Efficient Maximum Defective Clique Search in Massive Graphs

ABSTRACT

The study of k-defective cliques, defined as induced subgraphs that differ from cliques by at most k missing edges, has garnered 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 sound and practically efficient algorithm that leverages newly-designed branch reduction rules and a pivot-based branching rule. 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. To our knowledge, this algorithm achieves the best worst-case time complexity to date compared to existing state-of-the-art approaches. Furthermore, to minimize unnecessary branches, we introduce a time-efficient upper bound based pruning algorithm that fully considers essential factors, including the vertex degree, core number, number of distinct colors, and the presence of non-neighbors among vertices. Additionally, to further enhance computational efficiency, we improve our algorithm by employing an ordering-based heuristic approach as a preprocessing step. Finally, we conduct extensive experiments on massive graphs to evaluate the efficiency of the proposed solutions. The results demonstrate that our algorithm achieves up to at least 3 orders of magnitude faster than the state-of-the-art solutions on the majority of large graphs.

PVLDB Reference Format:
. Efficient Maximum Defective Clique Search in Massive Graphs. PVLDB, 14(1): XXX-XXX, 2020.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://anonymous.4open.science/r/xxxxxxx.

1 INTRODUCTION

Graph has emerged as a versatile model for representing diverse real-world networks, including social networks [2], web networks [25], biological networks [47], 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, 26], identification of protein complexes in protein-protein interaction (PPI) networks [61, 64], and statistical analysis in financial networks [6, 7] all can be formulated as cohesive subgraph mining problems. Perhaps, the classical clique [35], which requires every pair of vertices associated with an edge, is a commonly used cohesive subgraph model, as extensively advanced solutions have been investigated in literature [9, 16, 41, 57].

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 [46]. 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 [46], including the k-plex [53], quasi-clique [33], r-clique [34], k-club [39], k-defective clique [64], and others.

In this paper, we primarily focus on the concept of the k-defective clique, which is formally defined as a subgraph $G(S)$ induced by the subset S of the graph G, such that it contains at least $\binom{|S|}{2} - k$ edges. This notion was originally introduced in [64] and has proven to be valuable in predicting implicit interactions among proteins in biological graphs. 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. Recently, there has been a surge of interest in the maximum k-defective clique problem, which involves identifying a k-defective clique with the largest cardinality among all k-defective cliques in graph G. The prominence of this problem stems not only from its practical relevance to various real-world applications, including community detection in social networks [20, 22] and statistical analysis in financial networks [14], but also from its close relationship with other cohesive subgraph models, such as clique [35] and k-plex [53].

As shown in [58, 63], the problem of identifying the maximum k-defective clique of a given graph G is NP-complete, thereby establishing the infeasibility of a polynomial-time algorithm unless NP=P. To our knowledge, there are only a few solutions that address this challenging problem [13, 19, 21, 58]. Specifically, Trukhanov et al. [58] pioneer an exact algorithm for the maximum k-defective clique problem based on the Russian doll search technique [60]. Subsequently, Gschwinda et al. [21] further improve this algorithm by employing new preprocessing methods and an optimized implementation. Recently, several new enumeration algorithms [13, 19], based on a branch-and-bound technique [27], have also been developed to enhance the efficiency. Notably, Chen et al. [13] introduce a new branching rule and employ some reduction techniques to improve the search for the maximum k-defective clique. This novel branching rule prioritizes vertices with non-neighbors in the current k-defective clique for branching, ensuring a maximum of $k+1$ subbranches in each recursive call. Significantly, the authors establish that the worst-case time complexity of this technique 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 mitigate unnecessary computations, an improved branch-and-bound enumeration algorithm is presented in [19] based on some newly branch pruning techniques. To our knowledge, this

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

algorithm represents currently the most advanced solution to the problem of identifying the maximum k-defective cliques.

However, these existing solutions still exhibit several noteworthy issues. Firstly, the practical performance remains prohibitively expensive when processing real-world graphs. This issue primarily stems from the insufficient tightening of the upper bound-based pruning techniques and inefficient of the branching rules employed in these existing solutions. Specifically, the presence of overly loose upper bounds often hinders the timely termination of the algorithm during the branch-and-bound process. Moreover, the inefficient branching rules cannot extensively identify the branches that yield 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 the majority of existing solutions still have a worst-case time complexity of $O(P(n)2^n)$, with and only one approach [13] achieving a worst-case time complexity of $O(P(n)\alpha_k^n)$, where $\gamma_k < 2$. However, even for relatively small values of k, the corresponding of α_k approaches approximately 2. For instance, as reported in [13], when $k = 2$ and 3, the respective values of α_k are 1.984 and 1.996, which result is unacceptably high for the problem of finding the maximum k-defective clique in graph G. 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.

Novel enumeration algorithms. To efficiently identify the maximum k-defective clique of G, we develop an elegant enumeration algorithm, which mainly combines two newly-developed branching rules. These rules ensure the algorithm'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 mathematically prove that the time complexity of this algorithm 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 $k \geq 1$. To our knowledge, our algorithm currently represents the most optimal solution in terms of worst-case time complexity.

New optimization techniques. To further improve the efficiency of our algorithm, we develop a set of optimization techniques. These include upper-bound based pruning and an ordering-based heuristic approach. Our research demonstrates 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 [52], and number of distinct colors present in G. Furthermore, 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 algorithm 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 algorithm, 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 139 real-world graphs, 83 DIMACS10 graphs, and 114 Facebook graphs. The experimental results demonstrate that our algorithms substantially outperform the state-of-the-art algorithm for maximum k-defective clique enumeration by up to 3 orders of magnitude on the majority of real-world graphs. For example, On the flixster dataset (with 7.9 million edges), our algorithm takes less than one second 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.

2 PROBLEM DEFINITION

Consider an undirected and unweighted graph $G = (V, E)$, where V and E represent the sets of vertices and edges of the graph G, respectively. Let $n = |V|$ and $m = |E|$ denote the number of vertices and edges in G, respectively. For a vertex v of G, we define $N_v(G)$ as the set of neighbors of v in G, i.e., $N_v(G) = \{u \in V | (u, v) \in E\}$. The degree of v in G is the cardinality of $N_v(G)$, denoted by $d_v(G) = |N_v(G)|$. Given a vertex subset S of G, we let $G(S) = (S, E_S)$ be the subgraph of G induced by the subset S, where $E_S = \{(u, v) \in E | u \in S, v \in S\}$. Then, the formal definition of the k-defective clique [64] is defined as follows.

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

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

Property 1 (Hereditary [58]). Given a k-defective clique S of G, every subset of S is also a k-defective clique of G.

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

Property 2 (Small diameter [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 (having a diameter of two with a size no less than $k + 2$) of the k-defective clique but also provides an acceleration for enumerating relatively-large maximal k-defective cliques [14]. However, the problem of enumerating maximal k-defective cliques often suffers from excessively long computational times, as there can be an exponential number of maximal k-defective cliques compared to the number of vertices. To address this challenge, in this paper, we aim to the problem of finding a maximum k-defective clique of G and the formal problem definition is shown below.

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

In Section 1, we have conducted an analysis that reveals the inefficiency of existing solutions [13, 19, 21, 58] in finding the maximum k-defective clique. To overcome these challenges, this paper will present a theoretically and practically efficient algorithm in the following sections.

3 THE EXACT ENUMERATION PARADIGM

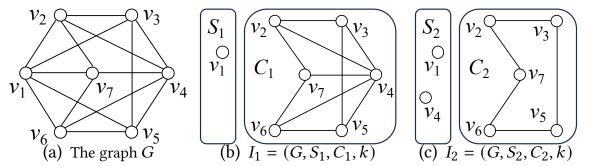
In this section, we present a novel algorithm designed to efficiently identify the maximum k-defective clique within a given graph G. Our algorithm builds upon the widely recognized branch-and-bound technique [27], which centers around dividing the current problem into smaller sub-problems. Specifically, we define an instance $I = (G, S, C, k)$ that aims to compute the maximum k-defective clique containing the set S within the subgraph $G(S \cup C)$. Here, S represents the current partial k-defective clique, while C denotes the candidate set used to expand S. By selecting a vertex v from C, the instance I can be split into two sub-instances: $I_1 = (G, S \cup \{v\}, C \setminus \{v\}, k)$ and $I_2 = (G, S, C \setminus \{v\}, k)$, utilizing the branch-and-bound technique. Notably, the solution to 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 introduce some new branch reduction rules and then present our enumeration algorithm.

3.1 Branch Reduction Rules

Given the 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.



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

Figure 1: Two non-neighbor reduction rule, where the maximum k-defective clique S^* of G is included in I_2 ($k \geq 1$).

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

(2) Two non-neighbors reduction: $\overline{d}_v(S \cup C) = 2$. Let u be the non-neighbor of v in $S \cup C$. It 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 $S \cup C$. Assuming 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 sole non-neighbor of v within $S \cup C$. 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 $I' = (G, S \cup \{v\}, C \setminus \{v\}, k)$ has the ability to retrieve a k-defective clique with a size no less than $|S^*|$. □

Based on Lemma 1 and Lemma 2, we can deduce 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 the scenario where the maximum k-defective clique of I includes vertex v. Consequently, this reduces the need to explore unnecessary sub-branches that aim to find the maximum k-defective clique excluding v. To illustrate these findings, the following example provides further clarification.

Example 1. Consider the graph G depicted in Fig. 1(a) with $k \geq 1$. It can be seen that $\overline{d}_{v_1}(G) = 2$. By leveraging Lemma 2, we can derive that there is a maximum k-defective clique (denoted as S^*) in G that includes v_1. Furthermore, as v_4 possesses the capability to expand the set $\{v_1\}$ while satisfying $\overline{d}_{v_4}(G) = 2$, it becomes evident that v_4 is also an integral part of S^*. This realization proves to be highly advantageous as it significantly reduces unnecessary computations.

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

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

Proof. Let S^* be a maximum k-defective clique of I. We will first prove the correctness for the case when $\overline{d}_v(S) = 0$. 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 $\overline{d}_v(S) = 0$. It can be observed that if either $\overline{d}_u(S) \geq 1$ (or $\overline{d}_w(S) \geq 1$) or $(u, w) \notin E$, then $S^* \setminus \{u\}$ (or $S^* \setminus \{w\}$) 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 of $\overline{d}_v(S) = 0$. Additionally, in sub-instance $I' = (G, S \cup \{u, w\}, C \setminus \{u, w\}, k)$, 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 when $\overline{d}_v(S) = 0$. The correctness for the case when $\overline{d}_v(S) = 1$ can be proven through a similar analysis to the previous case. Therefore, this lemma is established. □

The following example demonstrates the idea of Lemma 3.

Example 2. Consider a graph $G = (V, E)$ depicted in Fig. 2(a) where $k \geq 2$, we observe that $\overline{d}_{v_1}(G) = 3$. Utilizing the branch reduction rule (Lemma 3), we can identify the maximum k-defective clique S^* of G. Specifically, S^* 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. 2(b) and Fig. 2(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.

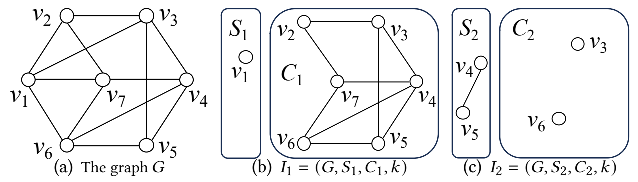


Figure 2: Three non-neighbor reduction rule, where the maximum k-defective clique S^* of G is either included in I_1 or in I_2 ($k \geq 2$).

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 restriction on the pivot vertex is overly stringent, resulting in numerous unnecessary computations when utilizing this pivot-based technique to identify the maximum k-defective clique of G. To illustrate, let us consider an instance $I = (G, S, C, k)$. If there exist two vertices $u \in S$ and $v \in S$ such that $N_v(G) \cap N_u(G) = \emptyset$, it becomes apparent that there is no vertex w in C that satisfies $S \subseteq N_w(G)$. Consequently, this instance I cannot be pruned using the pivot-based technique established in [14], leading to a substantial number of redundant computations. To address this concern, we introduce 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 solely 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, the sub-instance I_1 can yield a larger k-defective clique $S \cup D \cup \{v\}$ 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 acknowledge 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 represents a selected pivot vertex from C. For instance, let us consider S^* as the maximum k-defective clique of instance I. If $|S^* \setminus \{v\} \setminus N_v(G)| \geq 2$, it becomes apparent that such a maximum k-defective clique 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 address this concern, we propose an enhanced 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 two cases:*

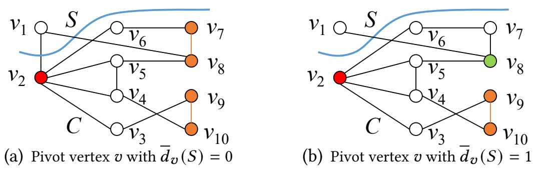
(1) If $\overline{d}_v(S) = 0$, the maximum k-defective clique for instance I either contains v or an edge in $G(P)$.

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

Proof. 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 $\overline{d}_v(S) = 0$. Now, consider a randomly chosen vertex u from D. Since D is an independent set, we can deduce 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 $S^* \setminus \{u\} \cup \{v\}$ 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 depicted in Fig. 3 serves as a practical illustration of the concept elucidated in Theorem 3.2.

Example 3. Consider the graph G shown in Fig. 3(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 deduce 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. 3(b).



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

Figure 3: The pivot-based technique of Theorem 3.2, where the red vertices are the pivot vertices. Then, the maximum k-defective clique either contains a red or green vertex or an orange edge ($k \geq 1$).

3.3 Algorithm Implementation

By incorporating the proposed branch reduction rules and pivoting techniques, we introduce a novel branching rule for efficiently finding the maximum k-defective clique in graph G, which is outlined below.

Branching rule. Consider an instance $I = (G, S, C, k)$ aiming to identify the maximum k-defective clique containing S in $G(S \cup C)$, where C is the set of vertices used to expand the current k-defective clique S. Let $N(S) = \{v \in V | S \subseteq N_v(G)\}$ be the set of common neighbors of S in G. Our branching rule is as follows:

- 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 $|C \setminus N(S)| + \overline{d}(S) \geq k \geq 1$, 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 technique described in Theorem 3.2 is employed to branch the instance I.

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

To begin, Algorithm 1 initializes the current maximum k-defective clique S^* as an empty set. Subsequently, 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. Specifically, if there exists a vertex u in C that satisfies $\overline{d}_u(S \cup C) \leq 3$ 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 $C \setminus N(S)$ is no larger than $|C| - k + \overline{d}(S)$ or each vertex in C has at least two non-neighbors in S (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 above conditions are not met,

Algorithm 1: The branch and bound algorithm

```
Input: The graph $G = (V, E)$ and a parameter $k$
Output: The maximum $k$-defective clique $S^*$ of $G$
$S^* \leftarrow \emptyset$;
$Branch(\emptyset, V)$;
return $S^*$;
Function: $Branch(S, C)$
    if $C = \emptyset$ then
        if $|S| > |S^*|$ then $S^* \leftarrow S$;
        return;
    $C_1 \leftarrow \{u \in C | \overline{d}_u(S) \leq 1\}$; $C_2 \leftarrow C \setminus C_1$;
    if $\exists u \in C_1$ such that $\overline{d}_u(S \cup C) \leq 3$ then
        Apply the branch reduction rules in Lemma 1-3;
    else if $|C \setminus N(S)| + \overline{d}(S) \geq k \geq 1$ or $C_1 = \emptyset$ then
        $v \leftarrow$ a vertex in $C \setminus N(S)$ with largest $\overline{d}_v(S)$;
        $C' \leftarrow Update(S, C, v)$;
        $Branch(S \cup \{v\}, C')$; $Branch(S, C \setminus \{v\})$;
    else
        $v \leftarrow$ a vertex in $C_1$ with the largest value of $d_v(C)$;
        $P_1 \leftarrow \{v\}$; $P_2 \leftarrow \overline{N}_v(C) \setminus P_1$;
        if $\overline{d}_v(S) = 1$ then
            $P_1 \leftarrow \{u \in \overline{N}_v(C) | \overline{d}_u(S) = 0\} \cup P_1$;
            $P_2 \leftarrow \overline{N}_v(C) \setminus P_1$;
        foreach $u \in P_1$ do
            $C' \leftarrow Update(S, C, u)$;
            $Branch(S \cup \{u\}, C')$; $C \leftarrow C \setminus \{u\}$;
        foreach $u \in P_2$ do
            $C' \leftarrow Update(S, C, u)$; $P_2' \leftarrow C' \cap P_2$;
            foreach $w \in P_2'$ s.t. $(u, w) \in E$ do
                $C'' \leftarrow Update(S \cup \{u\}, C', w)$;
                $Branch(S \cup \{u, w\}, C'')$; $C' \leftarrow C' \setminus \{w\}$;
            $C \leftarrow C \setminus \{u\}$;
```

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

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

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 $P_1 = \{v\} \cup \{u \in \overline{N}_v(C) | \overline{d}_v(S) = 0\}$ 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 the vertices in P_1 (lines 21-23) and the edges in $G(P_2)$ (lines 24-29). Finally, this recursion terminates when C becomes empty (lines 5-7). It is worth noting that if $C = \emptyset$ and the current detected k-defective clique S is larger than the current maximum result S^*, S^* is updated with the value of S (line 6).

In addition, to ensure that all remaining vertices in the candidate set can be utilized to expand $S \cup \{v\}$, it is necessary to update the candidate set when a vertex $v \in C$ is added to S. To address this requirement, we develop a procedure outlined in Algorithm 2. This algorithm 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, for $k = 1, 2$ 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 $|C \setminus N(S)| + \overline{d}(S) \geq k$, our algorithm selects a vertex v in $C \setminus N(S)$ 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 $C \setminus N(S)$

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 deduced that at most $k - \overline{d}(S) - 1$ vertices in $C \setminus N(S)$ can be added into S. Initially, when $\overline{d}(S) = 0$ and $|C \setminus N(S)| \geq k$, 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 $C \setminus N(S)$, the recurrence of Eq. (5) can be obtained accordingly. Otherwise, we can deduce that $|C \setminus N(S \cup \{v\})| \geq k - \overline{d}(S) + 2$, 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 $|C \setminus N(S)| + \overline{d}(S) \geq k$.

(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:

$$T(n) \leq \sum_{i=1}^{k/2+1} T(n-i), \quad \text{where } k \geq 2. \tag{7}$$

(4) If $|C \setminus N(S)| + \overline{d}(S) < k$, 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 deduce that $T(n) \leq \sum_{i=1}^{n} T(n-i)$. 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:

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

Based on the conditions of $T(n) \leq 2T(n-1)$ and $\sum_{i=1}^{\overline{d}-k} T(n-\overline{d}-2) \leq T(n-k-2)$ if $\overline{d} > k$, 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 deduce 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 bounded by $O(\kappa n + m)$.*

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 $O(\kappa n + m)$. ◻

4 OPTIMIZATIONS

In this section, we further improve our algorithm for finding the maximum k-defective clique of G, which involves three aspect improvements. Firstly, we incorporate the newly developed upper bound techniques to significantly reduce unnecessary branches within our branch-and-bound algorithm. Secondly, we introduce a novel heuristic approach aimed at identifying a near-maximum k-defective clique, denoted as S^*, within G. This allows us to reduce the scale of the graph by leveraging the size of $|S^*|$. Lastly, we introduce an ordering-based approach to efficiently identify the exact maximum k-defective clique from specific columns of subgraphs. In the following, we first introduce our proposed upper bounds, followed by a presentation of the improved algorithm.

4.1 Upper Bounds

Let κ (or $\kappa(C)$) denote the sizes of the maximum k-defective clique in graph G (or subgraph $G(C)$). In this subsection, we explore both conventional and innovative 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 we present the detailed result 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 can conclude that $\kappa \leq \max_{v \in V} d_v(G) + k + 1$.

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

Core-based upper bound. Now, we present a refined upper bound for both κ and $\kappa(C)$, drawing upon the well-established concept of k-core [52]. We provide the formal definition of k-core as follows.

Definition 2 ([52]). 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 represent k-core subgraph of G. The core number of a vertex v in G, denoted by $core_v(G)$, is defined as the maximum value of k such that v belongs to the k-core subgraph C_k of G, i.e., $core_v(G) = \max\{k \mid v \in C_k\}$. Based on this concept, we can establish a tighter upper bound as follows.

Lemma 5. For a given graph G, the size of the maximum k-defective clique in G containing a vertex $v \in V$ is bounded by $core_v(G) + k + 1$. Consequently, we can deduce that $\kappa \leq \max_{v \in V} core_v(G) + k + 1$.

Proof. It is easy to verify that any k-defective clique S in G also serves as a $(|S| - k - 1)$-core subgraph of G based on the definitions provided in Definition 1. Therefore, considering a maximum k-defective clique S^* of G, we can obtain that $core_v(G(S^*)) \geq |S^*| - k - 1$ for every $v \in S^*$. Thus, this lemma is established. □

To determine the core number for each vertex in a given graph G, we can employ the traditional peeling technique as described in [4]. This technique follows an iterative process of removing vertices from the remaining subgraph with degrees no larger than k, where k ranges in a non-decreasing order starting from $k = 0$ up to $k = n$. The core number of a vertex is assigned based on the value of k at the time of its removal. It is worth noting that the time complexity of this technique is bounded by $O(m)$, indicating its remarkable efficiency in generating the core-based upper bound.

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

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

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

Lemma 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, each vertex in S_1 has a unique color that differs from the colors of other vertices within S_1, while every vertex in S_2 shares the same color with at least one vertex in S_1. Formally, for each $v \in S_1$ (resp. $u \in S_2$), it holds that $\nexists w \in S_1 \setminus \{v\}$ with $color_v(G) = color_w(G)$ (resp. $\exists w \in S_1$ with $color_u(G) = color_w(G)$). Based on the concept of graph coloring, it is evident that two vertices u and v in G sharing the same color (i.e., $color_v(G) = color_u(G)$) are not connected by an edge ($(u, v) \notin E$). With this observation, we can deduce 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 emphasizes the importance of finding a smaller value for ω ($\omega(C)$) in order to 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 [28, 56]. In this paper, we adopt a widely used degeneracy ordering for graph coloring. The definition of degeneracy ordering [32] is presented below.

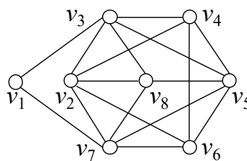
Definition 4. Given a graph G with the vertex set V, the degeneracy ordering is a permutation $\{v_1, v_2, ..., v_4\}$ of vertices in V such that for each vertex v_i, its degree is smallest in the subgraph of G induced by $\{v_i, v_{i+1}, ..., v_n\}$.

The degeneracy ordering, akin to the technique employed for computing the k-core of a graph, can be obtained by the peeling technique [4]. Specifically, the vertex removal ordering aligns with the degeneracy ordering, which can be accomplished within a time complexity of at most $O(m)$. Subsequently, we can systematically assign colors to each vertex v_i 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 $O(m)$ time.

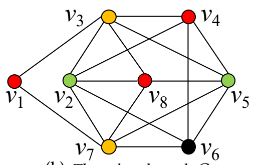
Let δ be the maximum core number of G, representing the highest value of k for which a non-empty k-core exists in G. Then, we derive the following relations for the aforementioned upper bounds.

Lemma 7. Given a graph G, let κ be the size of the maximum k-defective clique in G. We obtain that $\kappa \leq \omega + k \leq \delta + k + 1 \leq d_{max} + k + 1$, where d_{max} is the maximum degree of vertices in G.

Proof. It is easy to establish that $d_{\max} \geq \delta$ and $d_{\max} \geq \omega - 1$. Now, we only prove the correctness of $\omega \leq \delta + 1$. Assume that the vertex ordering $\{v_1, v_2, ..., v_n\}$ corresponds to the degeneracy ordering. Let $d^+_{v_i}$ be the degree of v_i in the subgraph $G(\{v_i, v_{i+1}, ..., v_n\})$. We observe that the color number assigned to v_i is at most $d^+_{v_i} + 1$ when only vertices in $\{v_{i+1}, v_{i+2}, ..., v_n\}$ have been colored. Based on the definition of the degeneracy ordering, we also deduce that



(a) The example graph G



(b) The colored graph G

Figure 4: An illustrative example for graph coloring.

for each vertex v_i, its degree is the smallest among all vertices in the subgraph $G(\{v_i, v_{i+1}, ..., v_n\})$, where $i \in [1, n]$. This result implies that the subgraph $G(\{v_i, v_{i+1}, ..., v_n\})$ also forms a $d^+_{v_i}$-core, indicating that $d^+_{v_i} \leq \delta$. Therefore, we establish $\omega \leq \delta + 1$, confirming the correctness of this lemma. □

The following example illustrates the above-mentioned upper bounds.

Example 4. Consider the graph G depicted in Fig. 4(a). It is easy to see that the maximum degree d_{max} and maximum core number δ of vertices in G are 5 and 4, respectively. Based on Lemma 4-5, we obtain that the upper bound on the size κ of the maximum k-defective clique is limited to 7 and 6 when $k = 1$. However, when employing the degeneracy ordering-based graph coloring (the colored graph shown in Fig. 4(b)), we compute that the number ω of distinct colors assigned to the vertices in G is 4. Consequently, we can enhance the upper bound of κ to 5 based on Lemma 6 when $k = 1$. This example serves as evidence of the efficiency of the proposed color-based upper bound.

Advanced color-based upper bound. Upon careful examination, we observe that the proposed technique for deriving the upper bound is not yet sufficiently tight. To illustrate this, let us consider a graph G depicted in Fig. 4(b), where the coloring is obtained using the degeneracy ordering. It is evident that G has 4 distinct colors. According to Lemma 6, the upper bound for κ is derived as 7 when $k = 3$, which corresponds to the number of vertices in G. However, it is worth noting that G itself does not form a 3-defective clique, thereby indicating that the upper bound for κ can be further tightened to 6. To address this limitation, we now introduce an improved 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 $\overline{d}_v(S)$ as the number of non-neighbors of vertex v in S, given by $\overline{d}_v(S) = |S \setminus N_v(S)|$. Moreover, we define $c_v(S)$ as the count of other vertices in S that share the same color as vertex v, denoted as $c_v(S) = |\{u \in S \setminus \{v\} | color_v(G) = color_u(G)\}|$. With these definitions established, we present the following lemma.

Lemma 8. 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 $G(S \cup D)$.

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. □

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.

Algorithm 3: Upperbound(S, C, k)

```
/* Each vertex in $S \cup C$ is colored based on the
   degeneracy ordering                                  */
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 $|S| + |D|$;
```

Lemma 9. 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 the subset D of C, we observe that there are at least $\frac{1}{2} \sum_{v \in D} c_v(D)$ missing edges in $G(D)$, as any pair of vertices with the same color in D has no edge between them. Moreover, since each vertex v in D has $\overline{d}_v(S)$ non-neighbors in S, we can deduce 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 $|S| + |D|$. □

Example 5. Reconsider the colored graph shown in Fig. 4(b). Assume that $S = \{v_1\}$ and $C = \{v_2, v_3, ..., v_7\}$, with $k = 1$. Applying Lemma 9, we can compute that $\kappa(S, C) \leq |S| + |D|$, where $D \subseteq C$. If $v_5 \in D$, we have $|D| \leq 3$ since $\overline{d}_{v_5}(S) = 1$ and there are only two different colors in $C \setminus \{v_5\}$. On the other hand, if $v_5 \notin D$, we still obtain that $|D| \leq 3$. Thus, we have $\kappa(S, C) \leq 4$. However, when utilizing Lemma 6, it yields $\kappa(S, C) \leq 5$, which is worse than the result obtained by Lemma 9.

Then, we present an algorithm, as shown in Algorithm 3, to compute the upper bound of $\kappa(S, C)$. This algorithm aims to determine the largest subset D of C. Initially, we initialize D as an empty set (line 1). Subsequently, 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). As 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). The algorithm terminates and outputs $|S| + |D|$ as the upper bound of $\kappa(S, C)$ when either C becomes empty or the number of missing edges in $G(S \cup D)$ violates the definition of a k-defective clique (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 $color_v(G) =$

$color_u(G)$, we obtain that $\overline{d}_v(S) > \overline{d}_u(S)$ and the vertex u must contain in D according to our algorithm. If $color_v(G) \neq color_u(G)$, we have $\overline{d}_v(S) + c_v(D) = \overline{d}_v(S) + c_v(D \cup \{u\})$ and the vertex u would be pushed into D in preference to v. However, we note that $u \notin D$, which is contradictory. Thus, we proved this lemma. □

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 involves sorting each vertex v in set C based on the size of $\overline{d}_v(S)$. This sorting process can be efficiently achieved in $O(kn)$ time using a bin sort. Furthermore, 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 $O(\overline{m})$. Therefore, this theorem is proven. □

It is worth noting that for real-world graphs, the preprocessing steps outlined in Lemma 4-6 often result in a higher density of the subgraph $G(S \cup C)$. Consequently, the number of missing edges in $G(C)$ tends to be relatively small. This advantageous characteristic enables Algorithm 3 to exhibit exceptional efficiency in computing the upper bound of $\kappa(S, C)$. The experiments presented in Sec. 5 further validate the effectiveness of this approach.

4.2 The Heuristic Algorithm

The underlying principle of this heuristic approach is to incrementally enlarge the current k-defective clique, denoted as S, by utilizing a candidate set C. Initially, we initialize S as an empty set, while C is populated to V. During each iteration, we carefully select a vertex from C and attempt to enlarge S. 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 maximize 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 $V^+_{v_i}$. Our heuristic approach, based on this ordering, focuses on computing the near-maximum k-defective clique within the subgraph $G^+_{v_i}$ that includes vertex v_i for each v_i in the ordering O. 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. Furthermore, we employ vertex selection strategies and pruning techniques throughout this process to enhance the quality of our result.

Vertex selection strategies. Given the vertex set C used to expand the current k-defective clique S, an intriguing question arises regarding the selection of vertices that can yield better results. We observe that vertices possessing larger degrees or core numbers in the subgraph are more likely to be part of the maximum k-defective clique. Leveraging this insight, we adopt an iterative approach to expand S using vertices from C with the maximum degree and core number, respectively. By doing so, we generate two candidate results and subsequently select the larger one as the final outcome.

Pruning techniques. In the process of expanding the set S with vertices from C, we observe that several vertices in C can be pruned

Algorithm 4: The heuristic algorithm

```
Input: The graph $G = (V, E)$ and a parameter $k \geq 0$
Output: A near maximum $k$-defective clique $S^*$ in $G$
1  Let $\{v_1, v_2, ..., v_n\}$ be the degeneracy ordering of vertices in $G$;
2  for $i = n$ to 1 s.t. $core(v_i) \geq |S^*| - k$ do
3      $S \leftarrow \{v_i\}$; $C \leftarrow N_{v_i}(G^+_{v_i})$;
4      while $\exists u \in C$ with $d_u(C) < |S^*| - k - 1$ do
5          $C \leftarrow C \setminus \{u\}$;
6      while $C \neq \emptyset$ do
7          $v \leftarrow$ a vertex in $C$ with maximum degree (or maximum
             core number) in $G(S \cup C)$;
8          $S \leftarrow S \cup \{v\}$ and remove each vertex $u$ from $C$ if $\overline{d}_u(S)$ is
             larger than $k - \overline{d}(S)$;
9          while $\exists u \in C$ s.t. $d_u(N_S(C)) < |S^*| - |S| - k + \overline{d}(S)$ do
10             Remove $u$ from $C$;
11         if $\exists u \in S$ with $d_u(N_S(C)) \leq |S^*| - |S| - k + \overline{d}(S)$ then
12             $S \leftarrow \emptyset$; $C \leftarrow \emptyset$;
13     foreach $v_j \in N^{=2}_{v_i}(G)$ s.t. $j > i$ do
14         if $d_{v_j}(S) \geq |S^*| - k + \overline{d}(S)$ then $C \leftarrow C \cup \{v_j\}$;
15     Further expand $S$ with vertices in $C$ as described in lines 6-12;
16     if $|S^*| < |S|$ then $S^* \leftarrow S$;
17 return $S^*$;
```

effectively. To achieve this, we employ three specific pruning techniques, considering 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 implementations. Equipped with the proposed techniques, we outline our heuristic approach in Algorithm 4.

Algorithm 4 begins by computes the degeneracy ordering of vertices in G using a method described in [4] (line 1). 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 O (lines 2-16). The algorithm constructs the candidate set C solely with the neighbors of v_i in $G^+_{v_i}$ to expand the current k-defective clique $S = \{v_i\}$. Then, it progressively selects a vertex v from C with the largest degree (or the maximum core number) to expand S until C becomes empty. 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)

Algorithm 5: Heuristic-based improved 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. $core(v_i) \geq |S^*| - k$ do
5      $S \leftarrow \{v_i\}$; $C_1 \leftarrow N_{v_i}(G^+_{v_i})$; $C_2 \leftarrow \emptyset$;
6      while $\exists u \in C_1$ with $d_u(C_1) < |S^*| - k - 1$ do
7          $C_1 \leftarrow C_1 \setminus \{u\}$;
8      if $|S \cup C_1| \leq |S^*| - k$ then continue;
9      for each $u \in N^{=2}_{v_i}(G^+_{v_i})$ do
10         if $d_u(C_1) \geq |S^*| - k$ then $C_2 \leftarrow C_2 \cup \{u\}$;
11     Coloring $G(S \cup C_1 \cup C_2)$ with degeneracy ordering;
12     $Branch(S, C_1 \cup C_2)$; /* Algorithm 3 is also invoked */
```

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

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

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

It is worth noting that the practical performance of Algorithm 4 can be highly efficient (as observed in the experimental results in Sec. 5). This can be attributed to the following reasons. Firstly, based on the property of the degeneracy ordering [32], we can deduce that $n' \leq \delta d_{max}$, where d_{max} is the maximum degree of vertices in G. Since δ tends to be relatively small in real-world graphs, n' remains reasonably small compared to n. Secondly, for most subgraphs $G(S \cup C)$, the number of vertices is significantly smaller than n'. Consequently, the practical cost of generating a near-maximum k-defective clique containing S in $G(S \cup C)$ is much lower than the worst-case time complexity. Lastly, the common neighbor based pruning technique effectively reduces the size of C, further enhancing the practical performance.

4.3 The Improved Algorithm

Equipping with the proposed heuristic approach, we then present an improved algorithm for identifying the maximum k-defective clique of G, outlined in Algorithm 5.

Algorithm 5 commences 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 exclusively focuses on computing the maximum k-defective clique of G within this subgraph (line 2). This approach is justified as other vertices are unequivocally excluded from being part of a k-defective clique with a size surpassing $|S^*|$. The algorithm then proceeds to iterate through each vertex v_i of G, following the reverse order of the degeneracy ordering, to compute the maximum k-defective clique containing v_i within G (lines 3-12). Prior to the enumeration 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 $|S^*| - k$, there exists no maximum k-defective clique in $G(S \cup C_1 \cup C_2)$ that both contains v_i and has a size larger than $|S^*|$. Consequently, the computation of v_i can be skipped (line 8). Following this, the algorithm employs the degeneracy ordering to color each vertex in $G(S \cup C_1 \cup C_2)$ (line 11), which serves as a prerequisite for calculating the proposed upper bound presented in Sec. 4.1. 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 12).

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 MDCE 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 KDBB and MADEC as the baseline for performance evaluation, where KDBB and MADEC are developed in [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 the proposed algorithms. The first set of datasets, consisting of 139 massive real-world graphs, is originally obtained from the Network Data Repository [49]. These datasets have been widely used in previous studies [18, 19, 65] and can be downloaded from http://lcs.ios.ac.cn/~caisw/graphs.html. The second set of datasets, which includes 114 Facebook graphs, is available at https://networkrepository.com/socfb.php. Lastly, The third set of datasets comprises 83 DIMACS10 graphs, which can be accessed at https://networkrepository.com/dimacs.php. These graphs have

Table 1: Runtime of various algorithms on 38 real-world graphs (in seconds), where 1K=10^3, 1M=10^6.

			$k = 1$			$k = 3$			$k = 5$			$k = 10$		
Datasets	n	m	MDCE	KDBB	MADEC	MDCE	KDBB	MADEC	MDCE	KDBB	MADEC	MDCE	KDBB	MADEC
web-spam	4.8K	37.4K	**0.004**	0.652	8.13	**0.008**	5.720	1226.7	**0.02**	44.452	—	**0.35**	911.1	—
tech-WHOIS	7.5K	56.9K	**0.02**	7.08	3.50	**0.02**	287.4	321.3	**0.03**	5665.3	8794.9	**0.09**	—	—
scc_retweet	1.2K	65.9K	**0.001**	0.10	0.033	**0.08**	0.55	0.67	**0.013**	32.97	11.63	**0.026**	220.29	2606.7
scc_fb-forum	488	71K	**0.02**	0.22	0.05	**0.03**	13.19	1.38	**0.04**	1820.9	41.37	**0.18**	—	—
ia-enron-large	33.6K	180.8K	**0.021**	0.105	430.67	**0.033**	6.63	—	**0.057**	50.74	—	**0.52**	344.66	—
soc-douban	155K	327K	**0.01**	0.01	350.24	**0.01**	0.6	597.8	**0.01**	3.01	1063.2	**0.02**	1018.2	2472
soc-slashdot	70K	359K	**0.06**	197.9	1433.3	**0.14**	618.7	—	**0.28**	1499.9	—	**1.32**	5046.4	—
socfb-Duke14	9.9K	506K	**0.60**	7070	—	**1.52**	8529	—	**2.24**	—	—	**11.06**	—	—
socfb-UConn	17.2K	604.8K	**0.08**	2.67	4725.9	**0.085**	19.76	—	**0.095**	21.93	—	**0.132**	35.61	—
tech-RL-caida	191K	608K	**0.01**	0.06	98.02	**0.01**	1.13	737.27	**0.01**	3.38	—	**0.04**	51.10	—
socfb-wosn-friends	6.4K	817K	**0.13**	13.37	—	**0.14**	44.02	—	**0.18**	171.4	—	**0.23**	637.2	—
socfb-Wisconsin87	24K	836K	**0.32**	49.07	—	**0.34**	97.58	—	**0.30**	251.2	—	**0.39**	1787	—
socfb-Berkeley13	22.9K	852.4K	**0.23**	12.82	—	**0.23**	22.00	—	**0.24**	64.18	—	**0.29**	155.39	—
soc-gowalla	197K	950K	**0.001**	0.001	900.4	**0.001**	0.009	—	**0.041**	0.087	—	**0.059**	47.94	—
socfb-UGA50	24K	1.2M	**0.61**	684.53	—	**0.62**	1358.6	—	**0.68**	2175.4	—	**1.14**	6039.2	—
socfb-Texas80	31K	1.2M	**0.34**	257.0	7262	**0.32**	1529	—	**0.40**	7479	—	**0.42**	10258	—
socfb-Indiana	29K	1.3M	**0.56**	43.43	—	**0.57**	160.5	—	**0.64**	215.8	—	**0.67**	803.9	—
sc-nasasrb	55K	1.3M	**0.15**	153.9	1384.8	**0.23**	311.5	1400.2	**0.43**	1472.5	1400.1	**2.42**	7070.0	1428.8
soc-delicious	536K	1.4M	**0.011**	0.15	62.4	**0.014**	0.12	2242.5	**0.017**	3.38	—	**0.035**	56.93	—
socfb-UF	35K	1.5M	**0.62**	617.83	—	**0.64**	1207.0	—	**0.59**	1567.5	—	**0.68**	2459.9	—
socfb-Texas84	36K	1.6M	**0.58**	1148.5	—	**0.62**	1506.1	—	**0.70**	2452.3	—	**0.95**	9621.6	—
sc-shipsec1	140K	1.7M	**0.02**	0.12	2180.9	**0.03**	0.17	2555.6	**0.03**	0.22	3534.8	**0.26**	389.0	—
soc-youtube	496K	1.9M	**0.44**	104.8	—	**1.19**	620.2	—	**3.39**	2487.7	—	**26.73**	—	—
sc-shipsec5	179K	2.2M	**0.06**	3.55	—	**0.09**	50.58	—	**0.12**	107.2	—	**0.30**	424.7	—
sc-pkustk11	88K	2.56M	**0.29**	1.79	896.35	**0.28**	4.01	1843.2	**0.28**	4.40	2159.6	**0.38**	35.61	4572.7
sc-pkustk13	94.8K	3.26M	**0.636**	34.37	6685.4	**0.655**	469.54	6758.3	**0.709**	976.00	—	**1.148**	—	—
scc_reality	6.8K	4.7M	**0.001**	0.004	0.29	**0.001**	0.004	0.29	**0.14**	38.12	1.47	**1.08**	—	—
sc-pwtk	218K	5.6M	**0.92**	4406.3	—	**1.78**	—	—	**2.83**	—	—	**11.1**	—	—
soc-digg	771K	5.9M	**39.7**	—	—	**131**	—	—	**46.6**	—	—	**120.2**	—	—
web-it-2004	509K	7.2M	**0.001**	0.001	0.57	**0.13**	2.22	0.55	**0.14**	4.22	0.55	**0.34**	16.95	5.8
soc-flixster	2.5M	7.9M	**0.59**	—	—	**0.67**	—	—	**1.05**	—	—	**2.51**	—	—
tech-as-skitter	1.7M	11M	**0.12**	0.824	705.63	**0.12**	38.18	—	**0.14**	329.4	—	**0.19**	—	—
web-uk-2005	130K	12M	**0.18**	0.78	0.86	**0.29**	3.34	3.44	**0.32**	11.12	3.49	**0.53**	34.25	9.19
sc-ldoor	909K	21M	**7.17**	—	—	**17.4**	—	—	**34.7**	—	—	**983**	—	—
socfb-B-anon	2.9M	21M	**10.8**	194.1	—	**11.6**	504.3	—	**11.7**	1899	—	**14.2**	—	—
soc-pokec	1.6M	22M	**4.92**	4.225	—	**5.46**	8.51	—	**6.07**	11.61	—	**8.29**	121.0	—
socfb-A-anon	3.1M	23M	**11.3**	113.6	—	**12.4**	574.9	—	**12.9**	808.8	—	**15.7**	10623	—
soc-orkut	3M	106M	**78.5**	—	—	**132.5**	—	—	**192.6**	—	—	**350.4**	—	—

been utilized as benchmark graphs for evaluating the performance of the maximum k-defective clique enumeration algorithms [19].

Parameters. In our experimental evaluations, we follow a similar approach as [13, 19] by considering values of k as 1, 3, 5, and 10. However, 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: Performance evaluation on representative benchmark real-world graphs. In this experiment, we evaluate the performance of various algorithms in finding the maximum k-defective clique. Table 1 presents the results of each algorithm, including MDCE, KDBB and MADEC, on 38 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). As observed, the runtime of our algorithm MDCE consistently outperforms all existing algorithms (KDBB and MADEC) on all tested benchmark datasets across different values varying k. More, specifically, our algorithm achieves 3 orders of magnitudes faster than KDBB and MADEC on most datasets. For instance, when $k = 1$ and considering the flixster dataset, MDCE only requires 0.59 seconds to identify the maximum k-defective clique, while KDBB and MADEC 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 tech-as-skitter dataset, when $k = 1$, MDCE, KDBB and MADEC take 0.12 seconds, 0.824 seconds, and 705.63 seconds, respectively, to find the maximum k-defective clique. However, when k grows to 10, MDCE only requires 0.19 seconds, while KDBB and MADEC fail to complete the task in 10800 seconds. This notable performance improvement is due to the effectiveness of the proposed branching rules and upper-bound techniques in pruning unnecessary branches during the enumeration process. These experimental results clearly demonstrate the superior of our proposed algorithm in terms of runtime efficiency.

Exp-2: Solved instance of various algorithms on massive graphs. In this experiment, we aim to evaluate the performance of our proposed algorithm on massive datasets, comprising 139 real-world graphs, 83 DIMACS10 graphs, and 114 Facebook graphs. Figure 5 showcases the number of solved instances with varying time thresholds for different values of k. From this figure, we observe that our algorithm, MDCE, consistently outperforms the state-of-the art algorithms, KDBB and MADEC, in terms of the number

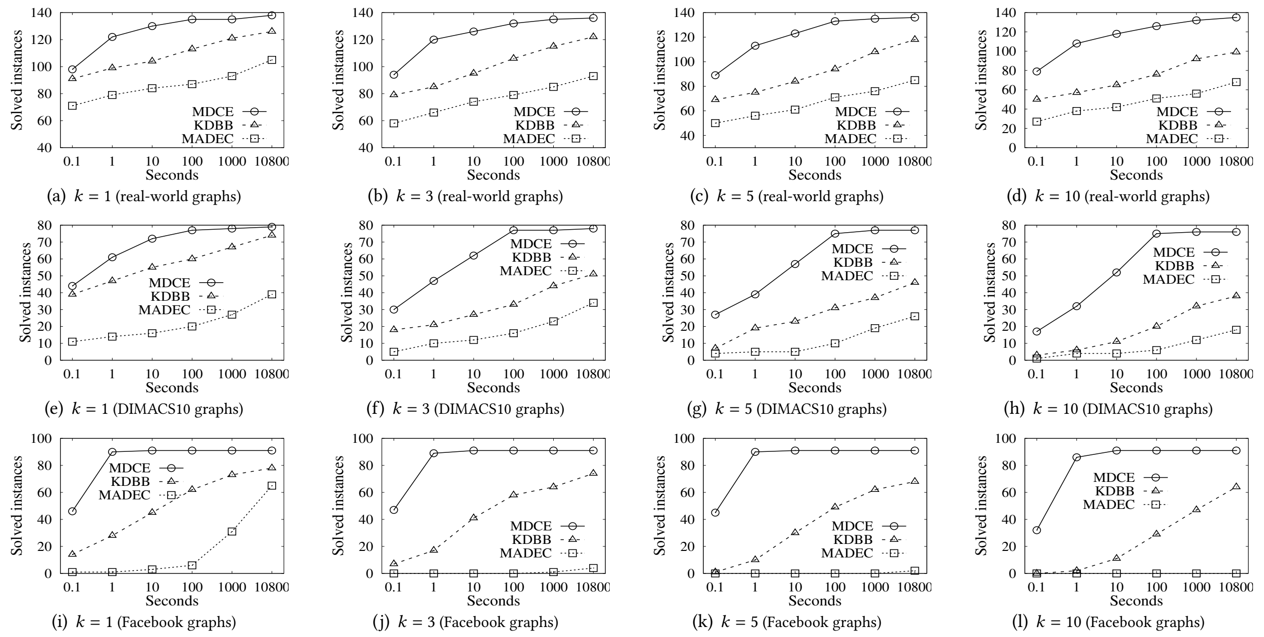


Figure 5: Number of solved instances of various algorithms on massive graphs with different time thresholds.

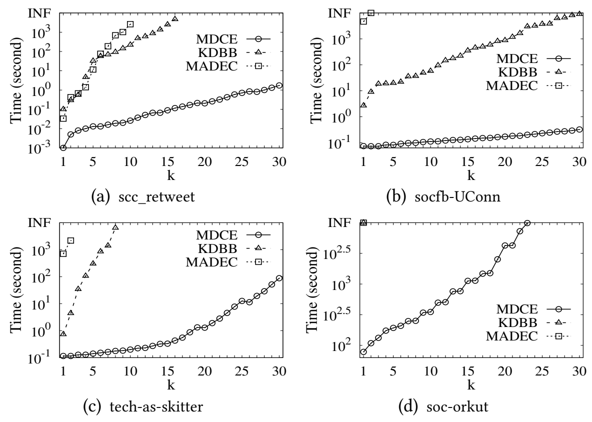
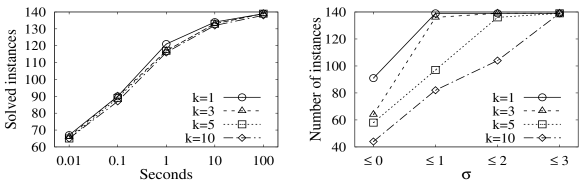


Figure 6: Runtime of various algorithms with k growing.

of solved instances. This result validates the efficiency of our propose algorithm in efficiently identifying the maximum k-defective clique of G. Furthermore, as the value of k increases, we observe that the decrease in the number of solved instances for MDCE is much less significant compared to the state-of-the-art algorithms KDBB and MADEC. For instance, when $k = 1$, MDCE, KDBB, and MADEC solve 135, 113 and 87 instances within 100 seconds, respectively. However, when k grows to 10, MDCE solves 126 instances, while KDBB and MADEC can only handle 76 and 51 instances, respectively, under the same conditions. This further emphasizes the efficiency of our proposed techniques in reducing unnecessary computations, even when dealing with larger values of k.

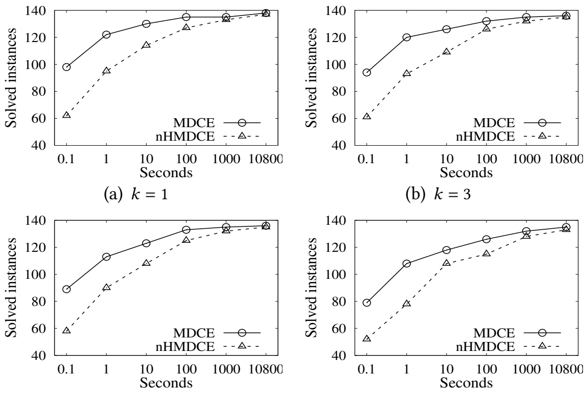
Exp-3: Runtime of various algorithms with k growing. In this experiment, we further evaluate the performance of each algorithm on 4 representative datasets with varying k. Figure 6 illustrates the detailed experimental results for three tested algorithms, with a time threshold of 3 hours. As can be seen, our algorithm, MDCE, successfully solves majority of tested graphs even when k is increased up to 30. However, the state-of-the-arts, 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, MDCE, increases smoothly with the increase of k, whereas the existing algorithms, KDBB and MADEC, exhibit sharp increases. For instance, on the scc_retweet dataset, when k takes values of 1, 10, and 20, MDCE completes the computations in 0.001, 0.026, and 0.205 seconds, respectively. However, the existing algorithm KDBB (MADEC) requires 0.1 (0.33), 220.34 (2606.74), and >10800 (>10800) seconds, respectively. This experimental result demonstrates that our algorithm maintains excellent pruning performance even as k grows significantly large.

Exp-4: Results of the proposed heuristic algorithm. Denote by HMDC our proposed heuristic approach (Algorithm 4). In this experiment, we assess the efficiency and effectiveness of HMDC by examining the number of instances it solves under different time threshold and the difference in size between the results returned by HMDC and MDCE. Fig. 7 displays the experimental results on 139 real-world graphs. Here, σ represents $\kappa - |D^*|$, where κ is the size of the maximum k-defective clique and D^* is the result obtained by HMDC. Note that the results on other datasets are consistent. From this figure, we can note that the runtime of the proposed heuristic algorithm, HMDC, is highly efficient, taking less than 100 seconds for almost all tested datasets when $k \leq 10$. Moreover, we observe that the runtime of HMDC remains stable even with changes in k, indicating its ability to handle large real-world graphs effectively. Additionally, we observe that the size of D^* obtained by HMDC closely approximates κ. For instance, when $k = 1$ and $k = 3$, the gap σ between κ and $|D^*|$ consistently remains below or equal to 3



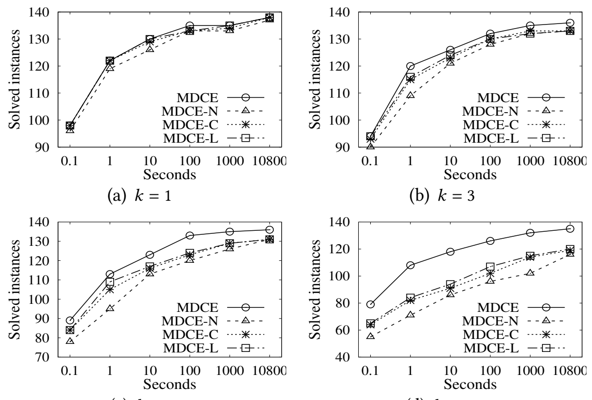
(a) 139 Real-world graphs (b) 139 Real-world graphs

Figure 7: Results of the heuristic approach.



(a) $k = 1$ (b) $k = 3$ (c) $k = 5$ (d) $k = 10$

Figure 8: Solved instances of MDCE without the heuristic approach.



(a) $k = 1$ (b) $k = 3$ (c) $k = 5$ (d) $k = 10$

Figure 9: Solved instances of MDCE without upper bound techniques.

Table 2: Runtime of MDCE without branch reduction rules.

	$k = 1$		$k = 5$		$k = 10$	
Datasets	MDCE	MDCE-R	MDCE	MDCE-R	MDCE	MDCE-R
scc_fb-messages	**0.29**	1.38	**0.32**	1.39	**0.54**	1.61
scc_reality	**2.29**	7.27	**2.64**	7.48	**3.80**	8.59
sc-ldoor	**7.17**	10.23	**34.75**	45.36	**983.3**	1373.1
socfb-Duke14	**0.60**	0.70	**2.24**	2.84	**11.06**	14.66
socfb-Texas84	**0.58**	0.65	**0.70**	0.88	**0.95**	1.41
soc-pokec	**4.92**	5.25	**6.07**	7.35	**8.29**	10.22
tech-WHOIS	**0.022**	0.023	**0.032**	0.042	**0.092**	0.27

across all 139 real-world graphs, thus further affirming the efficacy of our proposed heuristic approach.

Exp-5: Performance evaluation of the proposed algorithm without the heuristic approach. We define our algorithm MDCE without the heuristic approach as nHMDCE, which refers to Algorithm 5 without lines 1-2. In this experiment, we test the effectiveness of algorithm nHMDCE in finding the maximum k-defective clique of G. Fig. 8 illustrates the number of solved instanced of nHMDCE and MDCE on 139 real-world graphs with different time thresholds when varying k. From the results, we note that MDCE consistently solves more instances than nHMDCE. 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 the enumeration algorithm is substantially improved. Nevertheless, it is worth noting that nHMDCE 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-6: Performance evaluation of proposed upper bound techniques. In this experiment, we test the effectiveness of the proposed upper bounds. We designate MDCE-N as Algorithm 5 without any upper bounds, and let MDCE-C and MDCE-L be Algorithm 5 augmented with core-based (Lemma 5) and color-based (Lemma 6) upper bounds, respectively. We conduct experiments on a set of 139 real-world graphs, as described in Fig. 9. As can be seen, we note that MDCE consistently demonstrates superior performance compared to all other tested algorithms. This can be attributed to the tightness of the proposed advanced color-based upper bound and the algorithm's efficiency in computing our proposed upper bound. Additionally, we observe that within a given runtime threshold, the gap in number of solved instances between the algorithm MDCE and the other algorithms (MDCE-N, MDCE-C, and MDCE-L) increases as k grows. For instance, with a time threshold of 100 seconds, when $k = 1$, MDCE solves 135 instances, while both MDCE-N, MDCE-C, and MDCE-L solve 133 instances each. However, when k grows to 10, MDCE solves 126 instances, whereas MDCE-N, MDCE-C, and MDCE-L only manage to solve 96, 102, and 107 instances, respectively. This experiment further highlights the efficiency of the proposed upper bound pruning technique.

Exp-7: Performance evaluation of proposed branch reduction rules. This experiment aims to evaluate the performance of the proposed branch reduction rules. Let us denote MDCE-R the algorithm MDCE without the graph reduction rules developed in Sec. 3.1. Table 2 presents the experimental results obtained by executing MDCE and MDCE-R on 7 representative real-world graphs with varying k. The results clearly demonstrate that the runtime of MDCE consistently outperforms that of MDCE-R across all tested datasets. Moreover, even as k increases, MDCE maintains its superior performance compared to MDCE-R. These findings provide important evidence supporting the efficiency of the proposed branch reduction rules in effectively reducing unnecessary branches of our proposed algorithm.

6 RELATED WORKS

Maximum clique enumerations. The problem of determining the maximum clique in a graph G has been proven to be NP-hard [8], with approximating a satisfactory solution being a challenging task [66]. Over the past few decades, numerous exact algorithms have been developed to tackle this problem [10, 11, 29–31, 43, 50, 51, 54–56]. 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. [54–56] and Li et al. [29–31] have gained significant popularity. Specifically, Tomita et al. [54] introduced an algorithm based on degeneracy ordering and further improved it using the graph-recoloring technique [56] and the adjunct coloring-based ordering [55]. On the other hand, Li et al. proposed a branch-and-bound algorithm based on MaxSAT reasoning[31], and further improving it with incremental upper bounds [30] and dynamic and static vertex ordering strategies [29]. Additionally, several parallel approaches utilizing multi-cores have also been developed to enhance practical efficiency [50, 51]. 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 enumerations. 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 [46]. These include the k-plex [53], s-clique [46], and γ-quasi-clique [33], and others. Among these models, significant attention has been given to the problems of finding the maximum k-plex [3, 12, 18, 23, 37, 40, 62, 65] and maximum γ-quasi-clique [36, 38, 42, 44, 45, 48, 59] 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. [37] introduced a combinatorial algorithm based on co-k-plex coloring as an upper bound technique. Xiao et al. [62] 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 [65], partition-based upper bounds [23], and exploiting small dense subgraphs [12]. Concerning the maximum γ-quasi-clique problem, Pattillo et al. [45] and Veremyev et al. [59] formulated the problem using integer programming. Pajouh et al. [42], Pastukhov et al. [44], and Ribeiro et al. [48] 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 [36, 38]. 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.

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 introduce an efficient enumeration algorithm accompanied by a series of novel optimization techniques. Specifically, our enumeration 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 novel upper bounds that we developed. Moreover, we make additional improvements to our algorithm by incorporating an ordering-based heuristic-based algorithm as the preprocessing step. Finally, we conduct comprehensive experiments to validate the effectiveness and efficiency of our proposed approaches.

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] Randy Carraghan and Panos M Pardalos. 1990. An exact algorithm for the maximum clique problem. *Operations Research Letters* 9, 6 (1990), 375–382.
[11] Lijun Chang. 2020. Efficient maximum clique computation and enumeration over large sparse graphs. *VLDB J.* 29, 5 (2020), 999–1022.
[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] Andrea Lancichinetti and Santo Fortunato. 2009. Community detection algorithms: a comparative analysis. *Physical review E.* 80, 5 (2009), 056117.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

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

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

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

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

[36] 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.

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

[38] 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.

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

[40] 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.

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

[42] 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.

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

[44] 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.

[45] 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.

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

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

[48] 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.

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

[50] 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).

[51] 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.

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

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

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

[61] 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.

[62] 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.

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

[64] 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.

[65] 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.

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