Theoretically and Practically Efficient Maximum Defective Clique Search

ABSTRACT

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 γ_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 at least 3 orders of magnitude faster than the state-of-the-art solutions on most of real-world graphs.

1 INTRODUCTION

Graph has emerged as a versatile model for representing diverse real-world networks, including social networks [2], web networks [25], 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 statistical analysis in financial networks [6, 7] all can be formulated as cohesive subgraph mining problems. Perhaps, the classical clique [34], 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, 40, 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 [32], r-clique [33], k-club [38], 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 [20, 22] and statistical analysis in financial networks [14], as well as its close relationship with other cohesive subgraph models, such as the clique [34] 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, 21, 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, 21, 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. [21] improve this algorithm by employing new preprocessing methods and an optimized implementation. On the other hand, several new algorithms [11, 13, 19], based on a branch-and-bound search technique [26], 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 for the branch-and-bound process. Significantly, 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 α_k is a real-number less than 2. To further improve efficiency, Gao et al. [19] present an improved branch-and-bound algorithm based on several new branch pruning techniques. More recently, a significantly 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 β_k is a real-number smaller than α_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

SIGMOD '25, —, 2025, Berlin, Germany

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

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 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, 21, 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 α_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 α_k are 1.839 and 1.928, respectively. Consequently, there is an urgent demand for developing more efficient algorithms that can effectively identify the maximum k-defective clique of real-world graphs.

Contribution. 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 to address aforementioned issues. 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 γ_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 γ_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 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 at least 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://anonymous.4open.science/r/MaximumDefectiveClique-AB7C.

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**	...

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 $\overline{d}_v(S)$ ($\overline{d}_v(S)$), 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 non-negative integer k, the goal of this paper is to compute the maximum k-defective clique of G, i.e., finding a k-defective clique whose size is the largest among all maximal k-defective cliques of G.

As analyzed in Sec. 1, existing solutions [11, 13, 19, 21, 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 [26], which centers around dividing the current problem into smaller subproblems. 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. Notably, 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 exists a vertex in C that has at most three non-neighbors within $S \cup C$, denoted as v with $\overline{d}_v(S \cup C) \leq 3$, 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 $\overline{d}_v(S \cup C) = 1$, the maximum k-defective clique for instance I must be included 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. Let u be the non-neighbor of v within $S \cup C$. Assume that S^* is a maximum k-defective clique for instance I. It can be easily verified that $|S^* \cap \{v, u\}| \geq 1$, since u is the only non-neighbor of v in $S \cup C$. Next, we demonstrate the existence of a k-defective clique containing v, with a size no less than $|S^*|$, in the scenario where $v \notin S^*$. If $u \in C$, we can establish that $S^* \setminus \{u\} \cup \{v\}$ also forms a maximum k-defective clique for instance I. Conversely, if $u \in S$, we have $(S^* \setminus S) \subseteq N_v(G)$. Let w be a vertex in $S^* \setminus S$ with the minimum value of $d_w(S^*)$. If $d_w(S^*) = |S^*| - 1$, it follows that S^* represents a $(k-1)$-defective clique, as the vertex v with $(u, v) \notin E$ can be utilized to expand S. Therefore, if $d_w(S^*) = |S^*| - 1$, S^* must contain v. Alternatively, if $d_w(S^*) \leq |S^*| - 2$, then $S^* \setminus \{w\} \cup \{v\}$ forms a maximum k-defective clique for instance I. By combining the aforementioned analysis, we can conclude that the sub-instance $I' = (G, S \cup \{v\}, C \setminus \{v\}, k)$ can identify a k-defective clique with a size no less than $|S^*|$. □

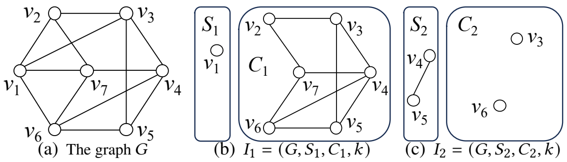
Based on Lemma 1 and Lemma 2, we can derive that for a given instance $I = (G, S, C, k)$, if there exists a vertex v in C such that $\overline{d}_v(S \cup C) \leq 2$, then it is sufficient to consider only the scenario where the maximum k-defective clique of I includes vertex v.

(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. Here we first prove the correctness for the case where $\overline{d}_v(S) = 0$. It can be easily verified that S^* must contain either vertex v, or both vertices u and w. This is because a k-defective clique with a size no less than $|S^*|$ can also be generated by sub-instance I_1 if S^* includes only one vertex from $\{u, w\}$. Next, we consider the scenario where $\{u, w\} \subseteq S^*$ and $\overline{d}_v(S) = 0$. We observe that if $\overline{d}_u(S) \geq 1$ (or $\overline{d}_w(S) \geq 1$) or $(u, w) \notin E$, then $S^* \setminus \{u\}$ (or $S^* \setminus \{w\}$)



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

Figure 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 ($k \geq 2$).

will form a $(k-1)$-defective clique in G, which can be expanded by vertex v. This implies that sub-instance I_1 can obtain a maximum k-defective clique with a size no less than $|S^*|$ if $(u, w) \notin E$ or $\overline{d}_u(S) + \overline{d}_w(S) \geq 1$ for the case where $\overline{d}_v(S) = 0$. However, if the conditions $\overline{d}_u(S) + \overline{d}_w(S) \geq 1$ and $(u, w) \notin E$ are not satisfied, the sub-instance $I_2 = (G, S \cup \{u, w\}, C \setminus \{u, w\}, k)$ is further invoked. In this scenario, we note that it is sufficient to use the common neighbors of u and w in C to further expand $S \cup \{u, w\}$. This is because if there exists a vertex v' in $C \setminus N_u(G)$ (or $C \setminus N_w(G)$) that belongs to S^*, it can be easily verified that $S^* \setminus \{u\}$ (or $S^* \setminus \{w\}$) forms a $(k-1)$-defective clique in G. Consequently, $S^* \setminus \{u\} \cup \{v\}$ (or $S^* \setminus \{w\} \cup \{v\}$) is also a maximum k-defective clique, obtainable through sub-instance I_1. Hence, the correctness has been established for the case where $\overline{d}_v(S) = 0$. The case where $\overline{d}_v(S) = 1$ can be proven similarly. Therefore, this lemma establishes. □

The following example illustrates the idea of Lemma 3.

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 $C \setminus \{v\} \setminus N_v(G)$.*

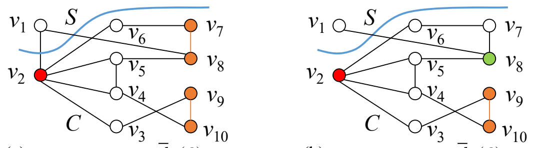
Proof. When $\overline{d}_v(S) = 0$, this theorem is clearly established [14]. Hence, we shall focus only on the case where $\overline{d}_v(S) = 1$. Given that $S \cup \{v\}$ forms a k-defective clique, it follows that the number of missing edges in $G(S)$ is less than k. Denote by $C_1 = C \cap N_v(G)$. Notably, this theorem disregards the maximum k-defective clique contained in $G(S \cup C_1)$. Assuming $S \cup D$ represents the maximum k-defective clique in $G(S \cup C_1)$, where $D \subseteq C_1$, we now demonstrate that the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ of I can generate a k-defective clique with a size no less than $|S \cup D|$. If each vertex u in D satisfies $\overline{d}_u(S \cup D) = 1$, then the number of missing edges in $G(S \cup D)$ is less than k. Consequently, $S \cup D \cup \{v\}$ is a larger k-defective clique in $G(S \cup C)$. On the other hand, if there exists a vertex u in D for which $\overline{d}_u(S \cup D) \geq 2$, considering the condition $\overline{d}_v(S \cup D) = 1$, we observe that $S \cup D \setminus \{u\} \cup \{v\}$ also constitutes a k-defective clique of $G(S \cup C)$, detectable through I_1. Thus, the maximum k-defective clique of the sub-instance I either contains v or a vertex in $C \setminus \{v\} \setminus N_v(G)$. This completes the proof. □

Although Theorem 3.1 effectively reduces redundant subbranches 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.*

- *Case $\overline{d}_v(S) = 0$: the maximum k-defective clique for instance I either contains v or an edge in $G(P)$.*
- *Case $\overline{d}_v(S) = 1$: 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. Let S^* denote the maximum k-defective clique that contains S in $G(S \cup C)$, and let $D = S^* \cap P$ be the subset of vertices in C that are also non-neighbors of v. We will now demonstrate the correctness of case (1). Suppose, on the contrary, that D forms an independent set if $v \notin S^*$. It can be easily verified that $D \neq \emptyset$, otherwise $S^* \cup \{v\}$ would form a larger k-defective clique of G based on the fact that $\overline{d}_v(S) = 0$. Consider a randomly chosen vertex u from D. Since D is an independent set, we can derive that $S^* \setminus \{u\}$ forms a $(k + 1 - |D|)$-defective clique in $G(S \cup C)$. Moreover, based on the condition $(S^* \setminus D) \subseteq N_v(G)$, we can conclude that $S^* \setminus \{u\} \cup \{v\}$



(a) Pivot vertex v with $\overline{d}_v(S) = 0$ (b) Pivot vertex v with $\overline{d}_v(S) = 1$

Figure 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 ($k \geq 1$).

must be also a k-defective clique in $G(S \cup C)$, detectable through the sub-instance $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$. Therefore, if $v \notin S^*$, there must exist at least one edge in $G(D)$ for case (1). Regarding case (2), we already know from Theorem 3.1 that S^* must contain a vertex in $\{v\} \cup P$. It is sufficient to demonstrate that when $S^* \cap (\{v\} \cup P_1) = \emptyset$, there must exist at least one edge in $G(D)$. Employing a similar analytical approach as in the previous case, the correctness of case (2) also establishes. As a result, this theorem is established. □

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

Based on the proposed branch reduction rules and pivoting rules, we develop a novel branching rule for efficiently finding the maximum k-defective clique in graph G, which is 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.
- Else 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.

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.

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       $C \leftarrow C \setminus \{u\}$;

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 larger than $|C| - k + \overline{d}(S)$ 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 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 is necessary 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

Algorithm 2: Update(S, C, v)

```
1 C' ← ∅; s ← the number of missing edges in G(S);
2 for u ∈ C, s.t. u ≠ v do
3     d̄ ← s + d̄_v(S) + d̄_u(S);
4     if d̄ ≤ k then
5         if u ∈ N_v(G) then C' ← C' ∪ {u};
6         else if d̄ < k then C' ← C' ∪ {u};
7 return C';
```

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 $O(n)$.

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 γ_k is the maximum real root of $x^{k+3} - 2x^{k+2} + x^2 - x + 1 = 0$ if $k \geq 1$. Specifically, when $k = 1, 2$ and 3, the corresponding values of γ_k are 1.466, 1.755, and 1.889, respectively.*

PROOF. Let $T(n)$ be the total number of leaves generated by the $branch(S, C)$ procedure outlined in Algorithm 1. We can establish that the time complexity of Algorithm 1 is bounded by $O(mT(n))$, as each recursive call of $branch$ requires at most $O(m)$ time. Now, we analyze the size of $T(n)$ by considering the following cases.

(1) If $\exists v \in C$ with $\overline{d}_v(S \cup C) \leq 3$ and $\overline{d}_v(S) \leq 1$, the algorithm employs the branch reduction rules outlined in Sec. 3.1 to identify the maximum k-defective clique of G. When $\overline{d}_v(S \cup C) \leq 2$, the following recurrence can be easily obtained:
$$T(n) \leq T(n-1). \tag{1}$$
However, if $\overline{d}_v(S \cup C) > 2$, the branch reduction rule presented in Lemma 3 is utilized. Let u and w be the two non-neighbors of v in $S \cup C$. If $\overline{d}_v(S) = 0$, the algorithm, in the worst-case, invokes two sub-branches $Branch(S \cup \{v\}, C \setminus \{v\})$ and $Branch(S \cup \{u, w\}, C \cap N_u(G) \cap N_w(G))$ to find the maximum k-defective clique. Considering the conditions $\overline{d}_u(S) + \overline{d}_w(S) = 0$, $\overline{d}_u(S \cup C) \geq 3$ and $\overline{d}_w(S \cup C) \geq 3$, we can conclude that $|C \cap N_u(G) \cap N_w(G)| \leq |C| - 4$. Thus, we have the recurrence relation:
$$T(n) \leq T(n-1) + T(n-4). \tag{2}$$
In the case where $\overline{d}_v(S) = 1$, the algorithm invoke sub-branches $Branch(S \cup \{v\}, C \setminus \{v\})$ and $Branch(S \cup \{u\}, C \cap N_u(G))$ to find the maximum k-defective clique in the worst-case. Since $|C \cap N_u(G)| \leq |C| - 3$, we have the recurrence relation:
$$T(n) \leq T(n-1) + T(n-3). \tag{3}$$
Hence, Eq. (3) represents the worst-case recurrence for the branch reduction rules.

(2) If $|C \setminus N_S(C)| + \overline{d}(S) \geq k$, our algorithm selects a vertex v in $C \setminus N_S(C)$ to perform the branch-and-bound, which leads to the following recurrence relation:
$$T(n) \leq T(n-1) + T(n-1). \tag{4}$$
It is important to note that if every vertex in $C \setminus \{v\}$ can be further used to expand $S \cup \{v\}$, then another vertex from $C \setminus N_S(C)$ will be selected to perform the branch-and-bound procedure in $branch(S \cup \{v\}, C \setminus \{v\})$. This recursive process continues until there are k missing edges in the current k-defective clique. When $\overline{d}_v(S) \geq 2$, it can be derived that at most $k - \overline{d}(S) - 1$ vertices in $C \setminus N_S(C)$ can be added into S. Initially, when $\overline{d}(S) = 0$ and $|C \setminus N_S(C)| \geq k$, the following recurrence can be obtained:
$$T(n) \leq \sum_{i=1}^{k} T(n-i), \quad \text{where } k \geq 2. \tag{5}$$
On the other hand, if $\overline{d}_v(S) \leq 1$, we can establish that $\overline{d}_v(S \cup C) \geq 4$ based on the branch reduction rules. If there exists a non-neighbor of v in $C \setminus N_S(C)$, the recurrence of Eq. (5) can be obtained accordingly. Otherwise, we can derive that $|C \setminus N_{S \cup \{v\}}(C)| \geq k - \overline{d}(S) + 2$, as $\overline{d}_v(S \cup C) \geq 4$. Hence, we obtain the following recurrence relation:
$$T(n) \leq \sum_{i=1}^{k} T(n-i) + T(n-k-2). \tag{6}$$
Based on the aforementioned analysis, Eq. (6) represents the worst-case recurrence for the case when $|C \setminus N_S(C)| + \overline{d}(S) \geq k$.

(3) If $C_1 = \emptyset$, it implies that each vertex v in C has at least two non-neighbors in S. As a result, there are at most $k/2$ vertices that can be added to S. This leads to the following recurrence:
$$T(n) \leq \sum_{i=1}^{k/2} T(n-i), \quad \text{where } k \geq 2. \tag{7}$$

(4) If $|C \setminus N_S(C)| + \overline{d}(S) < k$, the pivot-based branching rule is invoked. This gives rise to the following recurrence:
$$T(n) \leq \sum_{i=1}^{|P_1|} T(n-i) + \sum_{i=1}^{|P_2|} \sum_{j=1}^{|P_2|-i} T(n - |P_1| - i - j). \tag{8}$$
Building upon Eq. (4), we can derive that $\sum_{j=1}^{|P_2|-i} T(n - |P_1| - i - j) \leq T(n - |P_1| - i)$. Then, the recurrence for Eq. (8) can be improved as follows:
$$T(n) \leq \sum_{i=1}^{|P_1|} T(n-i) + \sum_{i=1}^{|P_2|} T(n - |P_1| - i). \tag{9}$$
To analyze the bound of Eq. (9), we assume $\overline{d}(S) = 0$. Let $\overline{d} = |P_1| + |P_2|$. It is easy to verify that $T(n) \leq \sum_{i=1}^{k} T(n-i)$ if $\overline{d} \leq k$. We now focus on the case where $\overline{d} > k$. If there exists a vertex $u \in \overline{N}_v(C)$ with $\overline{d}_u(S) \geq 2$, the branching rule for case (2) is applied in $Branch(S \cup \{v\}, C \setminus \{v\})$. Based on the definition of the k-defective clique and the condition of $\overline{d}_u(S \cup \{v\}) \geq 3$, it can be observed that at most $k-2$ vertices in $\overline{N}_v(C) \setminus \{v\}$ can be added into $S \cup \{v\}$. As a result, we have:
$$\begin{aligned} T(n) &\leq \sum_{i=2}^{\overline{d}} T(n-i) + \sum_{i=1}^{k-2} T(n-1-i) + T(n-\overline{d}) \\ &\leq \sum_{i=1}^{k-1} T(n-i) + T(n-\overline{d}) < \sum_{i=1}^{k} T(n-i). \end{aligned} \tag{10}$$

If there is no vertex $u \in \overline{N}_v(C)$ with $\overline{d}_u(S) \geq 2$, we can infer that $\overline{d}_u(C) \leq \overline{d}$ for each $u \in \overline{N}_v(G)$. Then, the first $\overline{d} - k$ sub-recursive calls of $Branch(S, C)$ will apply the branching rule of case (2). By combining Eq. (6) with Eq. (11), we obtain:

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

Based on the conditions of $\sum_{j=1}^{k} T(n-i-j) \leq T(n-i)$ and $\sum_{i=1}^{\overline{d}-k} T(n-\overline{d}-2) \leq T(n-k-2)$ if $\overline{d} > k$, we improve Eq. (11) as follows:

$$T(n) \leq \sum_{i=1}^{k} T(n-i) + T(n-k-2). \tag{12}$$

To summary, we establish that the maximum size of $T(n)$ is bounded by Eq. (6) or Eq. (12). By utilizing the theoretical result in [17], we can derive that the maximum size of $T(n)$ is bounded by $O(\gamma_k^n)$, where γ_k is the maximum real-root of function $x^{k+3} - 2x^{k+2} + x^2 - x + 1 = 0$ if $k \geq 1$. Thus, we complete this proof. □

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

PROOF. When $k = 0$, our algorithm employs either the branch reduction rules or the pivot-based branching rule to find the maximum k-defective clique. If the branch reduction rules are utilized, we obtain the recurrence in Eq. (2), as there is no vertex $v \in C$ that satisfies $\overline{d}_v(S) \geq 1$. Alternatively, 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 establish that $T(n) \leq 4T(n-4)$. Thus, when $k = 0$, the size of $T(n)$ is bounded by $O(\sqrt{2}^n)$. This theorem is proven. □

THEOREM 3.5. *The space complexity of Algorithm 1 is $O(\kappa n+m)$, where κ 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 (κn) 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 $O(\kappa n + m)$. □

Discussion. It is worth noting that our proposed branching rule is very different from all existing solutions [11, 13, 19, 21, 57] in identifying the maximum k-defective clique of G. Specifically, the Russian doll based approaches [21, 57] and the KDBB algorithm [19] directly use the classical branch-and-bound technique by randomly selecting a vertex v from the candidate set C to detect the maximum k-defective clique contains v and excludes v, respectively. Obviously, the time complexity of these approaches remains limited to $O(P(n)2^n)$. The MADEC algorithm [13] first develops a new branching rule based on the size of $C \setminus N_S(C)$, which is to generate at most $k+1$ sub-branches in each recursive call for identifying the maximum k-defective clique, as there allows at most k missing edges in the k-defective clique S. With the idea, this approach can achieve a time complexity of $O(P(n)\alpha_k^n)$, where α_k is the real values strictly less than 2. To further improve efficiency, a kDC algorithm [11] is developed, employing a *non-fully-adjacent-first* branching rule. This rule prioritizes the selection of a vertex $v \in C$ that has non-neighbors in S to identify the maximum k-defective clique contains v and excludes v, respectively, in each recursive call. The author shows that this algorithm achieves an improved worst-case time complexity of $O(P(n)\beta_k^n)$, where $\beta_k < \alpha_k$ for each $k \geq 1$. In contrast, we present a pivot-based branching rule for identifying the maximum k-defective clique in G. Our approach begins by selecting a vertex v from C that satisfies $\overline{d}_v(S) \leq 1$. Then, only vertex v and vertices in $\overline{N}_v(C)$ (or edges in $G(\overline{N}_v(G))$) are used to expand S, significantly reducing the unnecessary search space. As we analyzed in Theorem 3.3, the time complexity of our proposed framework is notably lower than that of all existing [11, 13, 19, 21, 57]. Thus, our proposed branching rule fundamentally sets itself apart from these existing solutions.

4 THE PROPOSED SEARCH ALGORITHM

Based on our branch-and-bound framework, we propose a new algorithm with several novel optimization techniques to further enhance the efficiency of our solution. Next, we first develop a new upper bound-based pruning technique and then present our algorithms.

4.1 The Proposed Upper Bounds

Let κ (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 κ 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 κ 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.

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 δ 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 κ 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 ω and $\omega(C)$ the number of distinct colors in G and the subgraph $G(C)$ of G induced by C, respectively. Based on the concept of graph coloring, we can derive the following upper bound for κ 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. Given a k-defective clique S in G, we can partition it into two subsets, namely S_1 and S_2, satisfying $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S$. In this partitioning, we impose the following constraint: 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 is evident that two vertices u and v in G sharing the same color (i.e., $col_v(G) = col_u(G)$) are not connected by an edge ($(u, v) \notin E$). With this observation, we can derive that $|S_2| \leq k$, as exceeding k vertices in S_2 would violate the definition of a k-defective clique. Considering the number of distinct colors ω, it follows that $|S_1| \leq \omega$. Consequently, the size of the maximum k-defective clique in G is bounded by $\omega + k$, thus establishing this lemma. □

Lemma 6 demonstrates that finding a smaller value for ω ($\omega(C)$) can achieve a better upper bound. However, it is worth noting that determining the smallest value of ω ($\omega(C)$) for graph coloring poses a computational challenge and is known to be NP-hard [24]. Thus, numerous heuristic approaches have been explored to address graph coloring problem [27, 55]. In this paper, we adopt a widely used degeneracy ordering heuristic for graph coloring. The definition of degeneracy ordering [31] 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.

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 $G(S \cup D)$.

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** $|S| + |D|$;

PROOF. This lemma can be readily established as each vertex in $D \setminus \{v\}$ that shares the same color as v is not the neighbor of v. □

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. Given a subset D of C, we observe that there are at least $\frac{1}{2}\sum_{v \in D} c_v(D)$ missing edges in $G(D)$, as any pair of vertices with the same color in D has no edge between them. Moreover, since each vertex v in D has $\overline{d}_v(S)$ non-neighbors in S, we can derive that incorporating the set D into the current k-deficient clique S will result in an increase of at least $\sum_{v \in D}(\overline{d}_v(S) + \frac{1}{2}c_v(D))$ missing edges. Therefore, by considering the largest subset D of C that satisfies $\sum_{v \in D}(\overline{d}_v(S) + \frac{1}{2}c_v(D)) \leq k - \overline{d}(S)$, we can conclude that the size of $\kappa(S, C)$ is at most $|S| + |D|$. □

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

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 π_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 π_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 π_i and assign weights $w(v_j)$ for each vertex v_j in π_i. Then, we can obtain that $\sum_{v_j \in D} w(v_j) \leq k - \overline{d}(S)$, as the size of π_i is no larger than $c_v(D)$, where v is a random vertex in π_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 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.

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

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 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 $N_{v_i}^{=2}(G) = \{u \in V | u \notin N_{v_i}(G), N_u(G) \cap N_{v_i}(G) \neq \emptyset\}$. 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.

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 $\quad S \leftarrow \{v_i\}$; $C_1 \leftarrow N_{v_i}(G^+_{v_i})$; $C_2 \leftarrow \emptyset$;
6 $\quad$ **while** $\exists u \in C_1$ *with* $d_u(C_1) < |S^*| - k - 1$ **do**
7 $\quad\quad C_1 \leftarrow C_1 \setminus \{u\}$;
8 $\quad$ **if** $|S \cup C_1| \leq |S^*| - k$ **then continue**;
9 $\quad$ **for** *each* $u \in N^{=2}_{v_i}(G^+_{v_i})$ **do**
10 $\quad\quad$ **if** $d_u(C_1) \geq |S^*| - k$ **then** $C_2 \leftarrow C_2 \cup \{u\}$;
11 $\quad$ **if** $|S^*| < k + 1$ **then** $C_2 \leftarrow \{v_{i+1}, v_{i+2}, ..., v_n\} \setminus C_1$;
12 $\quad$ Coloring $G(S \cup C_1 \cup C_2)$ with degeneracy ordering;
13 $\quad$ *Branch*$(S, C_1 \cup C_2)$; `// Algorithm 3 is employed`

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 κ vertices in C can be added into S. Hence, the total time-consuming of 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 δ, 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 the proposed 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 algorithms in finding the maximum k-defective clique [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 κ is the size of the maximum k-defective clique in a given graph G.

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. Table 2 presents the results of algorithms MDC, kDC, and KDBB, on 36 benchmark real-world graphs with varying k, where "—" denotes that the algorithm failed to complete the computations within a time threshold of 10800 seconds (3 hours). Note that in this experiment, we do not show the result of MADEC, as it is much less efficient than kDC and KDBB as reported in [11, 19]. As can be seen, except for a few datasets

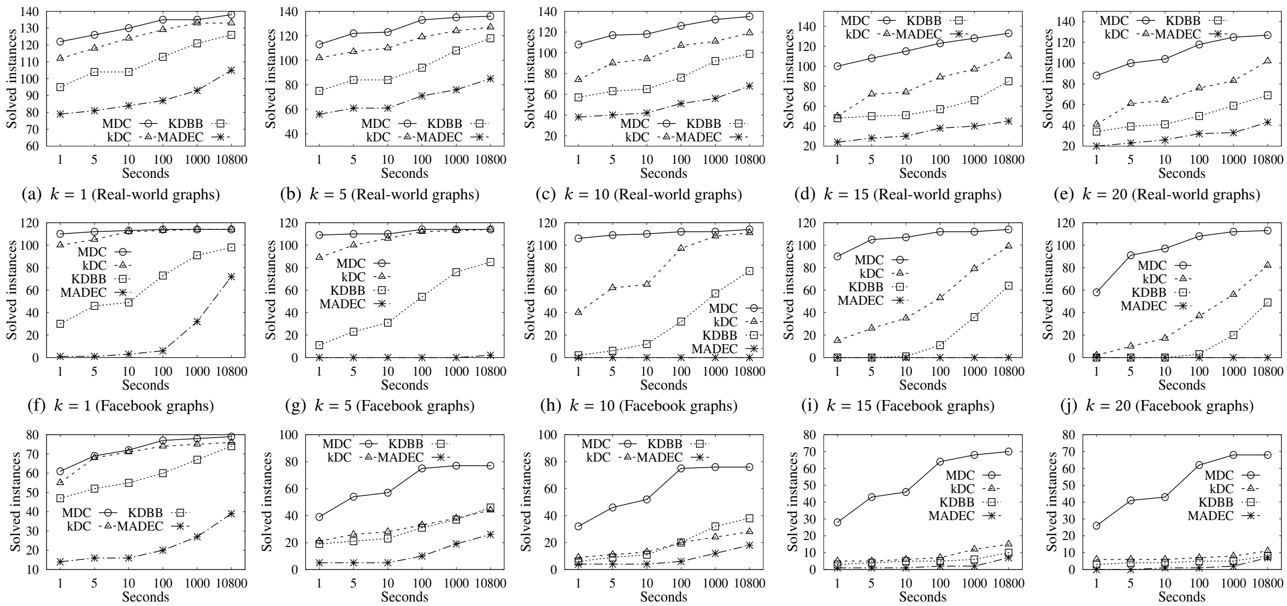
Table 2: Runtime of various algorithms on 36 benchmark real-world graphs (in seconds), where 1K=10^3, 1M=10^6.

Datasets	n	m	$k=1$			$k=5$			$k=10$			$k=15$			$k=20$		
			MDC	kDC	KDBB	MDC	kDC	KDBB	MDC	kDC	KDBB	MDC	kDC	KDBB	MDC	kDC	KDBB
ia-enron-large	33.6K	180K	**0.02**	0.06	0.10	**0.05**	1.11	50.74	**0.52**	43.79	344.66	**6.03**	1106.0	—	**69.26**	—	—
sc-ldoor	909K	20.8M	**7.17**	—	—	**34.72**	—	—	**983.18**	—	—	**3792.1**	—	—	—	—	—
sc-nasasrb	54.8K	1.3M	**0.15**	36.20	153.9	**0.43**	696.97	1472.5	**2.42**	1058.5	7070.0	**30.55**	2975.8	—	**173.76**	—	—
sc-pkustk11	87.8K	2.6M	**0.29**	0.41	1.79	**0.28**	0.49	4.40	**0.38**	2.54	35.61	**1.53**	2429.8	8403.4	**9.10**	4322.1	—
sc-pkustk13	94.8K	3.3M	**0.63**	6.46	34.37	**0.81**	41.47	976.07	**3.14**	1956.5	—	**9.85**	4334.0	—	**41.97**	8153.1	—
sc-pwtk	218K	5.6M	**0.92**	—	4406.3	**2.83**	—	—	**11.18**	—	—	**111.66**	—	—	**329.91**	—	—
sc-shipsec1	140K	1.7M	**0.03**	0.04	0.12	**0.03**	0.14	0.22	**0.26**	16.99	389.0	**1.44**	151.66	1225.8	**5.76**	1595.1	—
sc-shipsec5	179K	2.2M	**0.06**	1.84	3.55	**0.12**	11.19	107.2	**0.30**	29.79	424.7	**1.45**	606.7	2670.3	**3.99**	2495.6	—
scc_fb-forum	488	71K	0.03	**0.02**	0.05	**0.04**	0.12	1820.9	**0.18**	1.29	—	**0.34**	62.99	—	**0.81**	1745.7	—
scc_reality	6.8K	4.7M	**0.09**	0.38	3.19	**0.31**	7.60	38.12	**1.22**	9.05	—	**4.09**	15.91	—	**13.04**	1712.6	—
socfb-Amherst41	2.2K	90.9K	**0.12**	1.98	73.20	**0.47**	25.51	266.39	**2.00**	449.07	1725.2	**23.50**	6617.1	—	**41.92**	—	—
socfb-Auburn71	18.4K	973K	**0.41**	1.40	—	**0.47**	9.32	—	**0.51**	994.2	—	**1.27**	—	—	**5.29**	—	—
socfb-A-anon	3.1M	23.6M	7.35	**5.99**	194.1	8.97	**7.18**	1899	**10.73**	81.79	—	**34.52**	1452.9	—	**134.92**	8620.6	—
socfb-B-anon	2.9M	20.9M	**6.32**	8.90	113.6	**7.26**	10.63	1208.8	**9.48**	48.23	10623	**41.88**	2939.3		**409.40**	—	—
socfb-Columbia2	11.7K	444K	0.32	**0.23**	1.08	**0.38**	1.77	50.92	**0.43**	58.64	422.24	**1.00**	1063.2	5797.2	**2.29**	3247.4	—
socfb-Duke14	9.9K	506K	**0.60**	136.25	7070.3	**2.24**	1466.2	—	**11.06**	—	—	**54.82**	—	—	**509.23**	—	—
socfb-FSU53	27.7K	1.03M	**0.20**	0.36	—	**0.24**	3.04	—	**0.29**	269.93	—	**0.51**	—	—	**2.13**	—	—
socfb-Indiana	29.7K	1.3M	0.56	**0.45**	43.43	0.64	**0.55**	215.8	**0.67**	21.05	803.9	**0.71**	1200.7	5582.5	**0.83**	—	—
socfb-Texas80	31.6K	1.2M	**0.34**	0.62	257.08	**0.40**	0.76	7479.8	**0.42**	85.30	10258	**0.46**	—	—	**1.41**	—	—
socfb-Texas84	36.3K	1.6M	**0.48**	6.79	1148.5	**0.67**	75.07	2452.3	**0.95**	1420.8	9621.6	**2.82**	—	—	**14.02**	—	—
socfb-Tulane29	7.8K	284K	**0.12**	6.70	395.56	**0.13**	62.57	3288.7	**0.25**	800.06	7418.5	**1.19**	—	—	**5.53**	—	—
socfb-UF	35.1K	1.5M	**0.52**	0.57	617.83	**0.57**	0.59	1567.5	**0.57**	29.85	2459.9	**0.87**	—	—	**1.93**	—	—
socfb-UGA50	24.3K	1.2M	**0.61**	6.31	684.53	**0.68**	186.25	2175.4	**1.14**	3623.5	6039.2	**8.05**	—	—	**60.57**	—	—
socfb-Vanderbilt48	8.06K	427K	**0.22**	3.11	1484.8	**0.22**	9.17	—	**0.29**	3822.4	—	**0.55**	—	—	**2.84**	—	—
socfb-Wisconsin87	23.8K	836K	0.32	**0.18**	49.07	**0.30**	0.33	251.2	**0.39**	50.74	1787.5	**0.48**	2207.7	—	**1.61**	6240.7	—
soc-digg	771K	5.9M	**39.7**	911.16	—	**46.6**	4033.2	—	**120.2**	—	—	**621.29**	—	—	**9903.8**	—	—
soc-flixster	2.5M	7.9M	**0.59**	30.15	—	**1.05**	114.32	—	**2.51**	1766.49	—	**18.66**	—	—	**374.66**	—	—
soc-gowalla	197K	950K	**0.02**	0.12	0.32	**0.04**	0.15	0.18	**0.05**	22.52	47.94	**0.45**	3317.6	2218.1	**2.92**	—	—
soc-orkut	3M	106M	**78.5**	376.02	—	**192.62**	—	—	**350.4**	—	—	**465.0**	—	—	**2483.9**	—	—
soc-pokec	1.6M	22.4M	**5.10**	5.62	8.22	**4.65**	6.96	11.61	**5.55**	7.35	121.0	**8.31**	39.95	941.05	**13.58**	1114.7	—
soc-slashdot	70K	359K	**0.06**	3.21	197.90	**0.28**	41.79	1499.9	**1.32**	1629.8	5046.4	**9.10**	—	—	**91.67**	—	—
soc-youtube	496K	1.9M	**0.44**	1.44	104.8	**3.39**	15.66	2487.7	**26.73**	1154.4	—	**637.79**	—	—	—	—	—
tech-as-skitter	1.7M	11.1M	**0.12**	0.60	0.82	**0.14**	0.63	329.4	**0.19**	0.75	—	**0.26**	2.60	1758.3	**1.08**	—	—
tech-WHOIS	7.5K	56.9K	**0.02**	0.48	7.08	**0.03**	20.08	5665.3	**0.09**	4691.3	—	**2.04**	—	—	**17.42**	—	—
web-spam	4.8K	37.4K	**0.004**	0.01	0.652	**0.029**	0.93	44.452	**0.35**	32.54	911.1	**6.30**	1144.3	—	**16.17**	4335.4	—
web-uk-2005	130K	11.7M	**0.03**	0.16	0.78	**0.04**	0.34	11.12	**0.05**	0.78	34.25	**0.07**	1.27	76.28	**0.10**	1.67	114.90

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 at least 3 orders of magnitudes faster than kDC and KDBB on most parameter settings. For instance, when $k = 1$ and considering the sc-ldoor dataset, MDC only requires 7.17 seconds to identify the maximum k-defective clique, while kDC and KDBB fail to complete the computation within a given 10800 seconds. Furthermore, we observe that the time-consuming of our algorithm is relatively insensitive to increase in k on most datasets, whereas the performance of existing algorithms decreases sharply with increasing k. For example, on the socfb-Texas84 dataset, when $k = 1$, MDC, kDC, and KDBB take 0.48, 6.79, and 1148.5 seconds, respectively, to find the maximum k-defective clique. However, when k grows to 15, MDC only requires 2.82 seconds, while kDC and KDBB fail to complete the task in 10800 seconds. These experimental results clearly demonstrate the superior of our proposed algorithm in terms of running time. This is because the proposed branching rules and upper bound-based techniques are very efficient and effective in pruning unnecessary branches during the search process.

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. 3 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. 4 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,

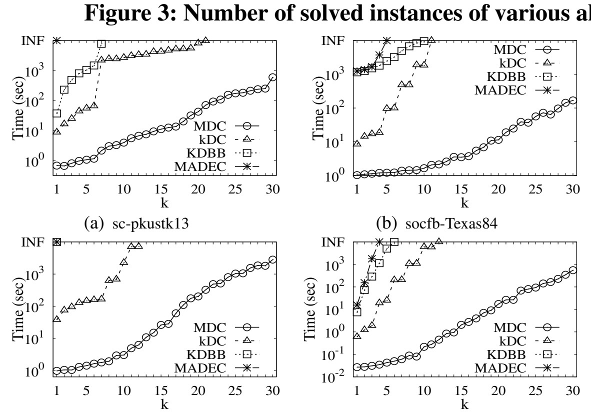


(a) $k = 1$ (Real-world graphs) (b) $k = 5$ (Real-world graphs) (c) $k = 10$ (Real-world graphs) (d) $k = 15$ (Real-world graphs) (e) $k = 20$ (Real-world graphs)

(f) $k = 1$ (Facebook graphs) (g) $k = 5$ (Facebook graphs) (h) $k = 10$ (Facebook graphs) (i) $k = 15$ (Facebook graphs) (j) $k = 20$ (Facebook graphs)

(k) $k = 1$ (DIMACS10 graphs) (l) $k = 5$ (DIMACS10 graphs) (m) $k = 10$ (DIMACS10 graphs) (n) $k = 15$ (DIMACS10 graphs) (o) $k = 20$ (DIMACS10 graphs)

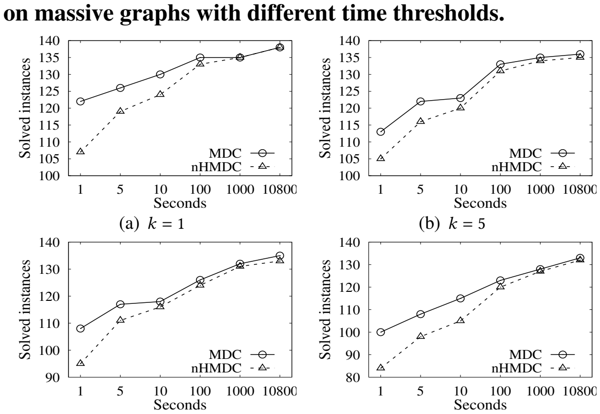
Figure 3: Number of solved instances of various algorithms on massive graphs with different time thresholds.



(a) sc-pkustk13 (b) socfb-Texas84

(c) soc-flixster (d) tech-WHOIS

Figure 4: Runtime of various algorithms with k growing.



(a) $k = 1$ (b) $k = 5$

(c) $k = 10$ (d) $k = 15$

Figure 5: Efficiency of MDC without the heuristic approach on 139 real-world graphs.

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. 5 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 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. 6. As can be seen, we note that MDC consistently

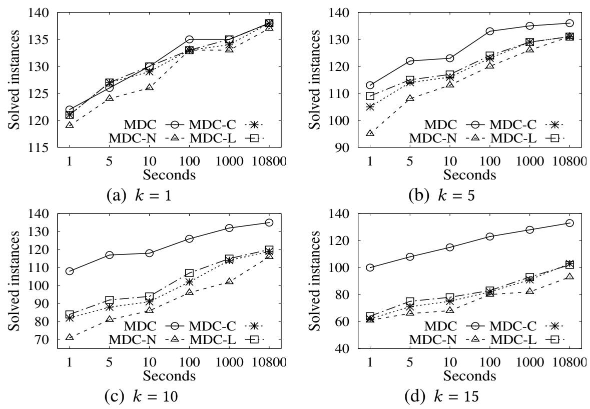


Figure 6: Efficiency of MDC with various upper bounds on 139 real-world graphs.

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 3 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 pivot-based branching rules is indeed efficient in reducing unnecessary branches of the search algorithm.

Table 3: Runtime of MDC without pivoting rules.

Datasets	$k = 1$		$k = 5$		$k = 10$	
	MDC	MDC-R	MDC	MDC-R	MDC	MDC-R
sc-nasasrb	**0.15**	0.22	**0.43**	0.65	**2.43**	3.15
socfb-Duke14	**0.60**	1.53	**2.24**	39.59	**11.06**	412.53
socfb-Texas84	**0.48**	0.55	**0.67**	1.45	**0.95**	20.66
soc-digg	**39.73**	50.01	**46.6**	76.29	**120.2**	1622.7
soc-flixster	**0.59**	0.59	**1.05**	7.91	**2.51**	139.57
tech-WHOIS	**0.02**	0.02	**0.03**	0.07	**0.09**	4.88

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, 28–30, 42, 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. [28–30] 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 [30], and further improving it with incremental upper bounds [29] and dynamic and static vertex ordering strategies [28]. 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.

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 γ-quasi-clique [32], and others. Among these models, significant attention has been given to the problems of finding the maximum k-plex [3, 12, 18, 23, 36, 39, 61, 64] and maximum γ-quasi-clique [35, 37, 41, 43, 44, 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. [36] 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 [23], and exploiting small dense subgraphs [12]. Concerning the maximum γ-quasi-clique problem, Pattillo et al. [44] and Veremyev et al. [58] formulated the problem using integer programming. Pajouh et al. [41], Pastukhov et al. [43], and Ribeiro et al. [47] respectively developed branch-and-bound algorithms incorporating various pruning techniques. Additionally, upper bounds on the maximum γ-quasi-clique number have been further explored in [35, 37]. 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 rule. Our analysis demonstrates that the proposed algorithm achieves a time complexity of $O(m\gamma_k^n)$, where γ_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 can be 3 orders of magnitude faster than the state-of-the-art solutions.

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, Fabio Furini, and Roberto Wolfler Calvo. 2021. A Branch-and-Price Framework for Decomposing Graphs into Relaxed Cliques. *INFORMS J. Comput.* 33, 3 (2021), 1070–1090.

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

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

[23] 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.

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

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

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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

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

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

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

[35] 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.

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

[37] 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.

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

[39] 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.

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

[41] 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.

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

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

[44] 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.

[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.