However, it is important to note that our proposed upper bounds can be tighter than those in [11, 13]. Specifically, in [13], it is demonstrated that the upper bound of $\kappa(C)$ is bounded by $\sum_{i=1}^{\omega} \min(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |\pi_i|)$, where $\pi_i$ is the subset of all vertices in $C$ with the color number of $i$. It can be easily verified that $\sum_{i=1}^{\omega} \min(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |\pi_i|) \geq \omega + k$ when $k$ is small. This result demonstrates that color-based upper bound in [13] is looser than the one proposed in Lemma 6. Moreover, in [11], an improved color-based upper bound technique is further developed. This technique first assigns a weight $w(v_j) = \overline{d}_{v_j}(S) + j - 1$ for each vertex $v_j$ in $\pi_i$ if $v_j$ is used to expand $S$. Then, the maximum value of $i$ plus $|S|$ serves as the corresponding upper bound if $\sum_{j=1}^{i} w(v_j) \leq k - \overline{d}(S)$. However, we observe that this result is dominated by our result shown in Lemma 8. Let $D$ be the subset obtained by Lemma 8. By utilizing the method developed in [11], we partition $D$ into each $\pi_i$ and assign weights $w(v_j)$ for each vertex $v_j$ in $\pi_i$. Then, we can obtain that $\sum_{v_j\in D} w(v_j) \leq k - \overline{d}(S)$, as the size of $\pi_i$ is no larger than $c_v(D)$, where $v$ is a random vertex in $\pi_i$. Thus, the upper bound obtained in [11] is not tighter than that in Lemma 8. These findings effectively demonstrate the tightness of our proposed upper bounds.

## 4.2 Finding a Heuristic Result

To find a heuristic result, we can make use of classical approach that iteratively selects a vertex from $C$ to expand $S$, where $S$ and $C$ are initialized as an empty set and $V$, respectively. Once no more vertices from $C$ can be added to $S$ (i.e., $C$ becomes empty), we obtain a near-maximum maximal $k$-defective clique denoted as $S^*$. To further improve the size of $S^*$, we propose an ordering-based heuristic approach, which is outlined as follows.

Let $O = \{v_1, v_2, ..., v_n\}$ be an ordering of vertices in $G$. We define $V^+_{v_i}$ as the set of vertices in $G$ that rank higher than $v_i$ in the ordering $O$. Let $G^+_{v_i}$ be the subgraph of $G$ induced by $V^+_{v_i}$. Then, our heuristic approach focuses on computing a near-maximum $k$-defective clique in a subgraph $G^+_{v_i}$ for

**Algorithm 4:** A 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  Let $\{v_1, v_2, ..., v_n\}$ be the degeneracy ordering of vertices in $G$;
2  for $i = n$ to 1 s.t. $core(v_i) \geq |S^*| - k$ do
3      $S \leftarrow \{v_i\}$; $C \leftarrow N_{v_i}(G^+_{v_i})$;
4      while $\exists u \in C$ with $d_u(C) < |S^*| - k - 1$ do
5          $C \leftarrow C \setminus \{u\}$;
6      while $C \neq \emptyset$ do
7          $v \leftarrow$ a vertex in $C$ with maximum degree (or maximum core number) in $G(S \cup C)$;
8          $S \leftarrow S \cup \{v\}$ and remove each vertex $u$ from $C$ if $\overline{d}_u(S)$ is larger than $k - \overline{d}(S)$;
9          while $\exists u \in C$ s.t. $d_u(N_S(C)) < |S^*| - |S| - k + \overline{d}(S)$ do
10             Remove $u$ from $C$;
11         if $\exists u \in S$ with $d_u(N_S(C)) \leq |S^*| - |S| - k + \overline{d}(S)$ then
12             $S \leftarrow \emptyset$; $C \leftarrow \emptyset$;
13     foreach $v_j \in N^{=2}_{v_i}(G)$ s.t. $j > i$ do
14         if $d_{v_j}(S) \geq |S^*| - k + \overline{d}(S)$ then $C \leftarrow C \cup \{v_j\}$;
15     Further expand $S$ with vertices in $C$ as described in lines 6-12;
16     if $|S^*| < |S|$ then $S^* \leftarrow S$;
17 return $S^*$;
```

each vertex $v_i$. By executing this approach, we obtain several $k$-defective cliques, and the largest among them is selected as the near-maximum $k$-defective clique of $G$. Below, we also introduce some pruning techniques throughout this process.

**Pruning techniques.** When expanding the set $S$ with vertices from $C$, we observe that many vertices in $C$ can be pruned effectively. Thus, we also employ three specific pruning techniques based on the current near-maximum $k$-defective clique denoted as $S^*$.

- *Distance-based pruning.* By Property 2, we determine that if $|S^*| \geq k + 2$, the diameter of $G(S^*)$ is at most 2. Consequently, during the expansion of $S = \{v_i\}$, only vertices in $G^+_{v_i}$ whose distance to $v_i$ is not greater than 2 will be initialized to $C$.
- *Non-neighbor-based pruning.* We obtain that any vertex in $C$ possessing more than $k - \overline{d}(S)$ non-neighbors in $S$ can be safely excluded from $C$, as such vertices cannot contribute to the formation of a larger $k$-defective clique when combined with $S$.
- *Common neighbor-based pruning.* Let $N_S(C) = \{u \in C | S \subseteq N_u(G)\}$ be the set of common neighbors of $S$ in $C$. Given a vertex $u \in C$, if $d_u(N_S(C)) < |S^*| - k - |S| + \overline{d}(S)$, it follows that $u$ cannot be part of a maximum $k$-defective clique with a size equal to or greater than $|S^*|$. Consequently, such a vertex $u$ can be safely removed from $C$.

**Implementation details.** Equipped with the proposed techniques, we outline our heuristic approach in Algorithm 4.

Initially, Algorithm 4 computes the degeneracy ordering of vertices in $G$ using a method described in [4]. Subsequently, the algorithm iteratively computes the near-maximum $k$-defective clique containing $v_i$ in $G^+_{v_i}$ for each $v_i$ in the degeneracy ordering (lines 2-16). The algorithm constructs the candidate set $C$ with vertices in $N_{v_i}(G^+_{v_i})$ to expand the current $k$-defective clique $S = \{v_i\}$. Then, it progressively selects a vertex $v$ from $C$ with the largest degree (or maximum core number) to expand $S$ until $C = \emptyset$. Notably, when a vertex $v$ is added to $S$, all the remaining vertices in the candidate set $C$ used to expand $S \cup \{v\}$ adhere to the non-neighbor-based pruning (line 8) and the common

**Algorithm 5:** The MDC Algorithm

```
Input: The graph $G = (V, E)$ and a parameter $k$
Output: The maximum $k$-defective clique $S^*$ of $G$
1  $S^*$ returned by Algorithm 4;
2  $G \leftarrow (|S^*| - k)$-core of $G$;
3  Let $\{v_1, v_2, ..., v_n\}$ be the degeneracy ordering of vertices in $G$;
4  for $i = n$ to $1$ s.t. $core_{v_i}(G) \geq |S^*| - k$ do
5      $S \leftarrow \{v_i\}; C_1 \leftarrow N_{v_i}(G^+_{v_i}); C_2 \leftarrow \emptyset$;
6      while $\exists u \in C_1$ with $d_u(C_1) < |S^*| - k - 1$ do
7          $C_1 \leftarrow C_1 \setminus \{u\}$;
8      if $|S \cup C_1| \leq |S^*| - k$ then continue;
9      for each $u \in N^{=2}_{v_i}(G^+_{v_i})$ do
10         if $d_u(C_1) \geq |S^*| - k$ then $C_2 \leftarrow C_2 \cup \{u\}$;
11     if $|S^*| < k + 1$ then $C_2 \leftarrow \{v_{i+1}, v_{i+2}, ..., v_n\} \setminus C_1$;
12     Coloring $G(S \cup C_1 \cup C_2)$ with degeneracy ordering;
13     $Branch(S, C_1 \cup C_2)$; // Equipped with the proposed upper bounds in Sec. 4.1
```

neighbor-based pruning (lines 4-5, lines 9-10). Furthermore, utilizing the distance-based pruning, the algorithm further expands $S$ by considering the vertices with a distance of 2 from $v_i$ using a similar technique as before (lines 13-16), where $N^{=2}_{v_i}(G) = \{u \in V | u \notin N_{v_i}(G), N_u(G) \cap N_{v_i}(G) \neq \emptyset\}$. Finally, the heuristic algorithm terminates after each vertex in $V$ has been processed and returns the largest $k$-defective clique detected as the final near-maximum $k$-defective clique of $G$ (line 17). The time complexity of Algorithm 4 is provided below.

Theorem 4.3. *Let $n'$ ($m'$) be the maximum number of vertices (edges) in $G(S \cup C)$ obtained in Algorithm 4. Then, the time complexity of Algorithm 4 is bounded by $O(n(\kappa n' + m'))$.*

Proof. It is evident that the algorithm requires at most $O(m')$ time to perform the common neighbor-based pruning in $G(S \cup C)$ (lines 4-5 and lines 9-10). Additionally, when a vertex $v$ is added into $S$, it necessitates at most $O(n')$ time to update the candidate set. Therefore, lines 7-8 of Algorithm 4 consume at most $O(\kappa n')$ time, as at most $\kappa$ vertices in $C$ can be added into $S$. Hence, the total time taken by Algorithm 4 is bounded by $O(n(\kappa n' + m'))$. □

Note that the practical performance of Algorithm 4 can be highly efficient (as highlighted in Sec. 5.2). This is mainly due to the fact that the size of $C$ in lines 2-12 is bounded by $\delta$, which is relatively small in real-world graphs. Moreover, for most subgraphs $G(S \cup C)$, the number of vertices is much less than $n'$. Consequently, the practical performance of Algorithm 4 is considerably lower than its worst-case time complexity.

## 4.3 The Proposed MDC Algorithm

By integrating all the above techniques, we propose our MDC algorithm in Algorithm 5. Specifically, it begins by employing the proposed heuristic algorithm (Algorithm 4) to acquire a near-maximum $k$-defective clique $S^*$ in $G$ (line 1). Subsequently, this algorithm identifies a $(|S^*| - k)$-core subgraph and focuses mainly on computing the maximum $k$-defective clique of $G$ within this subgraph (line 2). This is because other vertices cannot be part of a $k$-defective clique with a size surpassing $|S^*|$. The algorithm then iteratively computes the maximum $k$-defective clique containing $v_i$ within $G$, following the reverse order of the degeneracy ordering (lines 3-12). Prior to the branching process, this algorithm also employs the pruning techniques outlined in Sec. 4.2 to reduce the size of the candidate set of $S = \{v_i\}$. Notably, each vertex in the candidate set $C_1 \cup C_2$ satisfies two conditions:

it is at most 2 distances away from $v_i$, and it shares at least $|S^*| - k$ common neighbors with $v_i$ (lines 5-10). Moreover, if the size of $S \cup C_1$ is not larger than $|S^*| - k$, the computation for $v_i$ can be skipped by our observations (line 8). Following this, the algorithm employs the degeneracy ordering to color each vertex in $G(S \cup C_1 \cup C_2)$ (line 12), which serves as a prerequisite for Algorithm 3. Finally, the algorithm invokes the *Branch* procedure developed in Sec. 3.3 to compute the maximum $k$-defective clique containing $v_i$ in $G$, and updates the current maximum result $S^*$ if a larger $k$-defective clique is obtained (lines 13). Note that it shares the same worst-case time complexity with Algorithm 1, but it equipped with several non-trivial and effective pruning techniques, thus Algorithm 5 is very efficient in searching the maximum $k$-defective clique (as it confirmed in our experiments).

## 5 Experiments

In this section, we conduct extensive experiments to evaluate the efficiency of our algorithms. Below, we first introduce the experimental setup, and then report the experimental results.

### 5.1 Experimental Setup

**Algorithms.** We implement an algorithm called MDC to identify the maximum $k$-defective clique of $G$, which contains all proposed techniques as detailed in Algorithm 5. To assess the performance of our proposed algorithms, we also use the state-of-the-art algorithms kDC, KDBB, and MADEC as the baseline algorithms, where kDC, KDBB, and MADEC are the maximum $k$-defective clique search algorithms developed in [11], [19], and [13] respectively. It is worth noting that due to the absence of open-source code for KDBB, we utilize our own implementation for the experiments, which exhibits better performance compared to the reported results in the literature. All the tested algorithms are implemented in C++, and tested on a PC with one 2.2 GHz CPU and 64GB memory running CentOS operating system.

**Datasets.** We employ three distinct sets of datasets to evaluate the efficiency of our proposed algorithm. The first set of datasets, consisting of 139 massive real-world graphs, is originally obtained from the Network Data Repository [48]. These datasets have been widely used in various studies [18, 19, 64] and can be downloaded from http://lcs.ios.ac.cn/~caisw/graphs.html. The second set of datasets is available at https://networkrepository.com/socfb.php, which includes 114 Facebook graphs. Lastly, the third set of datasets comprises 84 DIMACS10 graphs, which can be accessed at https://networkrepository.com/dimacs.php. All tested graphs have been utilized as benchmark graphs for evaluating the performance of the maximum $k$-defective clique search algorithms [11, 19].

**Parameters.** In our experimental evaluations, we consider values of $k$ as 1, 5, 10, 15 and 20 following a similar approach as [11, 13, 19]. Moreover, we note that the size of the maximum $k$-defective clique in certain datasets is relatively small. Consequently, setting $k$ to excessively large values may lack meaningful implications. Thus, we also impose an additional constraint of $k < \kappa - 1$, where $\kappa$ is the size of the maximum $k$-defective clique in a given graph $G$.

Note that in various real-world graphs like social networks and web graphs, the communities tend to be relatively large (often exceeding 20 or even 30). Such a large size demonstrates that the communities obtained from the maximum $k$-defective clique still exhibit very high densities even when setting larger values for $k$. For example, when $k = 20$, the density of a $k$-defective clique with a size of at least 30 is approximately 0.95. Moreover, as shown in the experimental results in Table 3, we can observe that there are many real-world graph with the size of maximum $k$-defective clique no less than 24 when $k = 1$. These demonstrate that developing a high performance maximum $k$-defective search algorithm for larger $k$ values is also very necessary for handling real-world

Table 3. Statistics of benchmark real-world graphs, where $\delta$, $Clq$, and $\kappa$ represent the degeneracy size, the size of the maximum clique, and the size of the maximum $k$-defective clique of $G$, respectively

| ID | Dataset | $n$ | $m$ | $\delta$ | $Clq$ | $\kappa$ when $k$ = | | | | | ID | Dataset | $n$ | $m$ | $\delta$ | $Clq$ | $\kappa$ when $k$ = | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | 1 | 5 | 10 | 15 | 20 | | | | | | | 1 | 5 | 10 | 15 | 20 |
| D1 | ia-enron-large | 33.6K | 180K | 43 | 20 | 21 | 23 | 25 | 26 | 27 | D19 | socfb-Texas80 | 31.6K | 1.2M | 78 | 59 | 60 | 63 | 65 | 68 | 69 |
| D2 | sc-ldoor | 909K | 20.8M | 34 | 21 | 21 | 21 | 22 | 23 | 23 | D20 | socfb-Texas84 | 36.3K | 1.6M | 81 | 51 | 52 | 55 | 58 | 60 | 61 |
| D3 | sc-nasasrb | 54.8K | 1.3M | 35 | 24 | 24 | 24 | 24 | 25 | 25 | D21 | socfb-Tulane29 | 7.8K | 284K | 68 | 38 | 39 | 42 | 45 | 47 | 48 |
| D4 | sc-pkustk11 | 87.8K | 2.6M | 47 | 36 | 36 | 36 | 36 | 37 | 37 | D22 | socfb-UF | 35.1K | 1.5M | 83 | 55 | 56 | 59 | 62 | 64 | 66 |
| D5 | sc-pkustk13 | 94.8K | 3.3M | 41 | 36 | 36 | 36 | 37 | 37 | 38 | D23 | socfb-UGA50 | 24.3K | 1.2M | 86 | 52 | 53 | 55 | 58 | 59 | 61 |
| D6 | sc-pwtk | 218K | 5.6M | 35 | 24 | 24 | 24 | 25 | 26 | 27 | D24 | socfb-Vanderbilt48 | 8.06K | 427K | 86 | 45 | 46 | 50 | 52 | 55 | 56 |
| D7 | sc-shipsec1 | 140K | 1.7M | 24 | 24 | 24 | 24 | 25 | 26 | 27 | D25 | socfb-Wisconsin87 | 23.8K | 836K | 60 | 37 | 38 | 40 | 42 | 44 | 46 |
| D8 | sc-shipsec5 | 179K | 2.2M | 29 | 24 | 24 | 24 | 25 | 26 | 27 | D26 | soc-digg | 771K | 5.9M | 236 | 50 | 51 | 53 | 55 | 58 | 61 |
| D9 | scc_fb-forum | 488 | 71K | 272 | 266 | 266 | 268 | 269 | 270 | 271 | D27 | soc-flixster | 2.5M | 7.9M | 68 | 31 | 32 | 36 | 39 | 41 | 42 |
| D10 | scc_reality | 6.8K | 4.7M | 1235 | 1236 | 1236 | 1236 | 1237 | 1238 | 1239 | D28 | soc-gowalla | 197K | 950K | 51 | 29 | 30 | 31 | 32 | 33 | 34 |
| D11 | socfb-Amherst41 | 2.2K | 90.9K | 63 | 21 | 22 | 25 | 28 | 30 | 32 | D29 | soc-orkut | 3M | 106M | 230 | 47 | 48 | 50 | 53 | 55 | 57 |
| D12 | socfb-Auburn71 | 18.4K | 973K | 95 | 57 | 58 | 61 | 63 | 65 | 67 | D30 | soc-pokec | 1.6M | 22.4M | 47 | 29 | 30 | 31 | 32 | 33 | 34 |
| D13 | socfb-A-anon | 3.1M | 23.6M | 74 | 25 | 26 | 28 | 30 | 32 | 34 | D31 | soc-slashdot | 70K | 359K | 53 | 26 | 27 | 30 | 32 | 34 | 36 |
| D14 | socfb-B-anon | 2.9M | 20.9M | 63 | 24 | 25 | 28 | 30 | 32 | 33 | D32 | soc-youtube | 496K | 1.9M | 49 | 16 | 17 | 20 | 22 | 23 | — |
| D15 | socfb-Columbia2 | 11.7K | 444K | 66 | 31 | 32 | 33 | 35 | 37 | 39 | D33 | tech-as-skitter | 1.7M | 11.1M | 111 | 67 | 68 | 70 | 72 | 74 | 75 |
| D16 | socfb-Duke14 | 9.9K | 506K | 85 | 34 | 35 | 38 | 40 | 42 | 44 | D34 | tech-WHOIS | 7.5K | 56.9K | 88 | 58 | 59 | 62 | 64 | 67 | 69 |
| D17 | socfb-FSU53 | 27.7K | 1.03M | 81 | 56 | 57 | 60 | 63 | 65 | 67 | D35 | web-spam | 4.8K | 37.4K | 35 | 20 | 21 | 21 | 23 | 24 | 26 |
| D18 | socfb-Indiana | 29.7K | 1.3M | 76 | 48 | 49 | 51 | 53 | 55 | 57 | D36 | web-uk-2005 | 130K | 11.7M | 499 | 500 | 500 | 500 | 500 | 500 | 500 |

graphs. Thus, in our experiments, we also evaluate the performance of the proposed MDC algorithm for larger value of $k$ (i.e., $k$ grows to 20).

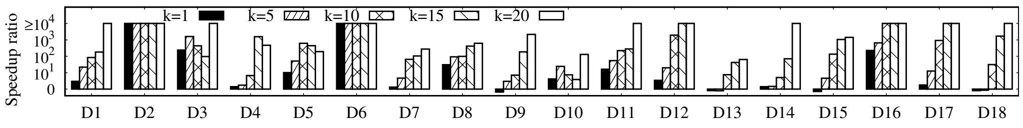
## 5.2 Experimental Results

**Exp-1: Results on representative benchmark graphs.** In this experiment, we evaluate the runtime of various algorithms in finding the maximum $k$-defective clique on representative graphs. Table 3 shows the detailed statistics of 36 benchmark real-world graphs, where the columns $\delta$ and $Clq$ denote the degeneracy and the size of the maximum clique of the graph, respectively, the columns "$\kappa$ when $k = 1$" represent the size of the maximum $k$-defective clique $\kappa$ in the graph under the specific value of $k$. Table 4 shows the runtime of MDC on different datasets with varying $k$; and Fig. 5 displays the speedup ratio of MDC over the best performance between kDC and KDBB in terms of runtime, where the runtime of each algorithm also incorporates the pre-processing time of the degeneracy ordering, "—" denotes that the algorithm failed to complete the computations within a time threshold of 10800 seconds (3 hours). From the results, we can obtain that except for a few datasets that are easily processed by all tested algorithms, the runtime of our algorithm MDC consistently outperforms all existing algorithms (kDC and KDBB) across different values of $k$. Specifically, our algorithm achieves a speedup of 3 orders of magnitudes over kDC and KDBB on most parameter settings. For instance, when $k = 1$ and considering the sc-ldoor dataset, MDC only requires 7.57 seconds to identify the maximum $k$-defective clique, whereas both kDC and KDBB fail to complete the computation within the given 10800 seconds. Furthermore, it can be observed that the time consumption of our algorithm is relatively insensitive to an increase in $k$ on most datasets. Conversely, existing algorithms exhibit a sharp increase in runtime as $k$ increases. For example, on the socfb-Texas84 dataset, when $k$ is set to 1 and 15, MDC takes only 0.48 and 2.84 seconds, respectively, to find the maximum $k$-defective clique. However, the speedup ratios of MDC over the state-of-the-art kDC are 14.14 and $\geq 10^4$, respectively, under the same $k$ settings. These experimental results strongly indicate the excellent practical efficiency of our algorithms.

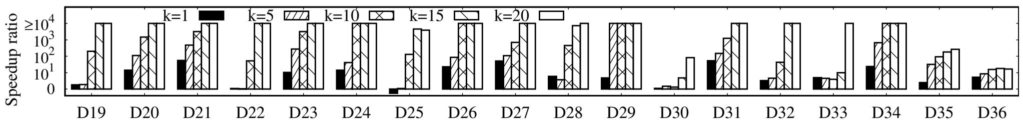
Moreover, from the results presented in Table 3 and Table 4, we observe a significant dependence of our algorithm MDC's runtime on both the graph density and the discrepancy between $\delta$ and $Clq$. Specifically, on real-world graphs where the density is not particularly high and the difference between $\delta$ and $Clq$ is relatively small, our algorithm exhibits exceptional performance. For example, on dataset sc-pkustk13 with the difference of 5 between $\delta$ and $Clq$, our algorithm can identify the maximum defective clique within just 0.67 second when $k = 1$, and even with $k = 20$, the time

Table 4. Runtime of MDC on benchmark real-world graphs with a time threshold of 3 hours (in seconds)

| ID | Runtime of MDC when $k$ = | | | | | ID | Runtime of MDC when $k$ = | | | | | ID | Runtime of MDC when $k$ = | | | | | ID | Runtime of MDC when $k$ = | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 5 | 10 | 15 | 20 | | 1 | 5 | 10 | 15 | 20 | | 1 | 5 | 10 | 15 | 20 | | 1 | 5 | 10 | 15 | 20 |
| D1 | 0.02 | 0.05 | 0.52 | 6.03 | 69.26 | D10 | 0.09 | 0.31 | 1.22 | 4.09 | 13.04 | D19 | 0.35 | 0.41 | 0.43 | 0.47 | 1.42 | D28 | 0.02 | 0.04 | 0.05 | 0.45 | 2.98 |
| D2 | 7.57 | 35.12 | 983.52 | 3792.4 | — | D11 | 0.12 | 0.47 | 2.00 | 23.50 | 41.92 | D20 | 0.48 | 0.68 | 0.97 | 2.84 | 14.03 | D29 | 80.5 | 194.92 | 352.8 | 468.1 | 2487.2 |
| D3 | 0.16 | 0.45 | 2.44 | 30.56 | 173.77 | D12 | 0.42 | 0.48 | 0.52 | 1.28 | 5.30 | D21 | 0.12 | 0.13 | 0.25 | 1.19 | 5.53 | D30 | 5.39 | 4.98 | 7.82 | 8.91 | 14.28 |
| D4 | 0.31 | 0.31 | 0.41 | 1.55 | 9.12 | D13 | 7.82 | 9.45 | 11.23 | 35.10 | 135.56 | D22 | 0.53 | 0.58 | 0.58 | 0.88 | 1.94 | D31 | 0.06 | 0.28 | 1.32 | 9.10 | 91.67 |
| D5 | 0.67 | 0.85 | 3.18 | 9.89 | 42.01 | D14 | 6.75 | 7.67 | 9.92 | 42.37 | 409.96 | D23 | 0.61 | 0.68 | 0.58 | 8.05 | 60.57 | D32 | 0.45 | 3.41 | 26.76 | 637.83 | — |
| D6 | 0.98 | 2.89 | 11.24 | 111.72 | 329.97 | D15 | 0.32 | 0.38 | 0.43 | 1.00 | 2.29 | D24 | 0.22 | 0.23 | 0.29 | 0.55 | 2.84 | D33 | 0.13 | 0.15 | 0.20 | 0.27 | 1.09 |
| D7 | 0.03 | 0.03 | 0.28 | 1.47 | 5.80 | D16 | 0.60 | 2.24 | 11.06 | 54.82 | 509.23 | D25 | 0.33 | 0.31 | 0.40 | 0.49 | 1.62 | D34 | 0.02 | 0.03 | 0.09 | 2.04 | 17.42 |
| D8 | 0.07 | 0.13 | 0.32 | 1.50 | 4.04 | D17 | 0.20 | 0.36 | 0.24 | 0.29 | 0.51 | D26 | 39.8 | 46.6 | 120.3 | 621.4 | 9903.9 | D35 | 0.004 | 0.029 | 0.35 | 6.30 | 16.17 |
| D9 | 0.03 | 0.04 | 0.18 | 0.34 | 0.81 | D18 | 0.57 | 0.65 | 0.68 | 0.72 | 0.84 | D27 | 0.61 | 1.08 | 2.54 | 18.69 | 374.71 | D36 | 0.03 | 0.04 | 0.05 | 0.07 | 0.10 |

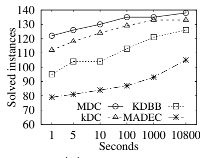


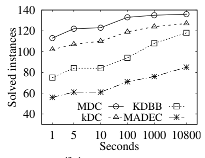
(a) Datasets from D1 to D18

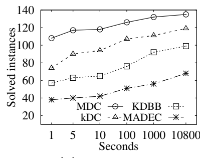


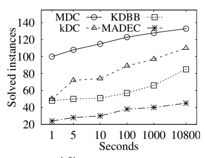
(b) Datasets from D19 to D36

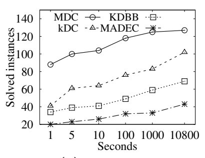
Fig. 5. Speedup ratio of MDC over the best performance among kDC and KDBB

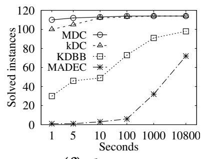

(a) $k = 1$

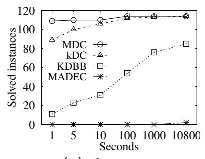

(b) $k = 5$

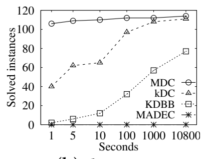

(c) $k = 10$

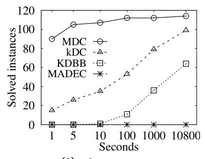

(d) $k = 15$

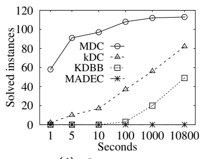

(e) $k = 20$

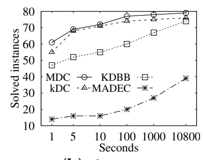

(f) $k = 1$

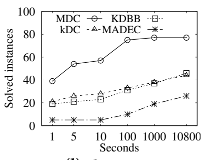

(g) $k = 5$

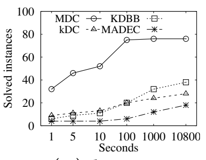

(h) $k = 10$

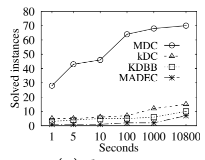

(i) $k = 15$


(j) $k = 20$


(k) $k = 1$


(l) $k = 5$


(m) $k = 10$


(n) $k = 15$

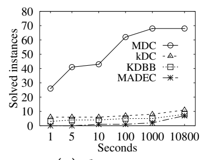
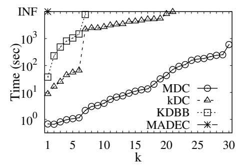

(o) $k = 20$

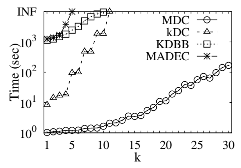
Fig. 6. Number of solved instances of various algorithms on massive graphs with different time thresholds (where sub-figures (a)-(e) are real-world graphs, (f)-(j) are Facebook graphs, and (k)-(o) are DIMACS10 graphs)

consumption remains within 42.01 seconds. This efficiency stems from the ability of our pivot-based technique to minimize the number of sub-branches generated during recursive calls, thereby greatly reducing redundant computations. Furthermore, on these graphs, the size of the maximum $k$-defective clique closely aligns with our proposed advanced color-based upper bound, leading to enhanced pruning capabilities and further improving the overall efficiency of our algorithm.

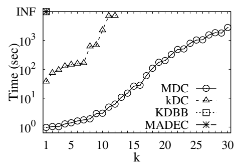
**Exp-2: Solved instance of various algorithms on massive graphs.** In this experiment, we aim to evaluate the performance of our proposed algorithm using a large collection of datasets, which includes 139 real-world graphs, 114 Facebook graphs, and 84 DIMACS10 graphs. Fig. 6



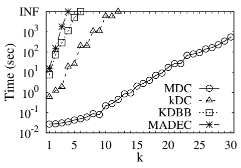
(a) sc-pkustk13



(b) socfb-Texas84

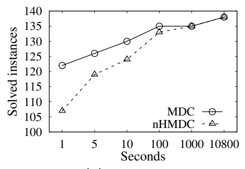


(c) soc-flixster

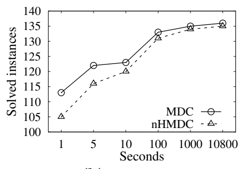


(d) tech-WHOIS

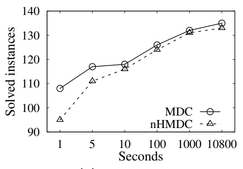
Fig. 7. Runtime of various algorithms with $k$ growing



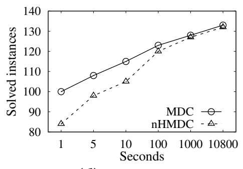
(a) $k = 1$



(b) $k = 5$



(c) $k = 10$



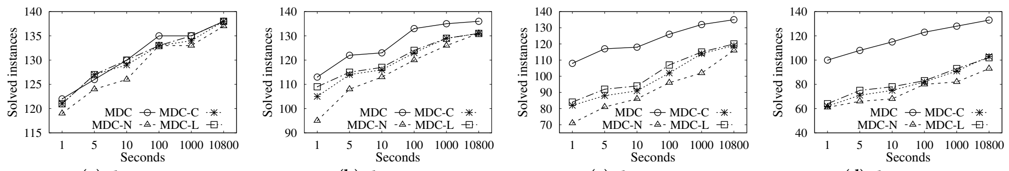
(d) $k = 15$

Fig. 8. Efficiency of MDC without the heuristic approach on 139 real-world graphs

showcases the number of solved instances with varying time thresholds for different values of $k$. From this figure, we observe that our algorithm MDC consistently outperforms the state-of-the-art algorithms, namely kDC, KDBB, and MADEC, in terms of the number of solved instances. These results strongly validate the efficiency of our proposed algorithm in efficiently identifying the maximum $k$-defective clique of $G$. Furthermore, with increasing values of $k$, we observe that the number of instances solved by MDC exhibits minimal variations, while the existing algorithms such as kDC, KDBB and MADEC experience a significant decrease. For instance, when $k = 1$, MDC, kDC, KDBB, and MADEC successfully solve 135, 129, 113, and 87 instances, respectively, within 100 seconds. However, as the value of $k$ grows to 15, MDC can solve 123 instances, while kDC, KDBB, and MADEC can only handle 89, 57, and 38 instances, respectively, under the same settings. This further emphasizes the efficiency of our proposed techniques in reducing unnecessary computations, even when confronted with larger $k$.

**Exp-3: Runtime of various algorithms with $k$ growing.** In this experiment, we further evaluate the performance trend of each algorithm on 4 representative datasets with $k$ growing. Fig. 7 illustrates the detailed experimental results for all tested algorithms. In cases where the algorithm fails to complete the computation with a time threshold of 3 hours, the runtime is denoted as "INF". Note that similar results can also be obtained from other datasets. As can be seen, our algorithm MDC successfully solves all tested graphs even when $k$ is increased up to 30. However, the state-of-the-arts kDC, KDBB, and MADEC are unable to complete the computations within a time threshold of 3 hours on most parameter settings. Furthermore, we observe that the runtime of our algorithm MDC increases smoothly with the increase of $k$, whereas all existing algorithms exhibit sharp increases. For instance, on the sc-pkustk13 dataset, when $k$ takes values of 1, 10, and 20, MDC completes the computations in 0.63, 3.14, and 41.97 seconds, respectively. However, the existing algorithm kDC requires 6.46, 1956.5, and 8153.1 seconds, respectively. This experimental result demonstrates that our algorithm maintains excellent pruning performance even as $k$ grows significantly large.

**Exp-4: Efficiency of the proposed algorithm without the heuristic approach.** We define our algorithm MDC without the heuristic approach as nHMDC, which refers to Algorithm 5 without lines 1-2. In this experiment, we test the effectiveness of algorithm nHMDC in finding the maximum $k$-defective clique of $G$. Fig. 8 illustrates the number of solved instanced of nHMDC and MDC on 139 real-world graphs with different time thresholds when varying $k$. From the results, we note that MDC consistently solves more instances than nHMDC. This disparity arises because the



(a) $k = 1$ (b) $k = 5$ (c) $k = 10$ (d) $k = 15$

Fig. 9. Efficiency of MDC with various upper bounds on 139 real-world graphs

proposed heuristic algorithm can efficiently identify a near-maximum $k$-defective clique, enabling significant pruning of vertices based on the obtained result. Consequently, the efficiency of our algorithm is substantially improved. Nevertheless, it is worth noting that nHMDC still outperforms the existing solutions KDBB and MADEC in most parameter settings, reinforcing the effectiveness of the proposed branching rules in identifying the maximum $k$-defective clique in graph $G$.

**Exp-5: Efficiency of proposed upper bound techniques.** In this experiment, we test the effectiveness of the proposed upper bounds. We designate MDC-N as Algorithm 5 without any upper bounds, and let MDC-C and MDC-L be Algorithm 5 augmented with core-based (Lemma 5) and color-based (Lemma 6) upper bounds, respectively. We conduct experiments on a set of 139 real-world graphs, as described in Fig. 9. As can be seen, we note that MDC consistently demonstrates superior performance compared to all other tested algorithms. This can be attributed to the tightness of the proposed advanced color-based upper bound and the algorithm's efficiency in computing our proposed upper bound. Additionally, we observe that within a given runtime threshold, the gap in number of solved instances between the algorithm MDC and the other algorithms (MDC-N, MDC-C, and MDC-L) increases as $k$ grows. For instance, with a time threshold of 100 seconds, when $k = 1$, MDC solves 135 instances, while both MDC-N, MDC-C, and MDC-L solve 133 instances each. However, when $k$ grows to 10, MDC solves 126 instances, whereas MDC-N, MDC-C, and MDC-L only manage to solve 96, 102, and 107 instances, respectively. This experiment highlights the efficiency of the proposed upper bound pruning technique.

**Exp-6: Efficiency of proposed pivoting rules.** This experiment aims to evaluate the performance of the proposed pivoting rules. Let us denote MDC-R the algorithm MDC without the pivot-based techniques developed in Sec. 3.1 (i.e., the procedure *branch* without lines 15-29). Table 5 presents the experimental results obtained by executing MDC and MDC-R on 6 representative real-world graphs with varying $k$, and the results on other datasets are consistent. From this figure, we note that the runtime of MDC consistently outperforms that of MDC-R across all tested datasets. Moreover, as $k$ increases, the performance of MDC-R experiences a significant decline compared to MDC. These findings confirm that our proposed pivoting rules is indeed efficient in reducing unnecessary branches of the search algorithm.

**Exp-7: Results on different synthetic graphs.** To further investigate the relationship between the efficiency of our proposed algorithm MDC and graph properties, we employed the graph analysis tool NetworkX (https://networkx.org/) to generate a diverse set of synthetic graphs with varying degree distributions and densities. Subsequently, we evaluated the runtime of various algorithms on these synthetic graphs. Table 6 showcases the runtime of MDC for different values of $k$ and $\rho$ on synthetic graphs generated using uniform and power-law distributions, where $\rho = \frac{2m}{n*(n-1)}$ denotes the graph density. For brevity, we exclude the results obtained from synthetic graphs generated by binomial and Gaussian distributions, as the insights derived from these graphs align consistently with those derived from graphs with uniform or power-law distributions. From Table 6, it becomes apparent that the performance of our algorithm varies notably on synthetic graphs generated by uniform distributions as $k$ increases, whereas it consistently exhibits strong

Table 5. Runtime of MDC without pivoting rules (in seconds)

| Datasets | $k=1$ | | $k=5$ | | $k=10$ | | $k=15$ | |
|---|---|---|---|---|---|---|---|---|
| | MDC | MDC-R | MDC | MDC-R | MDC | MDC-R | MDC | MDC-R |
| sc-nasasrb | **0.16** | 0.23 | **0.45** | 0.66 | **2.44** | 3.16 | **30.56** | 38.92 |
| socfb-Duke14 | **0.60** | 1.53 | **2.24** | 39.59 | **11.06** | 412.53 | **54.82** | 1460.9 |
| socfb-Texas84 | **0.48** | 0.55 | **0.68** | 1.45 | **0.97** | 20.66 | **2.84** | 68.3 |
| soc-digg | **39.8** | 50.01 | **46.6** | 76.29 | **120.3** | 1622.7 | **621.4** | — |
| soc-flixster | **0.61** | 0.59 | **1.08** | 7.91 | **2.54** | 139.57 | **18.69** | 1177.4 |
| tech-WHOIS | **0.02** | 0.02 | **0.03** | 0.07 | **0.09** | 4.88 | **2.04** | 130.3 |

Table 6. Runtime of MDC on synthetic graphs (in seconds)

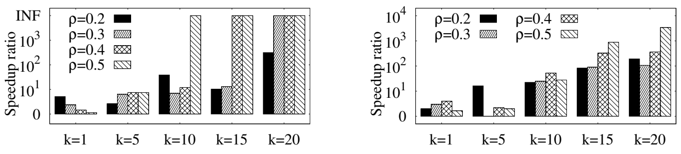
| Distribution | $\rho$ | $\delta$ | $Clq$ | $k=1$ | | $k=5$ | | $k=10$ | | $k=15$ | | $k=20$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $\kappa$ | MDC | $\kappa$ | MDC | $\kappa$ | MDC | $\kappa$ | MDC | $\kappa$ | MDC |
| Uniform ($n = 200$) | 0.2 | 40 | 5 | 6 | 0.07 | 8 | 4.10 | 9 | 101.88 | 10 | 324.74 | 11 | 307.63 |
| | 0.3 | 60 | 7 | 8 | 0.43 | 9 | 42.75 | 11 | 428.24 | 12 | 3356.3 | — | — |
| | 0.4 | 80 | 9 | 9 | 1.65 | 11 | 159.49 | 13 | 2017.0 | — | — | — | — |
| | 0.5 | 100 | 10 | 11 | 8.68 | 13 | 1011.3 | — | — | — | — | — | — |
| Power-law ($n = 200$) | 0.2 | 24 | 12 | 13 | 0.001 | 15 | 0.007 | 17 | 0.28 | 18 | 1.43 | 20 | 11.95 |
| | 0.3 | 38 | 19 | 20 | 0.001 | 23 | 0.005 | 24 | 0.16 | 25 | 1.42 | 26 | 7.07 |
| | 0.4 | 56 | 27 | 28 | 0.001 | 31 | 0.005 | 33 | 0.06 | 34 | 0.49 | 35 | 2.63 |
| | 0.5 | 64 | 44 | 45 | 0.003 | 47 | 0.003 | 48 | 0.01 | 50 | 0.13 | 51 | 0.84 |

performance across synthetic graphs generated by power-law distributions, regardless of changes in $k$ or $\rho$. This discrepancy in performance can be attributed to the significant difference between $\delta$ and $Clq$ observed on uniformly distributed synthetic graphs, whereas this dissimilarity is less pronounced in power-law distributed synthetic graphs. The pronounced difference between $\delta$ and $Clq$ poses challenges for our proposed pivot-based technique, particularly in reducing the number of sub-branches generated during recursive calls, consequently diminishing the pruning efficiency of our algorithm. Moreover, given that real-world networks predominantly follow power-law distributions, these findings also confirm the results obtained in Exp-1.

In addition, we also assess the speedup achieved by MDC over the best performance among existing solutions, namely kDC, KDBB, and MADEC, the results of which are depicted in Fig. 10. It is noteworthy that if all algorithms fail to complete computations within 3 hours, we denote the speedup ratio of MDC over the state-of-the-art algorithm as "INF". As observed, our algorithm consistently surpasses the state-of-the-art solution across most synthetic graphs, even in scenarios where processing proves challenging for our algorithm. For instance, on a uniform distributed synthetic graph with $k = 5$ and $\rho = 0.4$, MDC completes computation in merely 159.49 seconds, exhibiting a speedup of 7.49 times compared to the state-of-the-art algorithm kDC. These findings underscore the efficiency of our algorithm in comparison to existing solutions for identifying the maximum $k$-defective clique of $G$ across diverse graph structures.

## 6 Related Works

**Maximum clique search.** The problem of identifying the maximum clique in a graph $G$ has been proven to be NP-hard [8], with approximating a satisfactory solution being a challenging task [65]. Over the past few decades, numerous exact algorithms have been developed to tackle this problem [10, 27–29, 41, 49, 50, 53–55]. Most of these algorithms are built upon a branch-and-bound framework, and employing various enumeration strategies to find the maximum clique. Among the existing approaches, those proposed by Tomita et al. [53–55] and Li et al. [27–29] have gained significant popularity. Specifically, Tomita et al. [53] introduced an algorithm based on degeneracy ordering and further improved it using the graph recoloring technique [55] and the adjunct coloring-based ordering [54]. On the other hand, Li et al. proposed a branch-and-bound algorithm based on MaxSAT reasoning [29], and further improving it with incremental upper bounds [28] and dynamic and static vertex ordering strategies [27]. Additionally, several parallel approaches utilizing multi-cores have also been developed to enhance practical efficiency [49, 50]. However, the aforementioned upper bound techniques based on coloring and MaxSAT reasoning face challenges when extended to solve the problem of finding the maximum $k$-defective clique, wherein the subgraph allows at most $k$ missing edges. To address this issue, this paper introduces a novel upper bound approach based on graph coloring, which significantly differs from existing color-based upper bounds.



(a) Uniform graphs ($n = 200$)  (b) Power-law graphs ($n = 200$)

Fig. 10. Speedup ratio of MDC over the best performance between kDC, KDBB, and MADEC on synthetic graphs

**Maximum relaxed-clique search.** Since the clique model is often too restrictive for many real-world applications, several relaxed-clique models have also been developed to address this limitation [45]. These include the $k$-plex [52], $s$-clique [45], and $\gamma$-quasi-clique [31], and others. Among these models, significant attention has been given to the problems of finding the maximum $k$-plex [3, 12, 18, 22, 35, 38, 61, 64] and maximum $\gamma$-quasi-clique [34, 36, 40, 42, 43, 47, 58] in recent years. Regarding the maximum $k$-plex problem, Balasundaram et al. [3] proposed an integer programming formulation and a branch-and-cut algorithm. McClosky et al. [35] introduced a combinatorial algorithm based on co-$k$-plex coloring as an upper bound technique. Xiao et al. [61] developed an exact algorithm with a time complexity of $O(P(n)\alpha^n)$, employing a symmetric branching rule, where $\alpha < 2$. More recently, several novel techniques have also been developed to further improve the efficiency, including dynamic vertex selection strategies [18], second-order reduction and coloring-based upper bounds [64], partition-based upper bounds [22], and exploiting small dense subgraphs [12]. Concerning the maximum $\gamma$-quasi-clique problem, Pattillo et al. [43] and Veremyev et al. [58] formulated the problem using integer programming. Pajouh et al. [40], Pastukhov et al. [42], and Ribeiro et al. [47] respectively developed branch-and-bound algorithms incorporating various pruning techniques. Additionally, upper bounds on the maximum $\gamma$-quasi-clique number have been further explored in [34, 36]. Unfortunately, all these existing algorithms still face challenges when efficiently extended to solve the problem of finding the maximum $k$-defective clique. In this paper, we propose a theoretically and practically efficient solution to find the maximum $k$-defective clique of a given graph $G$.

## 7 Conclusion

In this paper, we focus on the problem of identifying the maximum $k$-defective clique of a given graph $G$. To address this problem, we propose an efficient search algorithm accompanied by a series of novel optimization techniques. Specifically, our algorithm leverages newly-developed graph reduction rules and a pivot-based branching technique. Our analysis demonstrates that the proposed algorithm achieves a time complexity of $O(m\gamma_k^n)$, where $\gamma_k$ is a real value strictly less than 2. To further enhance efficiency, we also present an efficient pruning algorithm based on several carefully-designed upper bounding techniques. Moreover, we make additional improvements to our algorithm by incorporating an ordering-based heuristic algorithm as the preprocessing step. Finally, we conduct comprehensive experiments to validate the effectiveness and efficiency of our proposed approaches, and the results demonstrate that our algorithm achieves a speedup of 3 orders of magnitude over the state-of-the-art solutions.

## Acknowledgments

This work was partially supported by (i) the National Science and Technology Major Project 2020AAA0108503, (ii) NSFC Grants U2241211 and 62072034, (iii) the China National Postdoctoral Program for Innovative Talents BX20240467, and (iv) the China Postdoctoral Science Foundation 2023M740245. Rong-Hua Li is the corresponding author of this paper.

## References

[1] Eytan Adar and Christopher Ré. 2007. Managing Uncertainty in Social Networks. *IEEE Data Eng. Bull.* 30, 2 (2007), 15–22.

[2] Charu C. Aggarwal. 2011. *Social Network Data Analytics.* Springer.

[3] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. 2011. Clique Relaxations in Social Network Analysis: The Maximum $k$-Plex Problem. *Oper. Res.* 59, 1 (2011), 133–142.

[4] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. *CoRR* cs.DS/0310049 (2003).

[5] Punam Bedi and Chhavi Sharma. 2016. Community detection in social networks. *Wiley interdisciplinary reviews: Data mining and knowledge discovery* 6, 3 (2016), 115–135.

[6] Vladimir Boginski, Sergiy Butenko, and Panos M Pardalos. 2005. Statistical analysis of financial networks. *Computational statistics & data analysis* 48, 2 (2005), 431–443.

[7] Vladimir Boginski, Sergiy Butenko, and Panos M Pardalos. 2006. Mining market data: A network approach. *Computers & Operations Research* 33, 11 (2006), 3171–3184.

[8] Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. 1999. The Maximum Clique Problem. In *Handbook of Combinatorial Optimization.* 1–74.

[9] Coenraad Bron and Joep Kerbosch. 1973. Finding All Cliques of an Undirected Graph (Algorithm 457). *Commun. ACM* 16, 9 (1973), 575–576.

[10] Lijun Chang. 2020. Efficient maximum clique computation and enumeration over large sparse graphs. *VLDB J.* 29, 5 (2020), 999–1022.

[11] Lijun Chang. 2023. Efficient Maximum k-Defective Clique Computation with Improved Time Complexity. *Proc. ACM Manag. Data* 1, 3 (2023), 209:1–209:26.

[12] Lijun Chang, Mouyi Xu, and Darren Strash. 2022. Efficient Maximum k-Plex Computation over Large Sparse Graphs. *Proc. VLDB Endow.* 16, 2 (2022), 127–139.

[13] 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.

[14] 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.

[15] Nan Du, Bin Wu, Xin Pei, Bai Wang, and Liutong Xu. 2007. Community detection in large-scale social networks. In *WebKDD and SNA-KDD workshop.* 16–25.

[16] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In *ISAAC*, Vol. 6506. 403–414.

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

[18] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. 2018. An Exact Algorithm for Maximum k-Plexes in Massive Graphs. In *IJCAI.* 1449–1455.

[19] 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.

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

[21] Shweta Jain and C. Seshadhri. 2020. Provably and Efficiently Approximating Near-cliques using the Turán Shadow: PEANUTS. In *WWW.* 1966–1976.

[22] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. 2021. A New Upper Bound Based on Vertex Partitioning for the Maximum K-plex Problem. In *IJCAI.* 1689–1696.

[23] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations.* 85–103.

[24] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Dandapani Sivakumar, Andrew Tompkins, and Eli Upfal. 2000. The Web as a graph. In *PODS.* 1–10.

[25] 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.

[26] R. M. R. Lewis. 2016. *A Guide to Graph Colouring - Algorithms and Applications.* Springer.

[27] Chu-Min Li, Hua Jiang, and Felip Manyà. 2017. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Comput. Oper. Res.* 84 (2017), 1–15.

[28] Chu-Min Li, Zhiwen Fang, and Ke Xu. 2013. Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In *ICTAI.* 939–946.

[29] Chu Min Li and Zhe Quan. 2010. An Efficient Branch-and-Bound Algorithm Based on MaxSAT for the Maximum Clique Problem. In *AAAI.* 128–133.

[30] Don R Lick and Arthur T White. 1970. k-Degenerate graphs. *Canadian Journal of Mathematics* 22, 5 (1970), 1082–1096.

[31] Guimei Liu and Limsoon Wong. 2008. Effective Pruning Techniques for Mining Quasi-Cliques. In *ECML/PKDD*, Vol. 5212. 33–49.

[32] R. Duncan Luce. 1950. Connectivity and generalized cliques in sociometric group structure. *Psychometrika* 15, 2 (1950), 169–190.

[33] R. Duncan Luce and Albert D. Perry. 1949. A method of matrix analysis of group structure. *Psychometrika* 14, 2 (1949), 95–116.

[34] Fabrizio Marinelli, Andrea Pizzuti, and Fabrizio Rossi. 2021. LP-based dual bounds for the maximum quasi-clique problem. *Discret. Appl. Math.* 296 (2021), 118–140.

[35] Benjamin McClosky and Illya V. Hicks. 2012. Combinatorial algorithms for the maximum k-plex problem. *J. Comb. Optim.* 23, 1 (2012), 29–49.

[36] Zhuqi Miao and Balabhaskar Balasundaram. 2020. An Ellipsoidal Bounding Scheme for the Quasi-Clique Number of a Graph. *INFORMS J. Comput.* 32, 3 (2020), 763–778.

[37] Robert J. Mokken et al. 1979. Cliques, clubs and clans. *Quality & Quantity* 13, 2 (1979), 161–173.

[38] Hannes Moser, Rolf Niedermeier, and Manuel Sorge. 2012. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *J. Comb. Optim.* 24, 3 (2012), 347–373.

[39] Kevin A. Naudé. 2016. Refined pivot selection for maximal clique enumeration in graphs. *Theor. Comput. Sci.* 613 (2016), 28–37.

[40] Foad Mahdavi Pajouh, Zhuqi Miao, and Balabhaskar Balasundaram. 2014. A branch-and-bound approach for maximum quasi-cliques. *Ann. Oper. Res.* 216, 1 (2014), 145–161.

[41] Panos M Pardalos and Jue Xue. 1994. The maximum clique problem. *Journal of global Optimization* 4 (1994), 301–328.

[42] Grigory Pastukhov, Alexander Veremyev, Vladimir Boginski, and Oleg A. Prokopyev. 2018. On maximum degree-based $\gamma$-quasi-clique problem: Complexity and exact approaches. *Networks* 71, 2 (2018), 136–152.

[43] Jeffrey Pattillo, Alexander Veremyev, Sergiy Butenko, and Vladimir Boginski. 2013. On the maximum quasi-clique problem. *Discret. Appl. Math.* 161, 1-2 (2013), 244–257.

[44] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2011. Clique relaxation models in social network analysis. In *Handbook of Optimization in Complex Networks: Communication and Social Networks*. Springer, 143–162.

[45] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2013. On clique relaxation models in network analysis. *Eur. J. Oper. Res.* 226, 1 (2013), 9–18.

[46] Eric M. Phizicky and Stanley Fields. 1995. Protein-protein interactions: methods for detection and analysis. *Microbiological reviews* 59, 1 (1995), 94–123.

[47] Celso C. Ribeiro and José A. Riveaux. 2019. An exact algorithm for the maximum quasi-clique problem. *Int. Trans. Oper. Res.* 26, 6 (2019), 2199–2229.

[48] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. 4292–4293.

[49] Ryan A. Rossi, David F. Gleich, and Assefaw Hadish Gebremedhin. 2015. Parallel Maximum Clique Algorithms with Applications to Network Analysis. *SIAM J. Sci. Comput.* 37, 5 (2015).

[50] Pablo San Segundo, Alvaro Lopez, and Panos M. Pardalos. 2016. A new exact maximum clique algorithm for large and massive sparse graphs. *Comput. Oper. Res.* 66 (2016), 81–94.

[51] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

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

[53] Etsuji Tomita and Toshikatsu Kameda. 2007. An Efficient Branch-and-bound Algorithm for Finding a Maximum Clique with Computational Experiments. *J. Glob. Optim.* 37, 1 (2007), 95–111.

[54] Etsuji Tomita, Sora Matsuzaki, Atsuki Nagao, Hiro Ito, and Mitsuo Wakatsuki. 2017. A Much Faster Algorithm for Finding a Maximum Clique with Computational Experiments. *J. Inf. Process.* 25 (2017), 667–677.

[55] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. 2010. A Simple and Faster Branch-and-Bound Algorithm for Finding a Maximum Clique. In *WALCOM*, Vol. 5942. 191–203.

[56] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.* 363, 1 (2006), 28–42.

[57] 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.

[58] Alexander Veremyev, Oleg A. Prokopyev, Sergiy Butenko, and Eduardo L. Pasiliao. 2016. Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs. *Comput. Optim. Appl.* 64, 1 (2016), 177–214.

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

[60] Jianxin Wang, Min Li, Youping Deng, and Yi Pan. 2010. Recent advances in clustering methods for protein interaction networks. *BMC genomics* 11, 3 (2010), 1–19.

[61] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. 2017. A Fast Algorithm to Compute Maximum $k$-Plexes in Social Network Analysis. In *AAAI*. 919–925.

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

[63] 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.

[64] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. 2021. Improving Maximum k-plex Solver via Second-Order Reduction and Graph Color Bounding. In *AAAI*. 12453–12460.

[65] David Zuckerman. 2006. Linear degree extractors and the inapproximability of max clique and chromatic number. In *STOC*. 681–690.

Received January 2024; revised April 2024; accepted May 2024