Efficient Defective Clique Enumeration and Search with Worst-Case Optimal Search Space

JIHOON JANG, Seoul National University, Republic of Korea
YEHYUN NAM, Seoul National University, Republic of Korea
KUNSOO PARK*, Seoul National University, Republic of Korea
HYUNJOON KIM, Hanyang University, Republic of Korea

A k-defective clique is a relaxation of the traditional clique definition, allowing up to k missing edges. This relaxation is crucial in various real-world applications such as link prediction, community detection, and social network analysis. Although the problems of enumerating maximal k-defective cliques and searching a maximum k-defective clique have been extensively studied, existing algorithms suffer from limitations such as the combinatorial explosion of small partial solutions and sub-optimal search spaces. To address these limitations, we propose a novel clique-first branch-and-bound framework that first generates cliques and then adds missing edges. Furthermore, we introduce a new pivoting technique that achieves a search space size of $O(3^{\frac{n}{3}} \cdot n^k)$, where n is the number of vertices in the input graph. We prove that the worst-case number of maximal k-defective cliques is $\Omega(3^{\frac{n}{3}} \cdot n^k)$ when k is a constant, establishing that our algorithm's search space is *worst-case optimal*. Leveraging the diameter-two property of defective cliques, we further reduce the search space size to $O(n \cdot 3^{\frac{\delta}{3}} \cdot (\delta\Delta)^k)$, where δ is the degeneracy and Δ is the maximum degree of the input graph. We also propose an efficient framework for maximum k-defective clique search based on our branch-and-bound, together with practical techniques to reduce the search space. Experiments on real-world benchmark datasets with more than 1 million edges demonstrate that each of our proposed algorithms for maximal k-defective clique enumeration and maximum k-defective clique search outperforms the respective state-of-the-art algorithms by up to four orders of magnitude in terms of processing time.

CCS Concepts: • **Mathematics of computing → Graph algorithms**.

Additional Key Words and Phrases: Defective clique, Branch-and-bound, Worst-case optimal

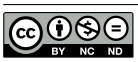
ACM Reference Format:
Jihoon Jang, Yehyun Nam, Kunsoo Park, and Hyunjoon Kim. 2025. Efficient Defective Clique Enumeration and Search with Worst-Case Optimal Search Space. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 322 (December 2025), 28 pages. https://doi.org/10.1145/3769787

1 Introduction

Graphs have been widely used to model relationships among entities in various domains, including social networks, biological systems, financial markets, and transportation networks. An important task in graph analysis is the identification of dense substructures, as these often reveal significant patterns or underlying relationships within the data [16, 50]. For instance, dense subgraphs can correspond to densely connected communities in social networks [6, 31, 48], protein complexes in

*Corresponding author.

Authors' Contact Information: Jihoon Jang, jhjang@theory.snu.ac.kr, Seoul National University, Seoul, Republic of Korea; Yehyun Nam, yhnam@theory.snu.ac.kr, Seoul National University, Seoul, Republic of Korea; Kunsoo Park, kpark@theory.snu.ac.kr, Seoul National University, Seoul, Republic of Korea; Hyunjoon Kim, hyunjoonkim@hanyang.ac.kr, Hanyang University, Seoul, Republic of Korea.



ACM 2836-6573/2025/12-ART322
https://doi.org/10.1145/3769787

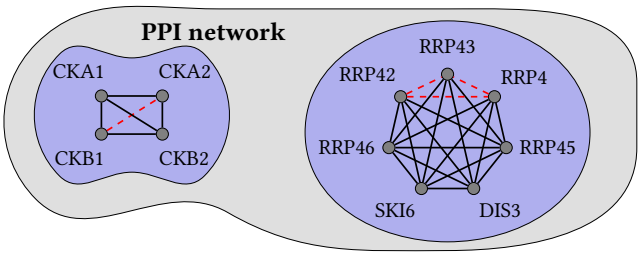


Fig. 1. A noisy protein-protein interaction (PPI) network [4] containing two defective cliques as subgraphs (highlighted in blue). Missing interactions in the noisy network can be predicted by identifying defective cliques and completing the missing edges (represented by red dashed edges). After completing the missing edges, two protein complexes can be identified (left: Casein Kinase II, right: Exosome complex), each forming a clique.

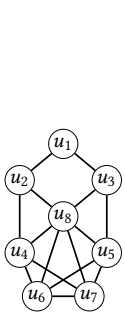
biological networks [39, 88], or anomalies in financial networks [2]. A clique, defined as a subgraph in which every vertex pair is connected by an edge, represents the densest possible subgraph. There has been growing interest in developing practical algorithms for solving clique-related problems, such as maximal clique enumeration [10, 19, 22, 30, 32, 77], maximum clique search [11, 12, 65, 76, 84], k-clique enumeration [28, 53, 58, 80, 90], and k-clique counting [3, 15, 40, 71, 87].

In real-world applications, the definition of a clique is often overly restrictive, as real-world data is frequently noisy or incomplete [88]. Moreover, requiring all members of a group to be connected is overly strict, as groups with a few missing relationships can still effectively represent a community [61]. To overcome this limitation, several relaxations of cliques have been proposed, including the quasi-clique [1], k-defective clique [88], k-plex [5], k-club [9], and k-bundle [61].

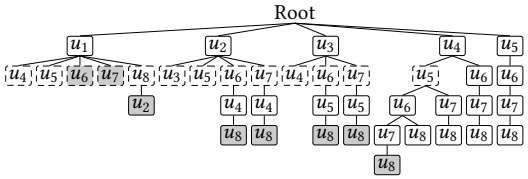
In this paper, we focus on the k-defective clique, which allows a subgraph to miss up to k edges to be a clique. Given a graph $G = (V, E)$ with n vertices and m edges, a set S of vertices in G is a k-defective clique if the subgraph induced by S has at least $\binom{|S|}{2} - k$ edges. For example, consider the graph G in Figure 2a. The set $S = \{u_4, u_5, u_6, u_7, u_8\}$ of vertices is a 1-defective clique, as the subgraph induced by S has $\binom{5}{2} - 1 = 9$ edges. The *size* of a k-defective clique is defined by the number of vertices it contains. A k-defective clique S is called *maximal* if there is no other k-defective clique including S and is called *maximum* if it has the largest size among all k-defective cliques; it is obvious that a maximum k-defective clique is also a maximal k-defective clique.

The analysis of real-world networks frequently involves the task of identifying defective cliques of large size. The missing edges in a defective clique can serve as a basis for predicting missing interactions or relationships within noisy networks. For instance, Yu et al. [88] proposed a method to predict missing interactions in protein-protein interaction (PPI) networks by completing the missing edges in defective cliques; see Figure 1 for illustration. In addition, defective cliques have diverse applications across various domains, including community detection in social networks [37, 41], cluster identification [72], optimization of transportation systems [70], and statistical analysis in financial networks [7, 8].

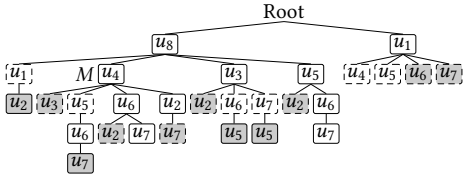
The importance of finding k-defective cliques has motivated extensive research focused on the *maximal k-defective clique enumeration* [26] and the *maximum k-defective clique search* [13, 14, 21, 25, 35, 45, 78]. The maximal k-defective clique enumeration problem is to enumerate all maximal k-defective cliques of size at least the given threshold q in a graph. The maximum k-defective clique search problem is to find a maximum k-defective clique in the graph. However, both of these



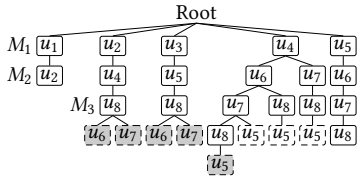
(a) G



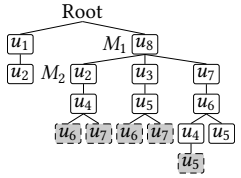
(b) A search tree of existing algorithm [26] without pivoting technique.



(c) A search tree of existing algorithm [26] with pivoting technique.



(d) A search tree of our algorithm without pivoting technique.



(e) A search tree of our algorithm with pivoting technique.

Fig. 2. An example graph G and search trees to enumerate all maximal 1-defective cliques of size at least 4 in G. The label of each search tree node represents the branching vertex that is last added to the partial solution, and the vertices from the root to each node form the partial solution of that node. A node with a dashed outline indicates that adding the branching vertex introduces missing edges in its solution, and a gray-shaded node indicates that its partial solution is maximal in G.

problems are NP-hard [78, 86]; as a result, solving them remains a computational bottleneck in the applications.

Existing algorithms and limitations. Although the two problems are closely related, the research on each problem has been conducted separately [13, 14, 21, 25, 26, 35, 45]. The commonality between approaches for the two problems is that they find *solutions* (i.e., maximum or maximal k-defective cliques) through a branch-and-bound algorithm [49].

A recursive call in the branch-and-bound algorithm maintains two disjoint sets of vertices: S and C. The set S represents a *partial solution*, which is a k-defective clique. The set C consists of *candidate vertices*, each of which can be added to S, resulting in another k-defective clique. The goal of the recursive call is to find all solutions that include S and are included by $S \cup C$. For each *branching vertex* $b \in C$, a recursive call is made to enumerate all solutions that include $S \cup \{b\}$. This procedure is repeated recursively to find all solutions.

In recursive calls, the order in which branching vertices are selected is a crucial issue. The state-of-the-art algorithms [13, 14, 25, 26] prioritize the selection of a branching vertex that introduces missing edges to the solution. However, this branching rule often causes a combinatorial explosion of small partial solutions, especially in real-world networks that are sparse and contain numerous vertices. Figure 2b illustrates the search tree generated by an algorithm in [26] for enumerating all maximal 1-defective cliques of size at least 4 in the graph G shown in Figure 2a. A total of 15 partial solutions of size 2 are generated, and 13 of them contain missing edges (e.g., $\{u_1, u_4\}$ and $\{u_2, u_3\}$).

From a theoretical perspective, existing algorithms provide upper bounds on the search space size (i.e., the number of nodes in the search tree); see Table 1. However, these search spaces are sub-optimal, as their sizes do not match the worst-case output size of maximal k-defective clique enumeration, which we prove to be $\Omega(3^{\frac{n}{3}} \cdot n^k)$ when k is a constant (Section 3.3). Moreover, as k increases, the search space size of existing algorithms approaches the trivial bound of $O(2^n)$, which corresponds to enumerating all subsets of vertices.

Table 1. Comparison of search space size (i.e., the number of search tree nodes) generated by different algorithms, and the worst-case output size of maximal k-defective clique enumeration when k is a constant.

Algorithms	# nodes	$k =$	1	2	3	4	$\cdots$
KDBB [35], KD-Club [45]	$O(2^n)$	–	–	–	–	–	$\cdots$
MADEC [21]	$O(\alpha_k^n)$	$\alpha_k =$	1.928	1.984	1.996	1.999	$\cdots$
kDC [13], Pivot [26]	$O(\beta_k^n)$	$\beta_k =$	1.839	1.928	1.966	1.984	$\cdots$
kDC2 [14]	$O(\beta_{k-1}^n)$	$\beta_{k-1} =$	1.618	1.839	1.928	1.966	$\cdots$
MDC [25]	$O(\gamma_k^n)$	$\gamma_k =$	1.466	1.755	1.889	1.948	$\cdots$
Our algorithm	$O(3^{\frac{n}{3}} \cdot n^k)$	$3^{\frac{1}{3}} =$	**1.442**	**1.442**	**1.442**	**1.442**	$\cdots$
Worst-case output size	$\Omega(3^{\frac{n}{3}} \cdot n^k)$	$3^{\frac{1}{3}} =$	1.442	1.442	1.442	1.442	$\cdots$

Contributions. In this paper, we propose theoretically and practically efficient frameworks that address both the maximal k-defective clique enumeration and the maximum k-defective clique search.

- We propose a novel branch-and-bound algorithm for maximal k-defective clique enumeration (Section 3.1). A key distinction of our algorithm is that it first generates cliques and then adds vertices that introduce missing edges, whereas previous algorithms first add vertices that introduce missing edges. Figure 2d illustrates the search tree generated by our algorithm without applying the pivoting technique. All partial solutions of size no greater than 3 are cliques (e.g., $\{u_1, u_2\}$, $\{u_2, u_4, u_8\}$, and $\{u_3, u_5, u_8\}$) and there are only six partial solutions of size 2. In this way, our algorithm effectively mitigates the combinatorial explosion of small partial solutions.
- We develop a pivoting technique combined with a new pivot selection strategy to reduce the search space (Section 3.2), and we prove that the search space of our algorithm with our pivoting technique is *worst-case optimal* when k is a constant (Section 3.3). Figure 2e illustrates a further reduced search tree generated by our algorithm with the pivoting technique incorporated, whereas Figure 2c shows the search tree of [26] with its pivoting technique. To establish worst-case optimality, we first prove that the worst-case number of maximal k-defective cliques in a graph with n vertices is $\Omega(3^{\frac{n}{3}} \cdot n^k)$ when k is a constant. We then show that the number of search tree nodes of our algorithm is $O(3^{\frac{n}{3}} \cdot n^k)$. Since all maximal k-defective cliques appear as leaf nodes in our search tree, our algorithm's search space is worst-case optimal. To the best of our knowledge, this is the first work to provide a worst-case optimal search space for relaxed clique problems.
- For real-world networks, which are usually sparse and have low degeneracy [32], we further reduce the search space size of our algorithm to $O(n \cdot 3^{\frac{\delta}{3}} \cdot (\delta\Delta)^k)$ (Section 3.4), where $\delta \leq \sqrt{m}$ is the *degeneracy*, and Δ is the maximum degree of the input graph. This improvement uses the *diameter-two property* for a k-defective clique of size at least $k + 2$ (Property 2.2).
- We propose a novel framework to solve maximum k-defective clique search using our branch-and-bound algorithm for maximal k-defective clique enumeration (Section 4). To enhance the efficiency of our framework, we also incorporate two practical techniques: (1) a technique for computing an *initial solution*, which is a k-defective clique of large size, and (2) a graph reduction technique to eliminate vertices and edges that cannot be contained in any maximum k-defective clique. Both techniques substantially reduce the search space.

We perform extensive experiments to evaluate the efficiency of the proposed algorithms and individual techniques on benchmark graph datasets (Section 5). The benchmark graph datasets

contain ten real-world graphs with more than 1 million edges and four large-scale graphs with more than 100 million edges. Our algorithms, WODC_E for maximal k-defective clique enumeration and WODC_S for maximum k-defective clique search (where WODC stands for *Worst-case Optimal Defective Clique*), are up to four orders of magnitude faster than the state-of-the-art algorithms for the respective problem. Additionally, we empirically demonstrate the effectiveness of our individual techniques.

2 Preliminaries

In this paper, we focus on undirected and simple graphs. A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges. We denote by $n = |V|$ and $m = |E|$, the number of vertices and edges of G, respectively. If $(u, v) \in E$, we say that u is *adjacent to* and a *neighbor* of v. The set of neighbors of u in G is defined as $N_G(u) = \{v \in V : (u, v) \in E\}$ and the *degree* of u in G is $d_G(u) = |N_G(u)|$. Given a vertex subset S of V, the *induced subgraph* $G[S]$ is the subgraph of G whose vertex set is S and whose edge set consists of all the edges in E with both endpoints in S.

A vertex subset S of V is a *clique* in G if every pair of vertices is adjacent to each other (i.e., $G[S]$ has $\binom{|S|}{2}$ edges). In this paper, we focus on a k-defective clique, which was first proposed in [88] and has been receiving significant attention recently [13, 14, 21, 25, 26, 35, 45].

Definition 2.1 (k-defective clique). Given a graph $G = (V, E)$ and an integer k, a vertex subset S of V is a *k-defective clique* in G if $G[S]$ misses at most k edges (i.e., $G[S]$ has at least $\binom{|S|}{2} - k$ edges).

We now present two properties of the k-defective clique, which are essential for explaining our algorithm.

Property 2.1 (Hereditary). *For any k-defective clique S in G, every subset H of S is also a k-defective clique in G.*

By Property 2.1, we can verify the maximality of a k-defective clique S by checking that there is no vertex $u \in V \setminus S$ such that $S \cup \{u\}$ is also a k-defective clique in G.

Property 2.2 (Diameter-two property [21]). *For any k-defective clique S in G with $|S| \geq k + 2$, its diameter is at most two (i.e., any two non-adjacent vertices must have a common neighbor in S).*

By Property 2.2, any k-defective clique of size greater than or equal to $k + 2$ must be both connected and dense. It is worth noting that in real-world applications, we are often interested in identifying k-defective cliques of large size that are connected and dense [23, 26]. Thus, it is natural to focus on finding such k-defective cliques.

To facilitate the presentation, we introduce additional notations. We denote the set of common neighbors of a vertex subset $S \subseteq V$ in G by $N_G(S) = \bigcap_{u \in S} N_G(u)$. We say that a graph $\overline{G} = (V, \overline{E})$ is a *complement graph* of $G = (V, E)$ if $\overline{E} = \{(u, v) \in V \times V : (u, v) \notin E \text{ and } u \neq v\}$. If $(u, v) \in \overline{E}$, we say that u is a *non-neighbor* of v. The set of non-neighbors of u in G is denoted by $\overline{N}_G(u) = \{v \in V : (u, v) \in \overline{E}\}$. We denote the set of edges in the induced subgraph $\overline{G}[S]$ by $\overline{E}_G(S)$. For brevity, we omit the subscript G from the notations when the context is clear. For any vertex subsets S and C of V and a vertex u in V, we denote by $N_C(u)$ the set of u's neighbors in C, by $N_C(S)$ the set of common neighbors of S that are in C, and by $\overline{N}_C(u)$ the set of u's non-neighbors in C.

Degeneracy ordering and s-core. We describe the concepts of degeneracy ordering and the s-core.

Definition 2.2 (Degeneracy ordering [56]). Given a graph $G = (V, E)$, an ordering $(v_1, v_2, \ldots, v_n)$ of its vertices V is a *degeneracy ordering* if, for each $1 \leq i \leq n$, v_i is a vertex with the smallest degree in the subgraph of G induced by vertices $\{v_i, v_{i+1}, \ldots, v_n\}$.

Definition 2.3 (s-core [68]). Given a graph $G = (V, E)$ and an integer s, the *s-core* of G is the maximal subgraph g of G in which every vertex in g has degree $d_g(u) \geq s$.

Table 2. Frequently used notations.

Notation	Meaning
$G = (V, E)$	input graph with vertex set V and edge set E
n, m	number of vertices and edges in G
δ, Δ	degeneracy and maximum degree of G
$S \subseteq V$	(partial) solution, which is k-defective clique
$C, X \subseteq V \setminus S$	sets of candidate vertices
$N_C(u)$	set of u's neighbors that are in C
$N_C(S)$	set of common neighbors of S that are in C
$\overline{N}_C(u)$	set of u's non-neighbors that are in C
$\overline{N}_C[u]$	$\overline{N}_C(u) \cup \{u\}$
$\overline{E}(S)$	set of edges in the subgraph of $\overline{G}$ induced by S

The degeneracy ordering and s-cores can be computed in $O(m)$ time [13, 56] by iteratively removing a vertex with the smallest degree from G and appending it to the end of the ordering. The largest s such that G includes a non-empty s-core is known as the *degeneracy* of G, denoted δ.

Given a degeneracy ordering $(v_1, v_2, \ldots, v_n)$ of G, we say that v_j is a *forward neighbor* of v_i if v_j is a neighbor of v_i and $i < j$. We denote the set of forward neighbors of v_i by $N_G^+(v_i) = N_G(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$. The number of forward neighbors is upper bounded by the degeneracy δ of G, i.e., $|N^+(v_i)| \leq \delta$ for every $1 \leq i \leq n$ [16].

Graph coloring. Given a graph $G = (V, E)$, a mapping $\chi : V \to \mathbb{N}$ is called a *coloring* of G if $\chi(u) \neq \chi(v)$ for every edge $(u, v) \in E$. The number of distinct colors used by χ is denoted by $|\chi|$.

Given a degeneracy ordering, one way to compute a coloring of a graph is to assign colors greedily to the vertices in reverse order, assigning each vertex the smallest available value [11]. In this way, we can obtain a coloring where the number of distinct colors $|\chi|$ is upper bounded by $\delta + 1$, where δ is the degeneracy of G.

Frequently used notations are summarized in Table 2.

2.1 Problem Statements

In this paper, we address two fundamental problems in computing k-defective cliques of large size in a given graph.

Maximal k-Defective Clique Enumeration. Given a graph $G = (V, E)$ and positive integers k and q (with $q \geq k + 2$), the maximal k-defective clique enumeration problem is to enumerate all maximal k-defective cliques of size at least q.

Maximum k-Defective Clique Search. Given a graph $G = (V, E)$ and a positive integer k, the maximum k-defective clique search problem is to compute a largest k-defective clique in G.

2.2 Related Works

Maximal (maximum) clique enumeration (search). The maximal clique enumeration problem has been extensively studied in recent decades. The most notable algorithm for this problem is the Bron-Kerbosch (BK) algorithm [10], a seminal backtracking algorithm that incorporates a pivoting technique to reduce the search space. Tomita et al. [77] improved upon this by introducing a carefully designed pivot selection strategy, achieving a worst-case optimal time complexity of $O(3^{\frac{n}{3}})$, as there exists a graph with $\Theta(3^{\frac{n}{3}})$ maximal cliques. Eppstein et al. [32] proposed a variant of BK algorithm based on degeneracy ordering, achieving a time complexity of $O(n \cdot 3^{\frac{\delta}{3}})$. In addition

to these theoretical advancements, various optimization techniques have been developed [22, 29, 30, 51, 59, 66, 81]. The maximum clique search problem also has been extensively studied both from a theoretical perspective [44, 63, 64, 73] and a practical perspective [11, 52, 55, 65, 67, 75]. However, these algorithms cannot be directly applied to our defective clique problems.

Maximal (maximum) relaxed clique enumeration (search). In real-world applications, the definition of a clique is often too restrictive. To address this, various relaxations have been proposed, including quasi-cliques [1], k-plexes [5], k-clubs [9], and k-bundles [61]. To solve these problems, numerous practical algorithms have been developed [17, 23, 24, 27, 36, 83, 92]. Some of these problems can be formulated as integer linear programs (ILP) [69, 72, 79]. However, ILP-based approaches are known to be less scalable than branch-and-bound methods, as demonstrated in [21]. Recently, branch-and-bound algorithms with search space sizes smaller than the trivial $O(2^n)$ bound have been proposed for each relaxation [14, 26, 54, 82, 85, 89, 93]. However, none of these algorithms have been proven to be worst-case optimal.

Graph mining systems. Numerous graph mining systems have been developed to efficiently support subgraph mining tasks, including the discovery of relaxed cliques such as quasi-cliques and defective cliques.

Arabesque [74] is a distributed graph mining platform that automates large-scale subgraph exploration through a high-level API, supporting tasks such as frequent subgraph mining, motif counting, and clique enumeration. Peregrine [42] introduces a pattern-aware exploration paradigm, which leverages the semantics of user-defined patterns to guide the exploration process and minimize unnecessary computations.

T-thinker [46] and Contigra [20] are graph mining frameworks that support quasi-clique mining and can be extended to other relaxed clique models. T-thinker employs pruning strategies specifically tailored for quasi-cliques. Contigra is designed to handle pattern mining (including quasi-cliques) with containment constraints such as maximality. It introduces search space pruning techniques to eliminate redundant computations that arise during maximality checking, and properties such as hereditary can also be leveraged within the exploration strategy to further enhance efficiency.

3 A New Branch-and-Bound Algorithm

In this section, we present our novel branch-and-bound algorithm for solving maximal k-defective clique enumeration.

3.1 A Novel Clique-First Approach

We first present our new approach which first generates cliques and then adds missing edges to them. The main observation underlying our clique-first approach is as follows.

Observation 3.1. *A k-defective clique of size q must contain a clique of size $q - k$.*

Proof. Let S be a k-defective clique of size q. By definition, S contains at most k missing edges. Consider the set of vertices incident to these missing edges. By removing one endpoint from each missing edge, we remove at most k vertices from S, and the remaining vertices form a clique of size at least $q - k$. Therefore, a k-defective clique of size q contains a clique of size $q - k$. □

To briefly illustrate the idea of our approach, consider a graph with $n = 100$ vertices and $k = 3, q = 10$. Then any 3-defective clique of size $q = 10$ must have a clique of size 7. The remaining task is to choose 3 vertices (from 93 vertices) with at most 3 missing edges, which has a much smaller search space than choosing 10 vertices from 100 vertices with at most 3 missing edges.

Algorithm 1: Our branch-and-bound algorithm

Input : A graph $G = (V, E)$ and two integers k, q
Output: Every maximal k-defective clique in G of size $\geq q$

1 bnb$(\emptyset, V, \emptyset)$;
2 **Procedure** bnb(S, C, X)
3 **if** $C \cup X = \emptyset$ *and* $|S| \geq q$ **then output** S;
4 If $N_C(S) = \emptyset$, set $B \leftarrow C$. Otherwise, select a pivot vertex p from $N_C(S)$ with the fewest non-neighbors in $N_C(S)$, then set $B \leftarrow \overline{N}_C[p]$;
5 **while** $B \neq \emptyset$ **do**
6 **if** ***UB1*** *of* (S, C) *is less than* q **then return**;
7 $b \leftarrow$ a vertex in B adjacent to all vertices in S. If there is no such a vertex, then choose an arbitrary vertex from B;
 // a vertex b that introduces a missing edge is selected only when $N_B(S) = \emptyset$.
8 $C', X' \leftarrow$ refine(S, C, X, b);
9 bnb$(S \cup \{b\}, C', X')$;
10 $B \leftarrow B \setminus \{b\}$; $C \leftarrow C \setminus \{b\}$; $X \leftarrow X \cup \{b\}$;
11 **Procedure** refine(S, C, X, b)
12 $C' \leftarrow \{u \in C \setminus \{b\} : S \cup \{b, u\}$ is a k-defective clique$\}$;
13 $X' \leftarrow \{u \in X : S \cup \{b, u\}$ is a k-defective clique$\}$;
14 **return** C', X';

When combined with clique listing, which can be done much faster (also has much smaller search space) than defective clique listing, our clique-first approach leads to a substantially smaller search space.

We now formally introduce our branch-and-bound algorithm. We refer to an output (i.e., a maximal k-defective clique) of the problem as a *solution* and a vertex subset of a solution which is a k-defective clique and possibly non-maximal as a *partial solution*. Given a partial solution S, we say that a vertex $u \in V \setminus S$ is a *candidate vertex* if $S \cup \{u\}$ is also a k-defective clique.

Algorithm 1 shows the pseudocode of our branch-and-bound algorithm. A recursive call to our branch-and-bound algorithm takes as input a tuple of three disjoint sets of vertices (S, C, X), where $S \subseteq V$ is a partial solution, which is a k-defective clique, and $C \cup X = \{u \in V \setminus S : S \cup \{u\}$ is a k-defective clique$\}$ is the set of candidate vertices. We say that the tuple (S, C, X) is an *instance* of the recursive call. The candidate vertices in C have not yet been explored, while those in X have already been explored in previous recursions. That is, the vertices in X must not be added to S in its recursive calls to prevent the duplicate output of the same k-defective cliques. The goal of the recursive call (S, C, X) is to output every maximal k-defective clique that includes S and is included by $S \cup C$.

For each recursive call (S, C, X), the algorithm checks the maximality of S by verifying whether $C \cup X = \emptyset$ and, according to Property 2.1, outputs S as a solution if this condition holds and $|S| \geq q$ (Line 3). Then, a set of branching vertices $B \subseteq C$ is computed (Line 4); for now, suppose that B is set to C, i.e., the pivoting technique (details will be provided in Section 3.2) is not applied. For each branching vertex $b \in B$, the recursive call $(S \cup \{b\}, C', X')$ is made, where $C' \subseteq C \setminus \{b\}$, $X' \subseteq X$, and $C' \cup X' = \{u \in V \setminus (S \cup \{b\}) : S \cup \{b, u\}$ is a k-defective clique$\}$ is the refined set of candidate

vertices. The recursive call $(S \cup \{b\}, C', X')$ enumerates every maximal k-defective clique that includes $S \cup \{b\}$ and is included by $S \cup \{b\} \cup C'$. After the recursive call, b is removed from B and moved from C to X (Lines 7–10).

Since we aim to find k-defective cliques of size at least q, we compute an upper bound on the size of a k-defective clique that includes S and is included by $S \cup C$ (Line 6). Whenever this upper bound is less than q, the remaining recursive calls are pruned, as they cannot generate any solution of size at least q. This upper-bound-based pruning is applied to (S, C) in Line 6, in order to reduce the search space efficiently.

Our algorithm uses the graph coloring-based upper bounding technique **UB1** proposed by Lijun Chang [13]. Given a tuple (S, C), **UB1** returns $|S| + c + r$, where $c \leq |N_C(S)|$ and $r \leq k - |\overline{E}(S)|$ are both non-negative integers.

In Line 7, our algorithm selects a branching vertex from B that is adjacent to all vertices in S, if such a vertex exists. That is, our algorithm first branches on vertices that do not introduce missing edges to the partial solution. A missing edge is introduced to the partial solution only after all such vertices have been processed. Thus, our algorithm first generates cliques and then adds missing edges.

Example 3.1. We are given the graph G in Figure 2a, with $k = 1$ and $q = 4$. Figure 2d shows the search tree of Algorithm 1 without the pivoting technique. At the root node, the first recursive call, $M_1 = (\{u_1\}, \{u_2, u_3, \ldots, u_8\}, \emptyset)$, is made. The node M_1 first branches with the branching vertex u_2, as it is adjacent to the vertex u_1 in S. This generates the next recursive call $M_2 = (\{u_1, u_2\}, \{u_3, u_4, u_8\}, \emptyset)$. However, the recursive calls of M_2 are pruned since the upper bound of $(\{u_1, u_2\}, \{u_3, u_4, u_8\})$ is $|S| + c + r = 2 + 0 + 1 = 3$, which is less than $q = 4$. After backtracking to M_1, u_2 is moved from C to X, and the upper bound of $(\{u_1\}, \{u_3, u_4, \ldots, u_8\})$ is computed in Line 6. Since the upper bound is 3, the remaining recursive calls of M_1 are pruned. After backtracking to the root node, a second recursive call, $(\{u_2\}, \{u_3, u_4, \ldots, u_8\}, \{u_1\})$, is generated, and the process continues down to node $M_3 = (\{u_2, u_4, u_8\}, \{u_6, u_7\}, \emptyset)$. In M_3, two recursive calls, $(\{u_2, u_4, u_8, u_6\}, \emptyset, \emptyset)$ and $(\{u_2, u_4, u_8, u_7\}, \emptyset, \emptyset)$, are made, and two solutions are output in Line 3. In this way, the search tree outputs five solutions (gray-shaded). In this search tree, all partial solutions of size at most 3 are cliques, as any partial solution of size at most 3 that contains missing edges is pruned in Line 6.

We note that there is still room for improvement in the search space of Algorithm 1: in the search tree of Figure 2d, nearly all subsets of the 1-defective clique $\{u_4, u_5, u_6, u_7, u_8\}$ are generated, but most are non-maximal and therefore unnecessary. To address this issue, we introduce a new pivoting technique for defective cliques that significantly reduces the search space.

3.2 A New Pivoting Technique for Defective Cliques

A technique called *pivoting* was originally developed to reduce the search space of the BK algorithm [10] to enumerate all maximal cliques. Tomita et al. [77] proved that the BK algorithm with the pivot selection strategy that minimizes the number of branches at each node leads to a worst-case optimal algorithm for maximal clique enumeration.

However, to the best of our knowledge, no other worst-case optimal backtracking algorithms have been developed for enumerating maximal relaxed cliques (Section 2.2). This presents two challenges: (1) identifying the worst-case output size, and (2) designing a backtracking algorithm whose search space matches the worst-case output size.

To address challenge (2), it is essential to develop a new pivoting technique for maximal k-defective clique enumeration, as the original pivoting technique can be applied only for maximal clique enumeration. Additionally, a carefully designed criterion for the pivot selection is necessary. We present a pivoting technique, along with a new pivot selection strategy for our problem. This

enables our algorithm to explore a worst-case optimal search space when k is a constant. Challenge (1) will be addressed in Section 3.3 with the time complexity analysis.

The pivoting technique. We first describe the main idea of the pivoting technique. Consider an instance $I = (S, C, X)$ where $N_C(S) \neq \emptyset$, and let p be a vertex in $N_C(S)$ (i.e., p is adjacent to all vertices in S). We refer to p as the *pivot vertex*, and the criterion for selecting it will be explained shortly. Each partial solution S' generated from I, i.e., which includes S and is included by $S \cup C$, falls into one of the following two types: (i) all vertices in S' are adjacent to p, or (ii) S' contains p or a non-neighbor of p. A partial solution of type (i) is not maximal, as adding p results in another k-defective clique. Thus, generating type (i) partial solutions is unnecessary. The pivoting technique ensures that every partial solution generated from I contains either p or a non-neighbor of p, thereby avoiding type (i) partial solutions. This is achieved by branching only with p and its non-neighbors.

For ease of presentation, we regard p as a non-neighbor of p, i.e., we define $\overline{N}_C[p] = \overline{N}_C(p) \cup \{p\}$. The following lemma formally introduces our pivoting technique.

Lemma 3.1 (k-defective clique pivoting). *Given an instance (S, C, X) of a recursive call where $N_C(S) \neq \emptyset$, let p be a vertex in $N_C(S)$. Every maximal k-defective clique that includes S and is included by $S \cup C$ must contain at least one vertex in $\overline{N}_C[p]$.*

Proof. Suppose not. Then, there exists a maximal k-defective clique S' that includes S but does not contain any vertex in $\overline{N}_C[p]$. Since p is in $N_C(S)$, all the vertices in S' are adjacent to p. Therefore, $S' \cup \{p\}$ is also a k-defective clique, and thus S' is not maximal. This contradiction proves the lemma. □

Line 4 of Algorithm 1 incorporates the pivoting technique into our branch-and-bound algorithm. If $N_C(S) = \emptyset$, we set $B = C$ since Lemma 3.1 cannot be applied. Otherwise (i.e., $N_C(S) \neq \emptyset$), a pivot vertex p is selected from $N_C(S)$, and we set the set of branching vertices B to $\overline{N}_C[p]$. In Lines 5–10, recursive calls are made only with the vertices in B. Lemma 3.1 guarantees that Algorithm 1 generates all solutions.

The remaining issue is determining the criterion for selecting a pivot vertex from the vertices in $N_C(S)$ (Line 4).

A new pivot selection strategy. Since we deal with the problem of enumerating k-defective cliques, the pivoting strategy of Tomita et al. [77], which is designed for maximal clique enumeration, cannot be applied directly. We present a new pivot selection strategy for our problem.

Consider an instance $I = (S, C, X)$, where a pivot vertex can be selected (i.e., $N_C(S) \neq \emptyset$). Our algorithm selects a pivot vertex p from $N_C(S)$ that has the fewest non-neighbors in $N_C(S)$, i.e.,

$$p = \arg\min_{u \in N_C(S)} |\overline{N}_C[u] \cap N_C(S)|.$$

Our algorithm has the following characteristics:

- Since our branching rule (Line 7) first makes branches with the vertices in $B = \overline{N}_C[p]$ that are adjacent to all vertices in S, it first generates cliques and then adds missing edges.
- Our pivoting technique aims to generate maximal solutions as early as possible by branching with p's non-neighbors (and delaying the addition of p's neighbors to S).
- Since our pivot selection strategy minimizes the number of non-neighbors in $N_C(S)$, it minimizes the number of branches on vertices in $N_C(S)$ (i.e., branches that do not introduce missing edges).

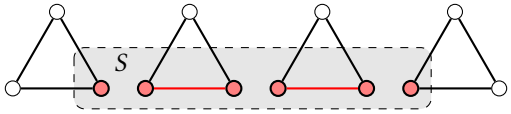


Fig. 3. The complement graph of the Moon-Moser graph with 12 vertices. The set S, consisting of six vertices and containing two missing edges (highlighted in red), is a maximal 2-defective clique in the Moon-Moser graph.

These three characteristics together lead to the worst-case optimal search space (Section 3.3). Therefore, this is the first algorithm that is proven to be worst-case optimal among all algorithms for relaxed clique problems.

Example 3.2. Figure 2e illustrates the search tree generated by Algorithm 1 with our proposed pivoting technique and pivot selection strategy. At the root node, $N_C(S) = \{u_1, \ldots, u_8\}$ and the vertex u_8 is selected by our pivot selection strategy, since u_8 has the fewest non-neighbors in $N_C(S) = \{u_1, \ldots, u_8\}$. As a result, B is set to $\overline{N}_C[u_8] = \{u_1, u_8\}$ (Line 4). Since the first branch with u_1 works similarly to Example 3.1, we explain the second branch, $M_1 = (\{u_8\}, \{u_2, u_3, u_4, u_5, u_6, u_7\}, \{u_1\})$. Here, $N_C(S) = \{u_2, u_3, u_4, u_5, u_6, u_7\}$, and the numbers of non-neighbors in $N_C(S)$ are 5, 5, 3, 3, 3, and 3 for $u = u_2, u_3, u_4, u_5, u_6$, and u_7, respectively. We select u_7 as the pivot vertex (we break the tie by selecting the vertex with the highest ID) and B is set to $\overline{N}_C[u_7] = \{u_2, u_3, u_7\}$ in Line 4. It first generates the branch $M_2 = (\{u_8, u_2\}, \{u_3, u_4, u_5, u_6, u_7\}, \{u_1\})$. In the subtree rooted at M_2, two solutions are output.

Comparison to existing pivoting technique. Dai et al. [26] also proposed a generalized pivoting technique that can be applied to algorithms for enumerating maximal substructures satisfying the hereditary property. However, this pivoting technique has two limitations: (i) its pivoting technique is applied only when every vertex in C is adjacent to all vertices in S (i.e., $C = N_C(S)$), and (ii) it does not provide criteria for selecting a pivot vertex.

Figure 2c shows the search tree of [26] with its pivoting technique. The node $M = (\{u_8, u_4\}, \{u_2, u_3, u_5, u_6, u_7\}, \emptyset)$ first branches with u_3 and u_5 without applying the pivoting technique since they each have a non-neighbor in $\{u_8, u_4\}$. Then, the pivoting technique is applied to $(\{u_8, u_4\}, \{u_2, u_6, u_7\}, \{u_3, u_5\})$. Since there are no criteria for pivot selection, any vertex from $\{u_2, u_6, u_7\}$ can be selected as a pivot, potentially leading to sub-optimal search space.

3.3 Theoretical Analysis

We now show that the search space of Algorithm 1 is worst-case optimal when k is a constant. Since real-world applications typically require small values of k in obtaining k-defective cliques [26], it is reasonable to assume that k is a constant. We first prove the lower bound $\Omega(3^{\frac{n}{3}} \cdot n^k)$ of the worst-case output size for maximal k-defective clique enumeration. Then, we prove the upper bound $O(3^{\frac{n}{3}} \cdot n^k)$ by analyzing the number of search tree nodes of Algorithm 1 in the worst case.

The Moon-Moser graph [57] is a graph that includes the largest number of maximal cliques among all graphs of the same size. Figure 3 shows the complement graph of the Moon-Moser graph with 12 vertices. When the size n of the Moon-Moser graph is a multiple of 3, the complement graph of the Moon-Moser graph consists of $\frac{n}{3}$ disjoint triangles.

The following theorem provides the lower bound on the worst-case output size.

Theorem 3.1. *The worst-case output size of maximal k-defective clique enumeration is $\Omega(3^{\frac{n}{3}} \cdot n^k)$ when k is a constant.*

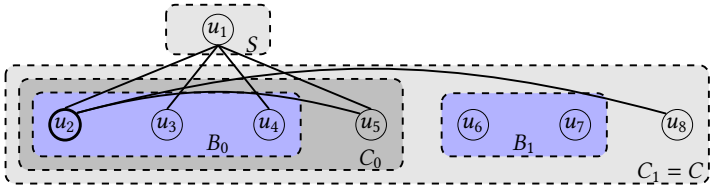


Fig. 4. A diagram illustrating the definitions of C_i and B_i for $I = (S, C, X)$, where $S = \{u_1\}$, $C = \{u_2, \ldots, u_8\}$, and $\kappa = k = 1$. The vertex u_2 is selected as the pivot vertex, and B is set to $\overline{N}_C[u_1] = \{u_2, u_3, u_4, u_6, u_7\}$.

Proof. We prove that the Moon-Moser graph with n vertices, where n is a multiple of 3 and $\frac{n}{3} \geq k$, contains at least $3^{\frac{n}{3}} \cdot \binom{n/3}{k}$ maximal k-defective cliques; note that $3^{\frac{n}{3}} \cdot \binom{n/3}{k} = \Omega(3^{\frac{n}{3}} \cdot n^k)$ when k is a constant. To construct a maximal k-defective clique in the Moon-Moser graph, we first select k triangles in the complement graph of the Moon-Moser graph. Then, we choose a missing edge from each of the k selected triangles and one vertex from each of the remaining $\frac{n}{3} - k$ triangles; for example in Figure 3, we first choose two missing edges from the second and third triangles and then choose two vertices from the remaining triangles. The resulting set of vertices forms a maximal k-defective clique in the Moon-Moser graph, since the set contains exactly k missing edges, and adding another vertex to the set introduces a missing edge. The total number of such maximal k-defective cliques is $3^{\frac{n}{3}} \cdot \binom{n/3}{k}$. □

Now, we prove an upper bound on the number of nodes in the search tree generated by Algorithm 1. Since the outputs (i.e., maximal k-defective cliques) appear at leaf nodes in the search tree, the output size is not greater than the number of search tree nodes. Thus, an upper bound on the number of search tree nodes is an upper bound on the output size.

To formally establish the upper bound, we introduce some notations. Let $\mathcal{T}$ be the search tree generated by Algorithm 1. Suppose that we are given an instance (S, C, X) of $\mathcal{T}$. We denote $T(S, C, X)$ by the number of nodes in the subtree of $\mathcal{T}$ rooted at (S, C, X). Let κ be the number of missing edges that can be added to S (i.e., $\kappa = k - |\overline{E}(S)|$). For every integer $0 \leq i \leq \kappa$, let C_i be the set of vertices in C that introduce at most i missing edges when added to S (i.e., $C_i = \{u \in C : |\overline{N}_S(u)| \leq i\}$); see Figure 4 for illustration (for now, ignore B_i). For κ non-negative integers $n_1, \ldots, n_\kappa$, we define

$$f^\kappa(n_1, \ldots, n_\kappa) = 1 + n_\kappa + n_{\kappa-1}n_\kappa + \cdots + n_1 \cdots n_\kappa = 1 + \sum_{i=1}^{\kappa} \prod_{j=i}^{\kappa} n_j.$$

We prove an upper bound of $T(S, C, X)$ in the following lemma (a complete proof is provided in the full version [43]).

Lemma 3.2. *For any instance $I = (S, C, X)$ of the search tree $\mathcal{T}$,*

$$T(S, C, X) \leq 2 \cdot 3^{\frac{|C_0|}{3}} \cdot f^\kappa(|C_1|, \ldots, |C_\kappa|). \tag{1}$$

Proof Sketch. We prove the inequality by induction. For the base case where I is a leaf node, it is trivial that $T(S, C, X) = 1 \leq 2 \cdot 3^{\frac{|C_0|}{3}} \cdot f^\kappa(|C_1|, \ldots, |C_\kappa|) = 2$ because $C_0 \subseteq C_1 \subseteq \cdots \subseteq C_\kappa = C = \emptyset$ (that is, $|C_0| = |C_1| = \cdots = |C_\kappa| = 0$). We now consider an arbitrary non-leaf node I. There are two cases as follows.

Case I: $C_0 = \emptyset$. In this case, adding a candidate vertex in C to the partial solution always increases the number of missing edges. We have $T(S, C, X) \leq f^\kappa(|C_1|, \ldots, |C_\kappa|)$ because the height of the subtree rooted at I is at most κ, and the number of nodes at depth i ($1 \leq i \leq \kappa$) in the subtree is at most $|C_{\kappa-i+1}| \cdots |C_\kappa|$.

Case II: $C_0 \neq \emptyset$. When $\kappa = 0$ (i.e., no missing edge is allowed), the problem degenerates to maximal clique enumeration. In this case, our pivot selection strategy coincides with that of Tomita et al. [77], and therefore we obtain $T(S, C, X) \leq 2 \cdot 3^{\frac{|C_0|}{3}}$ [34]. Thus, we consider the case $\kappa > 0$.

Since $C_0 = N_C(S)$, our pivot selection strategy selects a pivot vertex $p \in N_C(S)$ that has the fewest non-neighbors in $N_C(S)$, and then sets B to $\overline{N}_C[p]$ (Line 4). For every integer $0 \leq i \leq \kappa$, let B_i be the set of vertices in B that introduce exactly i missing edges when added to S (i.e., $B_i = \{u \in B : |\overline{N}_S(u)| = i\}$); see Figure 4 for illustration. For brevity, we denote $|C_i| = n_i$ and $|B_i| = y_i$ for $0 \leq i \leq \kappa$.

We refer to an instance of a recursive call from I as a *child instance* of I. We assume as the induction hypothesis that inequality (1) holds for every child instance of I, then we now show that the inequality holds for I where $C_0 \neq \emptyset$ and $\kappa > 0$. For a child instance $I' = (S' = S \cup \{b\}, C', X')$ of I, we obtain the following inequality by the induction hypothesis, depending on the branching vertex b.

(a) When $b = p$ (i.e., b is the pivot vertex), we have $T(S', C', X') \leq 2 \cdot 3^{\frac{n_0 - y_0}{3}} \cdot f^{\kappa}(n_1 - y_1 - 1, \ldots, n_{\kappa} - y_{\kappa} - 1)$. This follows from the fact that p is not adjacent to any vertex in $B = B_0 \cup B_1 \cup \cdots \cup B_{\kappa}$.

(b) When $b \in B_0 \setminus \{p\}$, $T(S', C', X') \leq 2 \cdot 3^{\frac{n_0 - y_0}{3}} \cdot f^{\kappa}(n_1, \ldots, n_{\kappa})$. This follows from our pivot selection strategy (Line 4). There are $y_0 - 1$ such branching vertices (note that $p \in B_0$).

(c) When $b \in B_i$ for $1 \leq i \leq \kappa$, $T(S', C', X') \leq 2 \cdot 3^{\frac{n_0 - y_0}{3}} \cdot f^{\kappa - i}(n_1 - y_0, \ldots, n_{\kappa - i} - y_0)$. This follows from our branching rule (Line 7). There are y_i such branching vertices.

Combining (a)–(c), we obtain the following inequality:

$$
\begin{aligned}
T(S, C, X) &= \sum_{\text{every child instance } (S', C', X') \text{ of } I} T(S', C', X') + 1 \\
&\leq 2 \cdot 3^{\frac{n_0 - y_0}{3}} \cdot f^{\kappa}(n_1 - y_1 - 1, \ldots, n_{\kappa} - y_{\kappa} - 1) \\
&+ (y_0 - 1) \cdot 2 \cdot 3^{\frac{n_0 - y_0}{3}} \cdot f^{\kappa}(n_1, \ldots, n_{\kappa}) \\
&+ \sum_{i=1}^{\kappa} y_i \cdot 2 \cdot 3^{\frac{n_0 - y_0}{3}} \cdot f^{\kappa - i}(n_1 - y_0, \ldots, n_{\kappa - i} - y_0) + 1.
\end{aligned}
$$

Now it suffices to show that the right-hand side of the above inequality is bounded by $2 \cdot 3^{\frac{|C_0|}{3}} \cdot f^{\kappa}(|C_1|, \ldots, |C_{\kappa}|) = 2 \cdot 3^{\frac{n_0}{3}} \cdot f^{\kappa}(n_1, \ldots, n_{\kappa})$. This follows from the inequality $y_0 \cdot 3^{-\frac{y_0}{3}} \leq 1$ [77], together with elementary algebraic manipulations. □

From Lemma 3.2, we obtain an upper bound on the number of search tree nodes generated by Algorithm 1.

Theorem 3.2. *Given a graph $G = (V, E)$ with n vertices and a non-negative integer k, the number of search tree nodes generated by Algorithm 1 is $O(3^{\frac{n}{3}} \cdot n^k)$.*

Proof. Consider the root node $(S, C, X) = (\emptyset, V, \emptyset)$ of the search tree. Since $S = \emptyset$, we have $C_0 = C_1 = \cdots = C_k = V$. Therefore, $T(\emptyset, V, \emptyset) \leq 2 \cdot 3^{\frac{n}{3}} \cdot f^{k}(n, \ldots, n) = 2 \cdot 3^{\frac{n}{3}} \cdot (1 + n + n^2 + \cdots + n^{\kappa}) = O(3^{\frac{n}{3}} \cdot n^k)$. □

Combined with Theorem 3.1, the search space size of Algorithm 1 matches the worst-case output size $\Omega(3^{\frac{n}{3}} \cdot n^k)$ of the problem when k is a constant.

From Theorem 3.2, the following theorem holds since each search tree node takes $O(m)$ time (a formal proof is in the full version [43]).

Theorem 3.3. *Algorithm 1 runs in $O(m \cdot 3^{\frac{n}{3}} \cdot n^k)$ time.*

Algorithm 2: Our branch-and-bound algorithm utilizing the diameter-two property

Input : A graph $G = (V, E)$ and two integers $k, q \geq k + 2$
Output: Every maximal k-defective clique in G of size $\geq q$

1 Let $(v_1, v_2, \ldots, v_n)$ be a degeneracy ordering of G;
2 **foreach** $v_i \in V$ **do**
3 $\quad V_i \leftarrow N(v_i) \cup \bigcup_{v \in N^+(v_i)} N(v)$;
4 $\quad C_i \leftarrow V_i \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$;
5 $\quad X_i \leftarrow V_i \cap \{v_1, v_2, \ldots, v_{i-1}\}$;
6 $\quad \mathsf{bnb}(\{v_i\}, C_i, X_i)$;

3.4 Using Diameter-Two Property

In this subsection, we describe how we can further reduce the exponent of the time complexity of our algorithm based on the diameter-two property (Property 2.2) when $q \geq k + 2$.

The main idea is to decompose the original problem into small subproblems; for each $v_i \in V$, enumerate every solution whose smallest vertex (with respect to a degeneracy ordering) is v_i and whose diameter is at most two, as in [23]. Algorithm 2 presents this approach. We first compute a degeneracy ordering $(v_1, v_2, \ldots, v_n)$ of G in Line 1. For each vertex $v_i \in V$, we compute the set $V_i = N(v_i) \cup \bigcup_{v \in N^+(v_i)} N(v)$ (Line 3). This set contains all solutions of diameter at most two, each of which contains v_i and a forward neighbor of v_i [27]. By the diameter-two property, all vertices in any solution of the subproblem, except for v_i, must exist in $V_i \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ [23]. Therefore, we set $C_i = V_i \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ (Line 4). Then, we set $X_i = V_i \cap \{v_1, v_2, \ldots, v_{i-1}\}$ (Line 5). Finally, a recursive call $(\{v_i\}, C_i, X_i)$ is made (Line 6) to enumerate all solutions of the subproblem.

Lemma 3.3. *The number of search tree nodes generated by Algorithm 2 is bounded by* $O(n \cdot 3^{\frac{\delta}{3}} \cdot (\delta\Delta)^k)$.

Proof. We prove the lemma by showing that the search space size of each subproblem is bounded by $O(3^{\frac{\delta}{3}} \cdot (\delta\Delta)^k)$. This follows from the facts that $|N_{C_i}(v_i)| \leq \delta$ and $|C_i| \leq \delta\Delta$ [14] for all $1 \leq i \leq n$. Then, applying Lemma 3.2, the overall bound follows. □

Since each search tree node takes $O((\delta\Delta)^2)$ time, the following theorem holds (a formal proof is given in the full version [43]).

Theorem 3.4. *Algorithm 2 runs in* $O(n \cdot 3^{\frac{\delta}{3}} \cdot (\delta\Delta)^{k+2})$ *time.*

4 Maximum k-Defective Clique Search

In this section, we introduce our framework for solving maximum k-defective clique search by using our branch-and-bound algorithm for maximal k-defective clique enumeration. This framework also incorporates new practical techniques to reduce the search space: (1) a technique for computing a large initial solution, and (2) a graph reduction technique for eliminating unpromising vertices and edges from the input graph. Due to the page limit, we provide the running examples in the full version [43].

4.1 Our Framework

Algorithm 3 presents our framework. Given an input graph G and an integer k, it first computes an *initial solution* (Section 4.2.1), which is a k-defective clique, and sets S^* to this initial solution. Throughout the execution, S^* maintains the largest k-defective clique found so far. Then, q is set to $|S^*| + 1$ (Line 1); since we have already found S^*, we aim to find a k-defective clique of size at least

Algorithm 3: Our framework for maximum k-defective clique search

Input : A graph $G = (V, E)$ and an integer k
Output: A maximum k-defective clique in G

1 $S^* \leftarrow$ `initial-solution`(G, k); $q \leftarrow |S^*| + 1$;
2 $g \leftarrow$ `reduce` (G, k, q);
3 **if** g *is empty* **then return** S^*;
4 **if** $q \geq k+2$ **then** run Algorithm 2 with g, k, and q, in which `bnb` is replaced with `bnb-maximum`;
5 **else** run Algorithm 1 with g, k, and q, in which `bnb` is replaced with `bnb-maximum`;
6 **return** S^*;
7 **Procedure** `bnb-maximum`(S, C, X)
8 **if** $|S| \geq q$ **then**
9 $S^* \leftarrow S$; $q \leftarrow |S^*| + 1$;
10 Same as Lines 4–10 in Algorithm 1, except that `bnb` in Line 9 is replaced with `bnb-maximum`;

$q = |S^*| + 1$. The algorithm then computes a *reduced graph* g (Section 4.2.2), by removing vertices and edges from G that cannot be contained in any k-defective clique of size at least q (Line 2). If g is empty (i.e., it contains no vertices and edges), then G does not contain any k-defective clique of size at least q. Therefore, the algorithm returns S^* as a maximum k-defective clique (Line 3). If g is not empty, the framework finds a maximum k-defective clique using our branch-and-bound algorithm (Lines 4–5). During the branch-and-bound search, whenever a k-defective clique S of size at least q (that is, larger than S^*) is found, we update S^* to S and set q to $|S^*| + 1$ (Lines 8–9). This approach is highly efficient because a larger q allows us to prune more of the search space.

4.2 Initial Solution and Graph Reduction

We now present our practical techniques for initial solution and graph reduction. To introduce our techniques, we first present the concepts of colorful s-core and colorful degeneracy ordering, which are introduced in [60, 91].

Definition 4.1 (Colorful degree). Given a graph $G = (V, E)$ and a coloring χ of G, the *colorful degree* of $u \in V$ in G with respect to χ, denoted by $d_G^\chi(u)$, is the number of distinct colors among u's neighbors. Formally, $d_G^\chi(u) = |\{\chi(v) : v \in N_G(u)\}|$.

Definition 4.2 (Colorful s-core). Given a graph $G = (V, E)$, a coloring χ of G and an integer s, the *colorful s-core* of G with respect to χ is the maximal subgraph g of G in which every vertex u in g has colorful degree $d_g^\chi(u) \geq s$.

Definition 4.3 (Colorful degeneracy ordering). Given a graph $G = (V, E)$ and a coloring χ of G, an ordering $(v_1, v_2, \ldots, v_n)$ of its vertices V is a *colorful degeneracy ordering* with respect to χ if, for each $1 \leq i \leq n$, v_i is a vertex with the smallest colorful degree with respect to χ in the subgraph of G induced by $\{v_i, v_{i+1}, \ldots, v_n\}$.

Colorful s-cores and a colorful degeneracy ordering can be computed by iteratively removing a vertex with the minimum colorful degree. We use the algorithms proposed in [60, 91] to compute every non-empty colorful s-core and a colorful degeneracy ordering, which run in $O(m)$ time and require $O(n \cdot |\chi|)$ space.

4.2.1 Computing a Large Initial Solution. We now introduce our technique to compute a large initial solution for maximum k-defective clique search.

We compute a large k-defective clique by iteratively removing a vertex with the smallest colorful degree. As the removal progresses, the remaining graph becomes denser and will eventually become a k-defective clique.

Initial solution. Given a graph G, a coloring χ of G, and a colorful degeneracy ordering $(v_1, v_2, \ldots, v_n)$ with respect to χ, let $S^* = \{v_i, v_{i+1}, \ldots, v_n\}$ be the longest suffix of $(v_1, v_2, \ldots, v_n)$ such that S^* is a k-defective clique in G. We use S^* as an initial solution.

Since a colorful degeneracy ordering can be obtained in $O(m)$ time, an initial solution can be computed in $O(m)$ time.

Comparison to techniques of existing algorithms. Several techniques have been proposed for computing an initial solution [13, 25, 45]. However, these techniques are complex and have non-linear time complexities. Specifically, the techniques of MDC [25], kDC [13], and KD-Club [45] require $O(n \cdot (\omega_k \cdot \delta + \delta^2))$, $O(m \cdot \delta)$, and $O(m \cdot \Delta)$ time, respectively, where ω_k denotes the size of a maximum k-defective clique. In contrast, our technique runs in $O(m)$ time, which is asymptotically faster than existing algorithms.

4.2.2 Graph Reduction Technique. We now introduce our graph reduction technique to remove vertices and edges that cannot be contained in any maximum k-defective clique.

After an initial solution S^* is computed, the threshold parameter q is set to $|S^*| + 1$ to find a larger k-defective clique. The following lemma formally introduces our reduction technique (a proof is provided in the full version [43]).

Lemma 4.1. *Given a graph $G = (V, E)$, a coloring χ of G, and positive integers k and q, every k-defective clique of size at least q is contained within the colorful $(q-k-1)$-core of G with respect to χ.*

According to Lemma 4.1, we reduce the input graph to its colorful $(q-k-1)$-core.

After computing the colorful $(q-k-1)$-core, we additionally apply the well-known truss-based reduction technique [13, 35], which iteratively removes an edge whose endpoints share at most $q-k-2$ common neighbors.

Let g be the colorful $(q-k-1)$-core of G, $m_g \leq m$ be the number of edges in g, and $\delta_g \leq \delta$ be the degeneracy of g. The time complexity of our reduction technique is $O(m + m_g \cdot \delta_g)$ because the colorful $(q-k-1)$ core can be computed in $O(m)$ time and the truss-based reduction takes $O(m_g \cdot \delta_g)$ time.

Comparison to reduction techniques of existing algorithms. The core-based reduction technique of [25, 26] reduces the input graph to its $(q-k-1)$-core. However, we emphasize that the colorful $(q-k-1)$-core is always a subgraph of the $(q-k-1)$-core. Thus, our reduction technique computes a smaller reduced graph than the core-based reduction technique. KD-Club [45] also proposes a reduction technique, but it has a high time complexity of $O(m \cdot \Delta \cdot |\chi|^2)$.

Applying the reduction technique for maximal k-defective clique enumeration. The graph reduction technique can also be applied for maximal k-defective clique enumeration. Before the branch-and-bound search, we apply our graph reduction technique using the threshold parameter q to compute the reduced graph. Then, we conduct the branch-and-bound search (Algorithm 2) with the reduced graph. In this way, we can further reduce the search space of our algorithm.

5 Performance Evaluation

In this section, we evaluate the performance of our algorithms as well as competing algorithms for both maximal k-defective clique enumeration and maximum k-defective clique search. We also evaluate the efficiency of our individual techniques. For brevity, we abbreviate 10^3, 10^6, and 10^9 as 1K, 1M, and 1B, respectively.

Table 3. Statistics of benchmark graph datasets, where ω represents the size of the maximum clique in the graph. D1 – D10 (top) are real-world graph datasets with more than 1M edges, and D11 – D14 (bottom) are large-scale graph datasets with more than 100M edges.

ID	Dataset	n	m	Δ	δ	ω
D1	soc-FourSquare	639K	3.21M	106K	63	30
D2	soc-buzznet	101K	2.76M	64.3K	153	31
D3	soc-digg	771K	5.91M	17.6K	236	50
D4	soc-BlogCatalog	88.8K	2.09M	9,444	221	45
D5	soc-LiveMocha	104K	2.19M	2,980	92	15
D6	socfb-Indiana	29.7K	1.31M	1,358	76	48
D7	socfb-UF	35.1K	1.47M	8,246	83	55
D8	socfb-UIllinois	30.8K	1.26M	4,632	85	57
D9	tech-as-skitter	1.69M	11.1M	35.5K	111	67
D10	web-wikipedia2009	1.86M	4.51M	2,624	66	31
D11	soc-orkut	3M	106M	27.5K	230	47
D12	enwiki-2023	6.62M	150M	237K	188	47
D13	it-2004	41.3M	1.03B	1.33M	3,224	3,222
D14	sk-2005	50.6M	1.81B	8.56M	4,510	4,511

5.1 Experimental Setup

5.1.1 *Algorithms.*

Maximal k-defective clique enumeration. We evaluate the efficiency of our algorithm by comparing it against the following two algorithms that solve maximal k-defective clique enumeration.

- WODC$_E$: our algorithm (Algorithm 2, incorporating the graph reduction technique of Section 4.2.2).
- Pivot2: algorithm proposed in [26].
- Pivot2+: state-of-the-art algorithm proposed in [26], an optimized version of Pivot2 enhanced with its pruning technique.

We emphasize that [26] is the first paper presenting algorithms solving maximal k-defective clique enumeration, and the two algorithms are by far significantly and consistently faster than the other algorithms presented in [26]. In our experiments, all algorithms report only the number of solutions. If the codes were modified to explicitly output all solutions, the total processing time would increase by the time required for outputting.

Maximum k-defective clique search. We evaluate the performance of our algorithm by comparing it against the following four state-of-the-art algorithms, which solve the maximum k-defective clique search problem and have significantly outperformed other existing algorithms [21, 35].

- WODC$_S$: our algorithm (Algorithm 3).
- MDC: algorithm proposed in [25].
- kDC2: algorithm proposed in [14], in which its reduction technique **RR3** is applied.
- kDC: algorithm proposed in [13].
- KD-Club: algorithm proposed in [45].

All the source codes were obtained from the authors of previous studies. All algorithms, including ours, are implemented in C++ and compiled with the `-O3` optimization flag. The experiments are conducted on a machine running CentOS, equipped with two Intel Xeon E5-2680 v3 2.50GHz CPUs and 256GB of RAM. The source code of this work is available at https://github.com/SNUCSE-CTA/WODC.

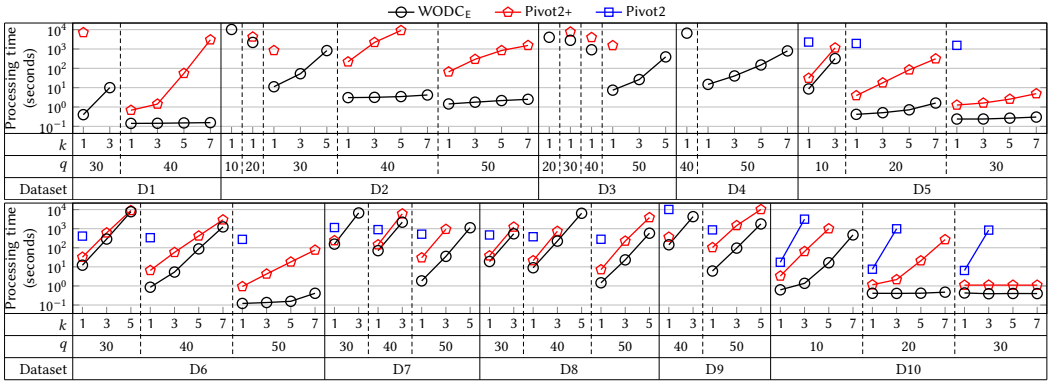


Fig. 5. Processing time (in seconds) of different algorithms for solving maximal k-defective clique enumeration on ten real-world graph datasets. Markers for algorithms that failed to solve the problem within the 3-hour time limit are not shown.

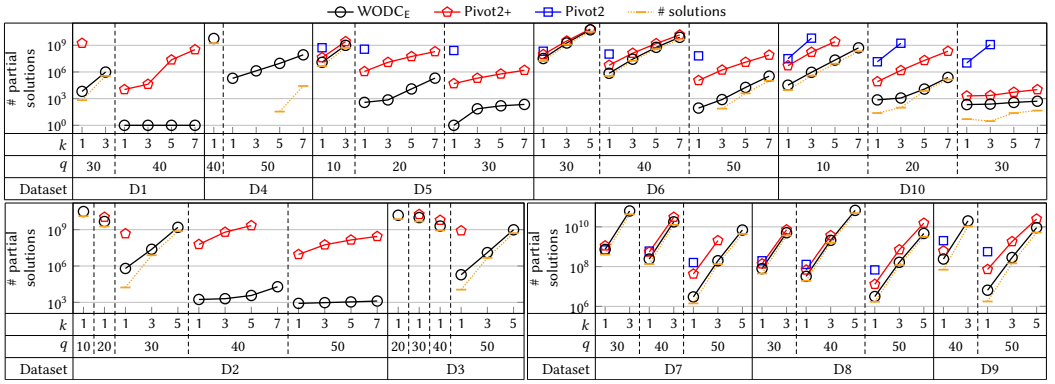


Fig. 6. Number of search tree nodes (# partial solutions) of different algorithms and number of solutions (# solutions) on real-world graph datasets. Markers for # solutions are not shown if there are no solutions.

5.1.2 Datasets. To evaluate the performance of different algorithms, we use 14 benchmark graph datasets. Table 3 shows statistics for the benchmark graph datasets D1 – D14. We obtained ten real-world graph datasets[1] D1 – D10 with more than 1M edges, which are widely used for evaluating the performance of maximal/maximum k-defective clique enumeration/search algorithms [13, 14, 21, 25, 26, 35, 45]. Additionally, we obtained four large-scale graph datasets[2] D11 – D14 with more than 100M edges to test both the efficiency and scalability of the algorithms.

5.1.3 Metrics. We record the total *processing time* required for running an algorithm on a graph dataset for given value(s) of k (and q). The processing time recorded reflects the total CPU time, excluding the I/O time spent loading the input graph from disk into main memory. Given that these problems are NP-hard, an algorithm may not terminate within a reasonable time; thus, we set a time limit of 3 hours for the real-world graphs and 24 hours for the large-scale graphs.

[1]https://lcs.ios.ac.cn/~caisw/Resource/realworld%20graphs.tar.gz
[2]https://law.di.unimi.it/datasets.php

Furthermore, we count the number of search tree nodes (i.e., the number of partial solutions) of the algorithms for maximal k-defective clique enumeration to verify that the result of our theoretical analysis matches the practical results.

5.1.4 Parameters. For maximal k-defective clique enumeration, there are two parameters: k and q. We select the parameters k and q from $\{1, 3, 5, 7\}$ and $\{10, 20, 30, 40, 50\}$, respectively. For large-scale graph datasets with more than 100M edges, we set q to a value slightly smaller than the size of the maximum clique [27], because no algorithm can terminate within the time limit for smaller q values. For maximum k-defective clique search, we choose the parameter k from $\{1, 5, 10, 15, 20\}$, as in [13, 14, 21, 25, 35, 45].

5.2 Maximal k-Defective Clique Enumeration

In this subsection, we evaluate the effectiveness of our algorithm WODC_E against the existing state-of-the-art algorithms.

Efficiency of different algorithms. Figure 5 and Figure 6 show the processing time and the number of search tree nodes of different algorithms on ten real-world graph datasets, respectively. Instances which are not displayed in the figures were unsolved by all algorithms within the time limit, e.g., when $k \geq 3$ and $q \leq 40$ in D3 of Figure 5. We can observe from Figure 5 that WODC_E is significantly and consistently faster for all datasets and all parameters. Notably, when $k = 1$ and $q = 30$ on D1, our algorithm achieves $16,290\times$ speedup over Pivot2+, and Pivot2 fails to solve the problem within the time limit. This is because the search space of our algorithm is not only theoretically worst-case optimal but also significantly smaller in practice; on D1 when $k = 1$ and $q = 30$, our algorithm generates only 6.5K search tree nodes, compared to 1.71B nodes generated by Pivot2+ (as shown in Figure 6). We also observe that the search space of our algorithm remains remarkably small on other datasets—for instance, on D2 with $q = 40$ and 50, it is four to six orders of magnitude smaller than that of Pivot2+, and on D5 with $q = 20$ and 30, it is three to five orders of magnitude smaller. These reductions in search space result in up to several thousand times faster processing time on D2 and D5, as shown in Figure 5. Moreover, in many cases, only WODC_E solves the problem within the time limit (e.g., on D1 when $k = 3$ and $q = 30$, and on D4). In most cases, Pivot2 fails to solve the problem within the time limit. In general, the speedup of WODC_E over Pivot2+ increases as k grows (e.g., on D1, D2, and D5). This is because the search space size of Pivot2+ approaches $O(2^n)$ as k grows; in contrast, our algorithm does not exhibit such growth. Similarly, as the size constraint q grows (e.g., on D3 when $k = 1$), the speedup improves, since our framework finds large solutions more efficiently than Pivot2+.

Evaluation of the search space size. Figure 6 shows the number of solutions, as well as the number of partial solutions generated by different algorithms. The search space of our algorithm is consistently smaller than that of competing algorithms for all datasets and all parameters. This result matches the theoretical analysis, as the number of search tree nodes is asymptotically smaller than the competing algorithms (see Table 1). Moreover, in most cases, the number of search tree nodes in our algorithm closely matches the number of solutions. This is because the search space of WODC_E is worst-case optimal. In some cases, WODC_E generates only a single partial solution (e.g., on D1 when $q = 40$ and on D5 when $k = 1$ and $q = 30$) because our reduction technique reduces the input graph to an empty graph. We also observe that our clique-first approach effectively mitigates the combinatorial explosion of small partial solutions. For example, on D10 with $k = 7$ and $q = 20$, Pivot2+ generates 215M partial solutions, 96% of which are small (i.e., of size at most 5). In contrast, our algorithm generates only 242K partial solutions.

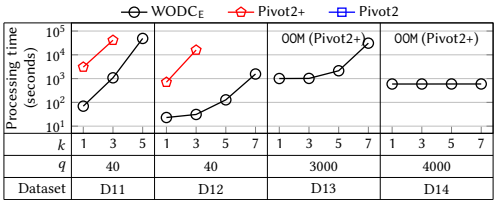


Fig. 7. Processing time (in seconds) of different algorithms for solving maximal k-defective clique enumeration on four large-scale graph datasets. Pivot2 failed to solve the problem within the 24-hour time limit for all large-scale datasets. OOM indicates "out of memory"; Pivot2+ runs out of memory when solving graphs with more than 1B edges (D13 and D14).

Scalability. We also evaluate the scalability of our algorithm by running the algorithms on large-scale graphs with more than 100M edges; see Figure 7. WODC_E always outperforms competing algorithms in terms of processing time across all datasets and all parameters. Pivot2 fails to solve the problem within the 24-hour time limit for all datasets. For two datasets, D13 and D14, with more than 1B edges, our algorithm is able to enumerate all solutions within the time limit. However, Pivot2+ runs out of memory; these results show that our algorithm is more scalable than the competitors.

Computational challenge of finding defective cliques. Maximal *clique* listing may be a problem which is NP-hard, but "easy" in most real-world graphs. We conducted an experiment to compare maximal clique listing against maximal defective clique listing. The well-known maximal clique listing algorithm by Eppstein et al. [33] successfully solved all instances (except D4) within 30 minutes on datasets D1–D10. In contrast, when $k = 5$ and $q = 10$ (and other cases) in Figure 5, none of the maximal k-defective clique enumeration algorithms were able to solve instances on D1–D9 within the 3-hour time limit. This shows the greater computational challenge posed by the defective clique problem.

5.3 Maximum k-Defective Clique Search

In this subsection, we evaluate the effectiveness of our algorithm WODC_S against the existing state-of-the-art algorithms.

Efficiency of different algorithms. Figure 8 shows the processing time on ten real-world graph datasets with varying k. Instances where all algorithms exceed the time limit are not shown, e.g., when $k \geq 15$ in D1. WODC_S is significantly faster for most of the datasets and varying k. Notably, when $k = 5$ on D1, our algorithm achieves a speedup of $18,339\times$ and $13,869\times$ over MDC and kDC2, respectively (other competing algorithms do not terminate within the time limit). This is because our clique-first approach quickly finds larger k-defective cliques (and updates q accordingly) by avoiding the combinatorial explosion of partial solutions; on D1 when $k = 5$, the number of partial solutions generated by MDC and kDC2 are 48.6M and 30K, respectively. In contrast, our algorithm generates only 6.51K partial solutions in total. Additionally, when $k = 1$ or 5 on D2 and D5, the search space of our algorithm is one to three orders of magnitude smaller than those of MDC, kDC2, and kDC. As a result, our algorithm achieves a speedup of one to four orders of magnitude compared to competing algorithms on D2 and D5. Though the search space of KD-Club is smaller than MDC or kDC2, KD-Club is always slower than WODC_S due to its expensive reduction technique. Also, WODC_S computes a large initial solution, which sets the threshold parameter q to a high value before the branch-and-bound search begins. For example, on D6–D8 when $k \leq 10$, our algorithm finds a maximum k-defective clique as the initial solution, thereby setting q to its maximum possible

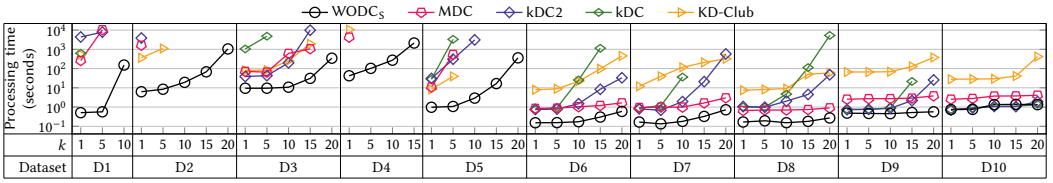


Fig. 8. Processing time (in seconds) of different algorithms for solving maximum k-defective clique search on ten real-world graphs. Markers for algorithms that failed to solve the problem within the 3-hour time limit are not shown.

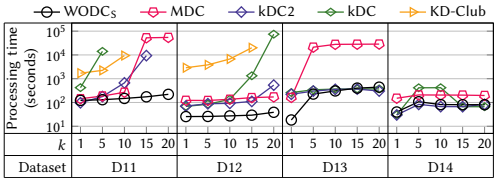


Fig. 9. Processing time (in seconds) of different algorithms for solving maximum k-defective clique search on four large-scale graphs. Markers for algorithms that failed to solve the problem within the 24-hour time limit are not shown.

value. As a result, our algorithm completes within a second on D6–D8 for all values of k. In general, the speedup of WODC$_S$ over the competing algorithms increases as k grows. This is because the search space size of the competing algorithms approaches $O(2^n)$ as k grows. Furthermore, as k increases, there are many cases where our algorithm solves the problem within the time limit, while all competing algorithms fail (e.g., on D1–D5). This demonstrates the efficiency of our algorithm for large values of k compared to the competing algorithms.

Scalability. We also evaluate the scalability of our algorithm with large-scale graph datasets with more than 100M edges; see Figure 9. In most cases, our algorithm is faster than all other competing algorithms. Notably, on D11 when $k = 20$, our algorithm solves the problem within four minutes, whereas all other competing algorithms, except MDC, fail to solve it within the 24-hour time limit (MDC takes more than 15 hours). On D13 and D14, KD-Club fails to solve the problem within the 24-hour time limit for all k-values. Across all datasets and all k values, our algorithm completes within ten minutes; these results indicate that WODC$_S$ is scalable for large-scale graphs.

5.4 Effectiveness of Individual Techniques

In this subsection, we evaluate the effectiveness of our individual techniques for maximal k-defective clique enumeration. The effectiveness of techniques for maximum k-defective clique search is provided in the full version [43].

We implemented the following variants of WODC$_E$ to evaluate the effectiveness of individual techniques.

- WODC$_E$-pivot: WODC$_E$ without the pivoting technique in Section 3.2.
- WODC$_E$-selection: WODC$_E$ in which the pivot vertex is randomly selected (i.e., without our pivot selection strategy).
- WODC$_E$-reduction: WODC$_E$ without the reduction technique in Section 4.2.2.

Figure 10 demonstrates the effectiveness of our individual techniques on datasets that show representative cases.

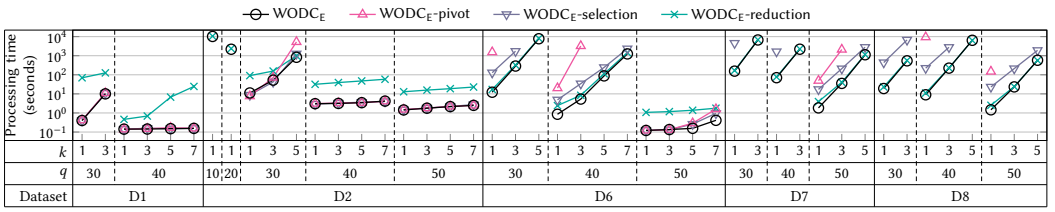
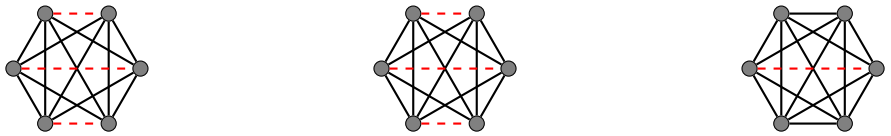


Fig. 10. Processing time (in seconds) of WODC_E and its variants. Markers for algorithms that failed to solve the problem within the 3-hour time limit are not shown.



(a) 2-plex (b) 0.8-quasi-clique (c) 1-defective clique

Fig. 11. Relaxed cliques with six vertices (missing edges are represented by red dashed edges).

- Our pivoting technique plays a crucial role in reducing the search space, especially for small q values and dense input graphs (e.g., D6–D8). In many cases, WODC_E solves the problem within the time limit, whereas WODC_E-pivot fails to do so (on D6–D8). This result highlights the effectiveness of the pivoting technique in eliminating unnecessary branches and reducing the search space.
- Our pivot selection strategy significantly reduces the search space. Specifically, WODC_E runs one to two orders of magnitude faster than WODC_E-selection on D6–D8 when $k = 1$ and $q = 30$. This demonstrates that our pivot selection strategy not only leads to a worst-case optimal search space theoretically but also effectively reduces the search space in practice.
- A comparison between WODC_E and WODC_E-reduction demonstrates that our reduction technique significantly reduces processing time, particularly for large q values and sparse input graphs (e.g., on D1 and D2). This is because as q increases, more vertices and edges are pruned by our reduction technique. On D1 when $q = 40$, our reduction technique reduces the input graph to an empty graph. These results confirm that our reduction technique substantially reduces the search space.

5.5 Case Study

In this subsection, we present a use case study to demonstrate the real-world applicability of our approach.

5.5.1 Comparison of Relaxed Cliques. We first give a conceptual comparison of relaxed cliques based on their definitions. The k-plex is defined by degrees of vertices [18, 23, 27], and the quasi-clique is defined by the density of a graph [38, 46, 89]. However, the defective clique specifies directly the number of missing edges. More specifically, the k-plex and the quasi-clique are defined as follows:

- k-plex: Each vertex is allowed to have at most $k - 1$ missing edges.
- γ-quasi-clique: Each vertex is connected to at least a γ fraction of the other vertices.

Note that both the 1-plex and the 1-quasi-clique are equivalent to a clique. Figure 11 illustrates examples of relaxed cliques with six vertices. A 2-plex of a clique with n vertices allows at most $\lfloor \frac{n}{2} \rfloor$ missing edges, as shown in Figure 11a. Similarly, a γ-quasi-clique with n vertices allows up to

Table 4. Link prediction on PPI network, where # cliq., # mis., and Acc. denote the number of relaxed cliques, the number of missing edges, and the accuracy, respectively.

k	$(1-0.1k)$-quasi-clique			$(k+1)$-plex			k-defective clique		
	# cliq.	# mis.	Acc.	# cliq.	# mis.	Acc.	# cliq.	# mis.	Acc.
1	143	64	82.8%	143	64	82.8%	56	40	**92.5%**
2	476	113	**88.5%**	508	113	**88.5%**	324	69	82.6%
3	1020	279	74.2%	1405	279	74.2%	804	84	**85.7%**
4	3521	671	44.9%	4831	671	44.9%	1482	108	**85.2%**

$\lfloor (1-\gamma) \cdot \binom{n}{2} \rfloor$ missing edges, e.g., a 0.8-quasi-clique with six vertices (shown in Figure 11b) allows up to 3 missing edges. That is, there is no way of specifying the case of just one missing edge (which is 1-defective clique) in the k-plex and γ-quasi-clique models. Therefore, the defective clique is arguably the most intuitive, and it is a better metric in specifying the relaxedness of a clique. In the following, we will show that the defective clique is also more effective than the k-plex and the quasi-clique in a downstream task.

5.5.2 Use Case Study. To show that the defective clique model achieves higher accuracy than other models in a downstream task (i.e., link prediction), we performed link prediction on PPI network using relaxed clique models. To enumerate maximal relaxed cliques, we employed state-of-the-art enumeration algorithms [27, 89].

The k-defective clique has the following relationships with other relaxed cliques:

- A k-defective clique is also a $(k+1)$-plex.
- A k-defective clique of size q is also a $(1-\frac{k}{q-1})$-quasi-clique.

For example, a 1-defective clique of size 6 in Figure 11c is also a 2-plex and a $1-\frac{1}{6-1}=0.8$ quasi-clique. In our comparison, therefore, we evaluated k-defective cliques, $(k+1)$-plexes, and $(1-\frac{k}{q-1})$-quasi-cliques for various values of k.

We conducted an experiment to predict missing interactions in a protein–protein interaction (PPI) network [47], where each vertex represents a protein and each edge indicates an interaction. A protein complex is a group of proteins that are physically co-located within a cell and collectively perform a specific biological function. Protein complexes typically form cliques, i.e., every pair of proteins in the complex interacts with each other [4, 88]. Prior studies have attempted to identify missing interactions in noisy PPI networks by detecting relaxed cliques and completing the missing edges within them [88]. In our experiment, we evaluated the three relaxed clique models using the procedure below:

(i) Enumerate all maximal relaxed cliques of size greater than 10 (i.e., $q=11$).
(ii) Identify all missing edges in detected relaxed cliques.
(iii) Evaluate the accuracy, defined as the ratio of *positive* missing edges, where a missing edge is considered positive [88] if the two proteins involved are members of the same protein complex according to a ground-truth dataset [62].

Table 4 shows the results. In most cases, the defective clique model achieves the highest prediction accuracy. Notably, 1-defective cliques attain the best performance, with an accuracy of 92.5%. Moreover, as k increases, the accuracy of k-defective cliques remains above 80%, while the performance of other models drops significantly. This indicates that the defective clique provides a more accurate link prediction on PPI network compared to the other models.

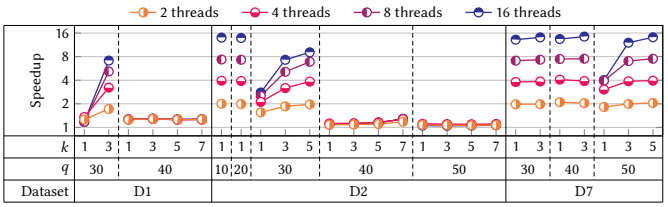


Fig. 12. Speedup achieved by the parallel implementation of WODC_E compared to the sequential baseline.

5.6 Parallelization

We implemented a parallel version of our enumeration algorithm WODC_E. WODC_E consists of two phases: graph reduction (Section 4.2.2) and branch-and-bound (Algorithm 2 in Section 3.4). We have parallelized Algorithm 2. Recall that Algorithm 2 partitions the original task into n subtasks, where each subtask enumerates all solutions whose smallest vertex is v_i for each $v_i \in V$ (Lines 3–6). Since these subtasks are independent, we dynamically assign them to multiple threads, allowing Lines 2–6 to be executed in parallel.

Figure 12 shows the speedup achieved by the parallel algorithm using 2, 4, 8, and 16 threads, compared to the sequential algorithm (i.e., WODC_E). In *hard* cases, i.e., when the branch-and-bound phase is time-consuming (taking more than 60 seconds in Figure 5), our algorithm achieves nearly linear speedup with respect to the number of threads, e.g., on D2 when $q \leq 20$ and on D7 when $q \leq 40$. This is because, in hard cases, the problem is typically divided into a large number of subtasks, ranging from thousands to millions, which enables effective load balancing over multiple threads. Cases where our algorithm did not exhibit linear speedup (e.g., on D1 and D2 when $q \geq 40$) correspond to *easy* instances, where graph reduction accounted for the majority of the total processing time.

6 Conclusion

In this paper, we proposed a novel branch-and-bound algorithm for enumerating maximal k-defective cliques with a worst-case optimal search space. Based on this algorithm, we also developed an efficient framework for maximum k-defective clique search. We show the effectiveness of our algorithms both theoretically and empirically.

It would be an interesting future work to apply our techniques to other problems related to relaxed cliques. For example, our clique-first approach is a high-level idea that may be applied to other relaxed cliques. Any relaxed clique consists of a large number of non-missing edges and a relatively smaller number of missing edges. Our clique-first approach first generates a clique (i.e., non-missing edges) and then adds missing edges, which reduces the search space as shown in Figure 2. This approach may be applied to other relaxed cliques.

Acknowledgments

J. Jang, Y. Nam, and K. Park were supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00551, Framework of Practical Algorithms for NP-hard Graph Problems). H. Kim was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2020-II201373, Artificial Intelligence Graduate School Program (Hanyang University)).

References

[1] James Abello, Mauricio GC Resende, and Sandra Sudarsky. 2002. Massive quasi-clique detection. In *Proceedings of LATIN*. 598–612.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. 2016. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems* 55 (2016), 278–288.

[3] Mohammad Almasri, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-mei Hwu. 2022. Parallel k-clique counting on gpus. In *Proceedings of ICS*. 1–14.

[4] Gary D Bader and Christopher WV Hogue. 2002. Analyzing yeast protein–protein interaction data obtained from different sources. *Nature biotechnology* 20, 10 (2002), 991–997.

[5] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V Hicks. 2011. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research* 59, 1 (2011), 133–142.

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

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

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

[9] Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. 2002. An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research* 138, 1 (2002), 21–28.

[10] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

[11] Lijun Chang. 2019. Efficient maximum clique computation over large sparse graphs. In *Proceedings of SIGKDD*. 529–538.

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

[13] Lijun Chang. 2023. Efficient Maximum k-Defective Clique Computation with Improved Time Complexity. *Proceedings of the ACM on Management of Data* 1, 3, Article 209 (2023), 26 pages.

[14] Lijun Chang. 2025. Maximum Defective Clique Computation: Improved Time Complexities and Practical Performance. *Proceedings of the VLDB Endowment* 18, 2 (2025), 200–212.

[15] Lijun Chang, Rashmika Gamage, and Jeffrey Xu Yu. 2024. Efficient k-Clique count estimation with accuracy guarantee. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3707–3719.

[16] Lijun Chang and Lu Qin. 2019. *Cohesive Subgraph Computation over Large Sparse Graphs*. Springer.

[17] Lijun Chang, Mouyi Xu, and Darren Strash. 2022. Efficient maximum k-plex computation over large sparse graphs. *Proceedings of the VLDB Endowment* 16, 2 (2022), 127–139.

[18] Lijun Chang and Kai Yao. 2024. Maximum k-plex computation: Theory and practice. *Proceedings of the ACM on Management of Data* 2, 1, Article 63 (2024), 26 pages.

[19] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. 2013. Fast maximal cliques enumeration in sparse graphs. *Algorithmica* 66 (2013), 173–186.

[20] Joanna Che, Kasra Jamshidi, and Keval Vora. 2024. Contigra: graph mining with containment constraints. In *Proceedings of EuroSys*. 50–65.

[21] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. 2021. Computing maximum k-defective cliques in massive graphs. *Computers & Operations Research* 127 (2021), 105131.

[22] James Cheng, Linhong Zhu, Yiping Ke, and Shumo Chu. 2012. Fast algorithms for maximal clique enumeration with limited memory. In *Proceedings of SIGKDD*. 1240–1248.

[23] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. 2018. D2K: scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of SIGKDD*. 1272–1281.

[24] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. 2017. Fast enumeration of large k-plexes. In *Proceedings of SIGKDD*. 115–124.

[25] Qiangqiang Dai, Ronghua Li, Donghang Cui, and Guoren Wang. 2024. Theoretically and Practically Efficient Maximum Defective Clique Search. *Proceedings of the ACM on Management of Data* 2, 4, Article 206 (2024), 27 pages.

[26] Qiangqiang Dai, Rong-Hua Li, Meihao Liao, and Guoren Wang. 2023. Maximal Defective Clique Enumeration. *Proceedings of the ACM on Management of Data* 1, 1, Article 77 (2023), 26 pages.

[27] Qiangqiang Dai, Rong-Hua Li, Hongchao Qin, Meihao Liao, and Guoren Wang. 2022. Scaling up maximal k-plex enumeration. In *Proceedings of CIKM*. 345–354.

[28] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-cliques in sparse real-world graphs. In *Proceedings of WWW*. 589–598.

[29] Apurba Das, Seyed-Vahid Sanei-Mehri, and Srikanta Tirthapura. 2018. Shared-memory parallel maximal clique enumeration. In *Proceedings of HiPC*. 62–71.

[30] Wen Deng, Weiguo Zheng, and Hong Cheng. 2024. Accelerating Maximal Clique Enumeration via Graph Reduction. *Proceedings of the VLDB Endowment* 17, 10 (2024), 2419–2431.

[31] 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.

[32] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing all maximal cliques in sparse graphs in near-optimal time. In *Proceedings of ISAAC*. 403–414.

[33] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics (JEA)* 18, Article 3.1 (2013), 21 pages.

[34] F.V. Fomin and D. Kratsch. 2010. *Exact Exponential Algorithms*. Springer.

[35] 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 *Proceedings of AAAI*. 10174–10183.

[36] Shuohao Gao, Kaiqiang Yu, Shengxin Liu, and Cheng Long. 2025. Maximum k-Plex Search: An Alternated Reduction-and-Bound Method. *Proceedings of the VLDB Endowment* 18, 2 (2025), 363–376.

[37] Timo Gschwind, Stefan Irnich, Fabio Furini, and Roberto Wolfler Calvo. 2021. A branch-and-price framework for decomposing graphs into relaxed cliques. *INFORMS Journal on Computing* 33, 3 (2021), 1070–1090.

[38] Guimu Guo, Da Yan, M Tamer Özsu, Zhe Jiang, and Jalal Khalil. 2020. Scalable mining of maximal quasi-cliques: an algorithm-system codesign approach. *Proceedings of the VLDB Endowment* 14, 4 (2020), 573–585.

[39] Eric Harley, Anthony Bonner, and Nathan Goodman. 2001. Uniform integration of genome mapping data using intersection graphs. *Bioinformatics* 17, 6 (2001), 487–494.

[40] Shweta Jain and C Seshadhri. 2017. A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of WWW*. 441–449.

[41] Shweta Jain and C Seshadhri. 2020. Provably and efficiently approximating near-cliques using the Turán shadow: PEANUTS. In *Proceedings of WWW*. 1966–1976.

[42] Kasra Jamshidi, Rakesh Mahadasa, and Keval Vora. 2020. Peregrine: a pattern-aware graph mining system. In *Proceedings of EuroSys*. Article 13, 16 pages.

[43] Jihoon Jang, Yehyun Nam, Kunsoo Park, and Hyunjoon Kim. 2025. Efficient Defective Clique Enumeration and Search with Worst-Case Optimal Search Space (Full Version). https://github.com/SNUCSE-CTA/WODC/blob/main/WODC-full.pdf.

[44] Tang Jian. 1986. An $O(2^{0.304n})$ algorithm for solving maximum independent set problem. *IEEE Trans. Comput.* 100, 9 (1986), 847–851.

[45] Mingming Jin, Jiongzhi Zheng, and Kun He. 2024. KD-Club: An Efficient Exact Algorithm with New Coloring-based Upper Bound for the Maximum k-Defective Clique Problem. In *Proceedings of AAAI*. 20735–20742.

[46] Jalal Khalil, Da Yan, Guimu Guo, and Lyuheng Yuan. 2022. Parallel mining of large maximal quasi-cliques. *The VLDB Journal* 31, 4 (2022), 649–674.

[47] Nevan J Krogan, Gerard Cagney, Haiyuan Yu, Gouqing Zhong, Xinghua Guo, Alexandr Ignatchenko, Joyce Li, Shuye Pu, Nira Datta, Aaron P Tikuisis, et al. 2006. Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. *Nature* 440, 7084 (2006), 637–643.

[48] Andrea Lancichinetti and Santo Fortunato. 2009. Community detection algorithms: a comparative analysis. *Physical Review E–Statistical, Nonlinear, and Soft Matter Physics* 80, 5, Article 056117 (2009), 11 pages.

[49] Ailsa H Land and Alison G Doig. 2010. *An automatic method for solving discrete programming problems*. Springer.

[50] Victor E Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. 2010. A survey of algorithms for dense subgraph discovery. *Managing and mining graph data* (2010), 303–336.

[51] Brenton Lessley, Talita Perciano, Manish Mathai, Hank Childs, and E Wes Bethel. 2017. Maximal clique enumeration with data-parallel primitives. In *Proceedings of LDAV*. 16–25.

[52] 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. *Computers & Operations Research* 84 (2017), 1–15.

[53] Ronghua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering Heuristics for k-clique Listing. *Proceedings of the VLDB Endowment* 13, 11 (2020), 2536–2548.

[54] Yang Liu, Hejiao Huang, Kaiqiang Yu, Shengxin Liu, and Cheng Long. 2025. Efficient Maximum s-Bundle Search via Local Vertex Connectivity. *Proceedings of the ACM on Management of Data* 3, 1, Article 37 (2025), 27 pages.

[55] Can Lu, Jeffrey Xu Yu, Hao Wei, and Yikai Zhang. 2017. Finding the maximum clique in massive graphs. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1538–1549.

[56] David W Matula and Leland L Beck. 1983. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM* 30, 3 (1983), 417–427.

[57] John W Moon and Leo Moser. 1965. On cliques in graphs. *Israel journal of Mathematics* 3 (1965), 23–28.

[58] Yehyun Nam, Jihoon Jang, Kunsoo Park, Jianye Yang, and Cheng Long. 2025. DIST: Efficient k-Clique Listing via Induced Subgraph Trie. *arXiv preprint arXiv:2502.00317* (2025).

[59] Kevin A Naudé. 2016. Refined pivot selection for maximal clique enumeration in graphs. *Theoretical Computer Science* 613 (2016), 28–37.
[60] Minjia Pan, Rong-Hua Li, Qi Zhang, Yongheng Dai, Qun Tian, and Guoren Wang. 2022. Fairness-aware maximal clique enumeration. In *Proceedings of ICDE*. 259–271.
[61] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2013. On clique relaxation models in network analysis. *European Journal of Operational Research* 226, 1 (2013), 9–18.
[62] Shuye Pu, Jessica Wong, Brian Turner, Emerson Cho, and Shoshana J Wodak. 2009. Up-to-date catalogues of yeast protein complexes. *Nucleic acids research* 37, 3 (2009), 825–831.
[63] John Michael Robson. 1986. Algorithms for maximum independent sets. *Journal of Algorithms* 7, 3 (1986), 425–440.
[64] John M Robson. 2001. Finding a maximum independent set in time $O(2^{n/4})$. https://www.labri.fr/perso/robson/mis/techrep.html
[65] Ryan A Rossi, David F Gleich, and Assefaw H Gebremedhin. 2015. Parallel maximum clique algorithms with applications to network analysis. *SIAM Journal on Scientific Computing* 37, 5 (2015), C589–C616.
[66] Pablo San Segundo, Jorge Artieda, and Darren Strash. 2018. Efficiently enumerating all maximal cliques with bit-parallelism. *Computers & Operations Research* 92 (2018), 37–46.
[67] Pablo San Segundo, Alvaro Lopez, and Panos M Pardalos. 2016. A new exact maximum clique algorithm for large and massive sparse graphs. *Computers & Operations Research* 66 (2016), 81–94.
[68] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.
[69] Hanif D Sherali and J Cole Smith. 2006. A polyhedral study of the generalized vertex packing problem. *Mathematical Programming* 107 (2006), 367–390.
[70] Hanif D Sherali, J Cole Smith, and Antonio A Trani. 2002. An airspace planning model for selecting flight-plans under workload, safety, and equity considerations. *Transportation Science* 36, 4 (2002), 378–397.
[71] Jessica Shi, Laxman Dhulipala, and Julian Shun. 2021. Parallel clique counting and peeling algorithms. In *Proceedings of ACDA*. 135–146.
[72] Vladimir Stozhkov, Austin Buchanan, Sergiy Butenko, and Vladimir Boginski. 2022. Continuous cubic formulations for cluster detection problems in networks. *Mathematical Programming* 196, 1 (2022), 279–307.
[73] Robert Endre Tarjan and Anthony E Trojanowski. 1977. Finding a maximum independent set. *SIAM J. Comput.* 6, 3 (1977), 537–546.
[74] Carlos HC Teixeira, Alexandre J Fonseca, Marco Serafini, Georgos Siganos, Mohammed J Zaki, and Ashraf Aboulnaga. 2015. Arabesque: a system for distributed graph mining. In *Proceedings of SOSP*. 425–440.
[75] Etsuji Tomita. 2017. Efficient algorithms for finding maximum and maximal cliques and their applications. In *Proceedings of WALCOM*. 3–15.
[76] Etsuji Tomita and Toshikatsu Kameda. 2007. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global optimization* 37 (2007), 95–111.
[77] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science* 363, 1 (2006), 28–42.
[78] Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. 2013. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications* 56 (2013), 113–130.
[79] Alexander Veremyev, Oleg A Prokopyev, Sergiy Butenko, and Eduardo L Pasiliao. 2016. Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications* 64, 1 (2016), 177–214.
[80] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2024. Efficient k-Clique Listing: An Edge-Oriented Branching Strategy. *Proceedings of the ACM on Management of Data* 2, 1, Article 7 (2024), 26 pages.
[81] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2025. Maximal Clique Enumeration with Hybrid Branching and Early Termination. In *Proceedings of ICDE*. 599–612.
[82] Zhengren Wang, Yi Zhou, Chunyu Luo, and Mingyu Xiao. 2023. A Fast Maximum k-Plex Algorithm Parameterized by the Degeneracy Gap. In *Proceedings of IJCAI*. 5648–5656.
[83] Zhengren Wang, Yi Zhou, Mingyu Xiao, and Bakhadyr Khoussainov. 2022. Listing maximal k-plexes in large real-world graphs. In *Proceedings of WWW*. 1517–1527.
[84] Jingen Xiang, Cong Guo, and Ashraf Aboulnaga. 2013. Scalable maximum clique computation using mapreduce. In *Proceedings of ICDE*. 74–85.
[85] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. 2017. A fast algorithm to compute maximum k-plexes in social network analysis. In *Proceedings of AAAI*. 919–925.
[86] Mihalis Yannakakis. 1978. Node-and edge-deletion NP-complete problems. In *Proceedings of STOC*. 253–264.
[87] Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongzhi Chen, and Guoren Wang. 2022. Lightning fast and space efficient k-clique counting. In *Proceedings of WWW*. 1191–1202.

[88] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. 2006. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics* 22, 7 (2006), 823–829.
[89] Kaiqiang Yu and Cheng Long. 2023. Fast maximal quasi-clique enumeration: A pruning and branching co-design approach. *Proceedings of the ACM on Management of Data* 1, 3, Article 211 (2023), 26 pages.
[90] Zhirong Yuan, You Peng, Peng Cheng, Li Han, Xuemin Lin, Lei Chen, and Wenjie Zhang. 2022. Efficient k-clique Listing with Set Intersection Speedup. In *Proceedings of ICDE*. 1955–1968.
[91] Qi Zhang, Rong-Hua Li, Minjia Pan, Yongheng Dai, Qun Tian, and Guoren Wang. 2023. Fairness-aware maximal clique in large graphs: concepts and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 35, 11 (2023), 11368–11387.
[92] 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 *Proceedings of AAAI*. 12453–12460.
[93] Yi Zhou, Jingwei Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. 2020. Enumerating maximal k-plexes with worst-case time guarantee. In *Proceedings of AAAI*. 2442–2449.

Received April 2025; revised July 2025; accepted August 2025