Maximal Defective Clique Enumeration

ABSTRACT

Maximal clique enumeration is a fundamental operator in graph analysis. The model of clique, however, is typically too restrictive for real-world applications as it requires an edge for every pair of vertices. To remedy this restriction, practical graph analysis applications often resort to find relaxed cliques as alternatives. In this work, we investigate a notable relaxed clique model, called s-defective clique, which allows at most s edges to be missing. Similar to the complexity of maximal clique enumeration, the problem of enumerating all maximal s-defective cliques is also NP-hard. To solve this problem, we first develop a new polynomial-delay algorithm based on a carefully-designed reverse search technique, which can output two consecutive results within polynomial time. To achieve better practical efficiency, we propose a branch-and-bound algorithm with a novel pivoting technique. We prove that the time complexity of this algorithm depends only on $O(\alpha_s^n)$ or $O(\alpha_s^\delta)$ when using a degeneracy ordering optimization, where α_s is a positive real number strictly less than 2, and δ ($\delta < n$) is the degeneracy of the graph. To our knowledge, this is the first algorithm that can break the $O(2^n)$ time complexity to enumerate all maximal s-defective cliques ($s > 0$). We also develop several new pruning techniques to further improve the efficiency of our branch-and-bound algorithm to enumerate all relatively-large maximal s-defective cliques. In addition, we also generalize our pivot-based branch-and-bound algorithm to enumerate all maximal subgraphs satisfying a hereditary property. Here we call a graph meeting the hereditary property if all its subgraphs have the same property as itself. Finally, extensive experiments on 25 datasets demonstrate the efficiency, effectiveness, and scalability of the proposed solutions.

1 INTRODUCTION

Mining cohesive subgraphs in real-world networks is a fundamental problem in graph analysis. There are numerous applications that can be modeled as a cohesive subgraph mining problem, including detecting communities in social networks [4, 17, 23], mining protein complexes in protein-protein interaction (PPI) networks [20, 46], and statistical analysis in financial networks [7, 8]. The classic maximal clique model which requires an edge for every pair of vertices is widely used to represent cohesive subgraphs, due to its tightly-connected interiors and the existence of many advanced approaches to find all maximal cliques [9, 16, 18, 35, 39, 43].

However, the condition that requires all possible relations to exist in the community may be too restrictive for real-world applications, since noises or faults may occur in real-world networks [15] and the interactions between individuals in a community can be accomplished through non-direct relationships [36]. To remedy this issue, many relaxed clique models have been developed as alternatives to represent cohesive subgraphs, such as s-defective clique [46], k-plex [6, 15, 40, 50], r-clique [5, 32], and γ-quasi-clique [31, 37]. In this paper, we focus mainly on the s-defective clique model, as it can well approximate the clique model [46].

Given a graph G, a subset S of vertices in G is a maximal s-defective clique if (1) the subgraph induced by S has at least $\binom{|S|}{2} - s$ edges, and (2) there does not exist any vertex subset satisfying (1) and containing S. For example, consider the graph G shown in Fig. 1(a). It is easy to check that Fig.1(b) is a maximal 1-defective clique and Fig. 1(c) is a maximal 2-defective clique. Clearly, the clique is the special case of the s-defective clique when $s = 0$. In addition, many other relaxed clique models are also closely related to the s-defective clique. For instance, an s-defective clique must be an $(s+1)$-plex, where a k-plex is a subgraph in which every vertex is adjacent to all but at most k vertices [40]. For any s-defective clique with size larger than $s + 1$, it is also a 2-clique [32] (a subgraph in which the distance between each pair of vertices is no larger than 2 in the original graph), since the diameter of such an s-defective clique is always no larger than 2 (see Property 2). Due to these nice properties, the s-defective clique can be used in many real-world applications, and some of them are listed as follows.

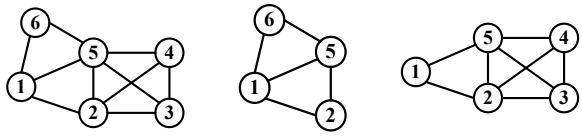
Implicit interactions prediction. Intuitively, given any two non-adjacent vertices, if their common neighbors in the group form a clique, then these two vertices are likely to be connected. This means that the missing edges in an s-defective clique have a strong prediction for the implicit interactions in the graph if s is small. Thus, in real-world networks, such as PPI networks, collaboration networks, and social networks, one can make use of s-defective cliques to predict implicit interactions. A notable example is that Yu et al. [46] have successfully applied the s-defective clique model to detect implicit protein interactions in PPI networks.

Community detection in social networks. In an s-defective clique, it only allows at most s edges to be missing, which ensures that each s-defective clique is densely-connected when s is small. Thus, in real-world networks, such as social networks, communication networks, and web graphs, the tightly-connected communities can be detected by enumerating s-defective cliques. Compared to the clique model, the communities identified by the s-defective clique model are more flexible, as they allow up to s missing links.

Statistical analysis in financial networks. In financial networks, such as stock market, the vertices can be represented as financial instruments and edges are the correlations between instruments. If the price of a financial instrument produces large fluctuations, the other instruments that are closely related to it may also fluctuate. By using the s-defective clique model, we can identify the group of financial instruments whose prices may collectively fluctuate. Note that the indirectly-linked (but closely-related) financial instruments can also be detected by the s-defective clique model, which cannot be identified by the traditional clique model.

SIGMOD '23, June 18–32, 2023, Seattle, WA, USA

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



(a) The original graph (b) 1-defective clique (c) 2-defective clique

Figure 1: A running example graph G.

Although the maximal s-defective clique model can be used for many applications, an efficient algorithm for enumerating all maximal s-defective cliques on large graphs is still lacking. To our knowledge, the only existing algorithm to identify maximal s-defective cliques was proposed in [46]. The main idea of this algorithm is that it first makes use of the Bron-Kerbosch (BK) algorithm [9] to enumerate all maximal cliques and then determines whether each possible combination of maximal cliques can form an s-defective clique. Clearly, such an algorithm is rather inefficient and requires very high memory overheads to store the maximal cliques. More importantly, this algorithm typically cannot obtain all maximal s-defective cliques of a graph. For instance, consider the graph G shown in Fig. 1(a), it is easy to see that $\{v_1, v_2, v_3, v_5\}$ is a maximal 1-defective clique. Such a maximal 1-defective clique, however, does not belong to any pair of maximal cliques, thus cannot be identified by the algorithm proposed in [46].

To overcome this problem, we develop two efficient and novel approaches to enumerate all maximal s-defective cliques. We show that both of our proposed algorithms have non-trivial worst-case time complexity guarantees. To our knowledge, our algorithms are the best algorithms so far in terms of the worst-case time complexity. Below, we briefly summarize the main contributions of our work.

A novel polynomial-delay algorithm. Our first solution for enumerating all maximal s-defective cliques is a polynomial-delay algorithm based on a reverse search technique [2], which can output any two consecutive results within polynomial time. Specifically, we first propose a novel graph-like structure to characterize the relationships of all maximal s-defective cliques in a given graph. Such a new structure provides us with guidelines on which possible results can be jumped from one to the other. Based on this, we then propose a reverse search procedure to recursively enumerate each result. We prove that the delay of any two consecutive results output by our algorithm is bounded in $O(n^2 \frac{\Delta^{2s+1}}{4^s})$ time, where n is the number of vertices of the graph, Δ is the size of the maximum s-defective clique, and s is a small constant. To our knowledge, this is the first theoretical result on enumerating maximal s-defective cliques within polynomial-delay time.

New branch-and-bound enumeration algorithms. Although our polynomial-delay algorithm has a nice theoretical property, its practical performance may be inefficient for large real-world graphs. To obtain a more practical solution, we propose a new branch-and-bound enumeration algorithm with a carefully-designed pivoting technique. We prove that the time complexity of our branch-and-bound algorithm depends mainly on $O(\alpha_s^n)$, where α_s is a positive real number and strictly less than 2. To our knowledge, this is the first algorithm that breaks the $O(2^n)$ barrier of the basic branch-and-bound algorithm for maximal s-defective clique enumeration. To boost the efficiency, we also develop an improved branch-and-bound algorithm with a degeneracy-ordering optimization technique. We show that the worst-case time complexity of such an improved algorithm depends mainly on $O(\alpha_s^\delta)$. Here δ denotes the degeneracy of the graph which is often very small for large real-world graphs [18, 28]. In addition, several non-trivial and efficient pruning techniques are also proposed to further improve the performance of our algorithms.

A general pivoting paradigm. Inspired by the proposed pivot-based branch-and-bound algorithms, we further develop a general pivoting paradigm for enumerating all maximal subgraphs that satisfy the hereditary property. Here we call a graph meeting the hereditary property if all its subgraphs have the same property as itself (e.g., both cliques and s-defective cliques satisfy the hereditary property). We believe that the proposed general pivoting paradigm could be of independent interests, which can provide useful guidelines to devise efficient pivot-based algorithms for other maximal hereditary subgraph enumeration problems.

Extensive experimental evaluations. We conduct extensive experiments to evaluate the efficiency, effectiveness, and scalability of the proposed algorithms on 25 datasets including DIMACS benchmark graphs and real-world graphs. The experimental results show that (1) the proposed polynomial-delay algorithm performs well in terms of time delay and also achieve a good practical performance when processing small dense graphs; (2) the proposed pivot-based algorithms consistently achieve the best performance, which can be several orders of magnitude faster than the basic branch-and-bound algorithm (without pivoting technique). For example, on the dblp-2010 dataset (226,413 vertices and 1,432,920 edges), our best pivot-based algorithm takes less than 0.2 seconds to enumerate all maximal 1-defective cliques with size no less than 10, while the basic branch-and-bound algorithm cannot terminate within 24 hours. In addition, we also apply our approaches to predict the implicit interactions in a PPI network and a collaboration network, the results show that our s-defective clique based approaches significantly outperform k-plex based solutions in terms of both running time and prediction accuracy.

2 PROBLEM STATEMENT

Let $G = (V, E)$ be the undirected and unweighted graph, where V and E are the set of vertices and edges respectively. Denote by $n = |V|$ and $m = |E|$ the number of vertices and edges of G, respectively. The complementary graph of G is denoted by $\overline{G} = (V, \overline{E})$, where each edge $(u, v) \in \overline{E}$ if and only if $(u, v) \notin E$. For a vertex v, the set of neighbors in G is defined as $N_v(G) = \{u \in V | (v, u) \in E\}$, and the degree of v in G is $d_v(G) = |N_v(G)|$. Similarity, we use $\overline{N}_v(G)$ and $\overline{d}_v(G)$ to denote the set of neighbors and the degree of v in $\overline{G}$ respectively. Given a vertex subset S of V, we let $G(S) = (S, E_S)$ be the subgraph of G induced by S, where $E_S = \{(u, v) \in E | u, v \in C\}$. If the context is clear, we abbreviate $N_v(G(S))$ to $N_v(S)$ and $d_v(G(S))$ to $d_v(S)$ respectively.

A vertex subset S of V is a clique if every pair of vertices is adjacent to each other (i.e., $(u, v) \in E$ for each $u, v \in S$). We refer to a clique S as a maximal clique if no vertex in $V \setminus S$ is adjacent to all vertices in S. The problem of finding all maximal cliques on a graph has been well studied [9, 16, 18, 35, 39, 43]. However,

as we discussed in Section 1, the concept of clique may be overly restrictive for many real-world applications [36], thus relaxed clique models are often used as alternatives in practice [5, 6, 15, 31, 46]. In this paper, we focus mainly on a notable relaxed clique model, called s-defective clique, which was first proposed in [46].

Definition 1 (Maximal s-defective clique). Given a graph G and an integer s, a maximal s-defective clique S is a subset of vertices of G if (1) the subgraph $G(S)$ induced by S contains at least $\binom{|S|}{2} - s$ edges, and (2) there does not exist any vertex subset satisfying (1) and containing S.

Clearly, an s-defective clique is a clique when $s = 0$, which indicates that the clique is a special case of the s-defective clique. The following example illustrates the concept of s-defective clique.

Example 1. Given an undirected graph shown in Fig. 1(a), the vertex set $\{v_1, v_2, v_5, v_6\}$ is a maximal 1-defective (see Fig. 1(b)), and it is not a clique as v_2 is not the neighbor of v_6. Similarity, the vertex set $\{v_1, v_2, v_3, v_4, v_5\}$ is a maximal 2-defective clique (see Fig. 1(c)), and it is not a 1-defective clique. Because only one edge is allowed to be missing in 1-defective clique, while two edges (v_1, v_3) and (v_1, v_4) are missed in this subgraph.

We then show that the s-defective clique has the following two useful properties, which will be used to devise our enumeration algorithms.

Property 1 (Hereditary). Given an arbitrary s-defective clique S of G, for each subset H of S, H is also an s-defective clique of G.

PROOF. Suppose on the contrary that there is a subset H' of S that is not a s-defective clique. Then, by definition, the number of missing edges in $G(H')$ is larger than s. However, when we add all vertices in $S \setminus H'$ back to $G(H')$, the number of missing edges in $G(H')$ keeps unchanged. This leads to S not being an s-defective clique, which a contradiction. □

By Property 1, it is easy to check that an s-defective clique S is maximal in G if there is no vertex in $V \setminus S$ that can be added to S.

Property 2. Given an s-defective clique S of G, the diameter of $G(S)$ is at most 2 if $|S| \geq s + 2$.

PROOF. Assume that the diameter r of $G(S)$ is no less than 3. There must exist two vertices u and v in S satisfying $N_v(S) \cap N_u(S) = \emptyset$. Then, we have $d_v(S) + d_u(S) + 2 \leq |S|$. By Definition 1, we also have $d_v(S) + d_u(S) \geq 2(|S| - 2) - (s - 1) = 2|S| - s - 3$ since the number of missing edges in $G(S)$ is at most s. When combining above two results, we can derive that $2|S| - s - 3 \leq |S| - 2$. As a result, we have $|S| \leq s + 1$ if $r \geq 3$, and this property is proved. □

By Property 2, each s-defective clique with size no less than $s + 2$ must be densely connected, since the diameter of the subgraph is no larger than 2. Thus, in real-world applications, we are often interested in enumerating all maximal s-defective cliques with size q no less than $s + 2$. To obtain a good relaxation of a clique, the parameter s in the s-defective clique model is generally very small (e.g., $s \leq 5$), so that the size constraint $q \geq s + 2$ is easy to meet for relatively-large maximal s-defective cliques. In this paper, we also investigate the problem of enumerating all relatively-large maximal s-defective cliques. Below, we formally define our problems.

Algorithm 1: The reverse search framework [2].

```
Input: The graph G.
Output: All objects C of G.
Let S_0 be a root object in G;
ReverseSearch(S_0);
Function: ReverseSearch(S)
    Output S as a solution;
    foreach S' ∈ Γ(S) do
        if f(S') = S then
            ReverseSearch(S');
```

Problem definition. Given an undirected graph G and a parameter $s \geq 1$, the goal of this paper is to enumerate: (1) all maximal s-defective cliques, and (2) all maximal s-defective cliques with size q no less than $s + 2$ (i.e., $q \geq s + 2$).

It is already known that finding the maximum s-defective clique is NP-hard [12, 44], thus the problem of enumerating all maximal s-defective cliques is also NP-hard. As a consequence, there does not exist a polynomial-time algorithm to solve our problems unless NP = P. Moreover, the problem of enumerating all maximal s-defective cliques is typically more difficult than the classic maximal clique enumeration problem. This is because the number of maximal s-defective cliques is often much larger than the number of maximal cliques due to the relaxed constraint of the clique structure, thus requiring more time overheads to enumerate them.

To our knowledge, most previous solutions focus on finding the maximum s-defective clique [12, 44]. There is only one algorithm that may be used to enumerate maximal s-defective cliques [46]. However, as we discussed in Section 1, such an algorithm cannot ensure to obtain all maximal s-defective cliques. This motivates us to develop new approaches to efficiently solve the maximal s-defective clique enumeration problems. In the following sections, we will develop two different types of algorithms to efficiently solve our problems on real-world graphs.

3 A POLYNOMIAL-DELAY ALGORITHM

In this section, we develop an output-sensitive algorithm to enumerate all maximal s-defective cliques whose time complexity mainly relies to the number of results. A nice feature of this algorithm is that it can output two consecutive maximal s-defective cliques within polynomial time. Such an algorithm is inspired by the classic reverse search technique [2], which was originally developed for enumerating vertices in polyhedra. Below, we first briefly describe the reverse search technique and then present our solutions.

3.1 A Brief Overview of Reverse Search

To make use of the reverse search technique for enumeration, a graph-like structure which captures the relationships of all enumeration results must be defined first, which is shown in Definition 2.

Definition 2 ([2]). Given a graph G, let $\mathcal{C}$ be the set of all objects of G that needs to be output. Then, the graph-like structure is defined as follows:

- *Root: a unique root object in G can be found in polynomial time.*

- *Neighbors: the set of neighbors of an object $S \in C$, denoted by $\Gamma(S)$, i.e., it is possible for S to jump to each object in $\Gamma(S)$.*
- *Parent: projection from a non-root object S to its unique neighbor by a local search function $f(S)$, which satisfies that the root object can be found by a finite number of local search functions from S, i.e., $f^k(S) = f(f(...f(S))) = root$, where k is a finite positive integer.*

Based on Definition 2, it is easy to see that all objects in C form a connected graph, revealing which objects are allowed to jump from one to another. Then, with the parent relationship and the local search function, any non-root object in C has a unique and acyclic path from it to the root object, which forms a spanning tree. Thus, all objects can be output by a depth-first search (DFS) starting from the root object in the graph. Such an enumeration technique is called reverse search which is detailed in Algorithm 1. As shown in [2], Algorithm 1 has several interesting theoretical properties.

THEOREM 3.1. *Let $t(f)$ and $t(\Gamma)$ be the time used to compute f and Γ. Then, the time complexity of Algorithm 1 is $O(K\omega(t(\Gamma) + t(f)))$, where $K = |C|$ and $\omega = max_{S \in C}|\Gamma(S)|$.*

THEOREM 3.2. *Algorithm 1 is polynomial delay if both f and Γ can be computed within polynomial time.*

3.2 The Proposed Reverse Search Algorithm

As shown in Algorithm 1, the framework of the reverse search technique is quite implicit; and the definition of a suitable graph-like structure for different problems is also quite challenging. In the following, we propose a novel graph-like structure to characterize the relationships among all maximal s-defective cliques of a graph G, based on which a new enumeration algorithm is proposed.

Devising the graph-like structure. Given a set S of vertices, we denote by $S_{<v}$ the subset of vertices in S that are smaller than v, i.e., $S_{<v} = \{u \in S | u < v\}$. Similarly, $S_{\leq v}$ is defined as $S_{\leq v} = \{u \in S | u \leq v\}$. For two vertex sets S_1 and S_2 of G, we call $S_1 < S_2$ if S_1 is lexicographically smaller than S_2. Then, given a set S of G, we use $Extend(S)$ to denote the lexicographically smallest maximal s-defective clique that contains S. For instance, given a graph shown in Fig. 1(a) and $s = 1$, suppose that $S = \{v_2, v_5\}$, we have $Extend(S) = \{v_1, v_2, v_3, v_5\}$. Based on these notations, we define the graph-like structure for enumerating all maximal s-defective cliques as follows.

Definition 3 (Root). Given a graph G, we define the root node of G as the maximal s-defective clique which is the lexicographically smallest among all maximal s-defective cliques, i.e., $root = Extend(\{v_1\})$.

Definition 4 (Neighbors). Let C be the set of all maximal s-defective cliques of G. For a maximal s-defective clique S, its neighbor set is defined as $\Gamma(S) = \{S' \in C | S' \cap S \neq \emptyset\}$.

Example 2. Given a graph shown in Fig. 1(a) and a maximal 1-defective $S_1 = \{v_1, v_2, v_3, v_5\}$, the maximal 1-defective cliques $S_2 = \{v_1, v_2, v_5, v_6\}$ and $S_3 = \{v_2, v_3, v_4, v_5\}$ are the neighbors of S_1.

Given a maximal s-defective clique S of G, we refer to a vertex $v \in S$ as the *critical vertex*, denoted by π_S, if v is smallest in S such that $Extend(S_{\leq v}) = S$, i.e., $\pi_S = argmin_{v \in S}\{Extend(S_{\leq v}) = S\}$. Then, we define the parent of maximal s-defective clique S in G.

Algorithm 2: $GenerateNbrs(S, u)$

```
1  N_S ← ∅; X ← S \ N_u(G); P ← S ∩ N_u(G);
2  for each I ⊆ X s.t. |I| ≤ s do
3      S' ← I ∪ P ∪ {u};
4      if S' is a maximal in G(S ∪ {u}) then
5          Add S' into N_S;
6      else if the missing edges in G(S') is larger than s then
7          Let s' be the number of missing edges in G(S')
             (s' ≤ s + |I|);
8          for each I' ⊆ P s.t. |I'| ≤ s' − s do
9              S'' ← S' \ I';
10             if S'' is maximal in G(S ∪ {u}) then
11                 add S'' into N_S
12 return N_S;
```

Definition 5 (Parent). Given a graph G and two maximal s-defective cliques A and B, we call A the parent of B if $A \in \Gamma(B)$ and $A = Extend(B_{<v_p})$, where $v_p = \pi_B$.

Example 3. Given a graph G shown in Fig. 1(a), let $S_1 = \{v_1, v_2, v_4, v_5\}$ and $S_2 = \{v_2, v_3, v_4, v_5\}$ be the two maximal 1-defective cliques of G. For a maximal 1-defective clique $S_3 = \{v_4, v_5, v_6\}$, we can see that the parent of S_3 is S_1 instead of S_2. Although both S_1 and S_2 are the neighbors of S_3, the critical vertex π_{S_3} of S_3 is v_6 and we have $Extend(S_{3<v_6}) = Extend(\{v_4, v_5\}) = \{v_1, v_2, v_4, v_5\} = S_1$. Thus, only S_1 can be the parent of S_3.

Determining all neighbors. Although the graph-like structure of maximal s-defective cliques of G has been established, it is still unclear how to compute all neighbors of a maximal s-defective clique. Our solution to tackle this issue is based on the following observation. Note that in Algorithm 1, only the child nodes of a maximal s-defective clique S (here the child nodes represent the maximal s-defective cliques whose parent is S) have an opportunity to enter into the next recursion (which are what we exactly want), while the other neighbors can be ignored. Thus, we only need to find all possible child nodes of a maximal s-defective clique instead of all neighbor nodes.

By Definition 5, for the parent node A of a maximal s-defective clique S, we have the following two relationships: 1) $\pi_S \in S \setminus A$ and 2) $S_{<\pi_S} \subset A$ (see Lemma 3). This means that, for any two maximal s-defective cliques A and B adjacent to each other, if they do not satisfy these two relationships, A (or B) must not be the child node of B (or A). For example, consider a maximal 1-defective clique $S_1 = \{v_1, v_2, v_4, v_5\}$ shown in Fig. 1(a), and a neighbor node $S_2 = \{v_2, v_3, v_4, v_5\}$ of S_1. We can see that $\pi_{S_2} = v_4$ and $S_{2<v_4} = \{v_2, v_3\} \not\subset S_1$, Then, S_2 is not the child node of S_1 and such a node can be omitted if S_1 being processed.

As a consequence, all possible child nodes of the maximal s-defective clique can be computed by the following approach. Given a parent node S, for each vertex $u \notin S$ of G, we first find each possible subset $S' \subseteq S_{<u}$ satisfying that $S' \cup \{u\}$ is an s-defective clique. Then, we use the *Extend* function to guarantee the maximality so that the possible child nodes of S containing $S' \cup \{u\}$ can be obtained. Interestingly, we also notice that $S' \cup \{u\}$ can be maximal in $G(S_{<u} \cup$

Algorithm 3: The proposed reverse search algorithm for enumerating all maximal s-defective cliques.

Input: The graph G and a parameter s.
Output: All maximal s-defective cliques of G.

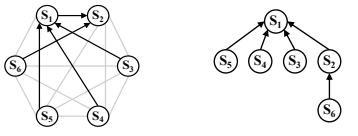
```
1  Let v_1, ..., v_n be the vertices in V;
2  S ← Extend({v_1});
3  Enum(S, 0);
4  Function: Extend(S)
5      for i = 1 to n do
6          if S ∪ {v_i} is the s-defective clique s.t. v_i ∉ S then
7              S ← S ∪ {v_i};
8      return S;
9  Function: Enum(S, i)
10     if i is odd then Output S;
11     forall u in V \ S s.t. u > π_S do
12         N_S ← GenerateNbrs(S_<u, u);
13         for each S' ∈ N_S do
14             R ← Extend(S');
15             if R ≥ S' s.t. S = Extend(S'_<u) then
16                 Enum(R, i + 1);
17     if i is even then Output S;
```

$\{u\}$) (Lemma 5), because for any child node B of S with $\pi_B = u$, it always satisfies that $B_{<\pi_B} \subset S$ and $B_{<\pi_B}$ is maximal in $G(V_{<\pi_B})$. The detailed procedure is shown in Algorithm 2.

In Algorithm 2, it admits two parameters $S_{<u}$ and u, where S is a maximal s-defective clique and u is a vertex of G such that $u \notin S$ (here we assume that all vertices in S are less than u). Then, the algorithm first initializes three sets $\mathcal{N}_S$, X, and P (line 1), where X and P are subset of S in which each vertex is non-adjacent and adjacent to u, respectively. Note that the set $P \cup \{u\}$ must be an s-defective clique in $G(S \cup \{u\})$ since all vertices in P are adjacent to u and P is an s-defective clique. This means that a maximal s-defective clique in $G(S \cup \{u\})$ must contain no vertex or at most s vertices in X. Therefore, the algorithm recursively selects a subset I from X such that the size of I does not exceed s to detect whether $S' = I \cup P \cup \{u\}$ is a maximal s-defective clique in $G(S \cup \{u\})$ (lines 2-11). If not, we need to remove a subset I' in P from S' to eliminate conflicts (lines 6-10). Here the size of I' is no larger than $|I|$ since the missing edges in $G(S')$ is no larger than $s + |I|$.

Example 4. Given a graph G shown in Fig. 1(a) and a maximal 1*-defective clique* $S_1 = \{v_1, v_2, v_4, v_5\}$*. Suppose that we need to find all possible child nodes of* S_1 *that contains* v_6*. Algorithm 2 first invokes* $S_{1<v_6} = \{v_1, v_2, v_4, v_5\} = S_1$ *and* v_6*. Then,* S_1 *can be partitioned as* $X = \{v_2, v_4\}$ *and* $P = \{v_1, v_5\}$*. Since* $s = 1$*, at most one vertex in X may form an s-defective clique with* v_6*. First, when setting* $I = \emptyset$*, we have* $S' = I \cup P \cup \{v_6\} = \{v_1, v_5, v_6\}$*, which is not maximal in* $G(S \cup \{v_6\})$*, thus we omit this result. Then, for* $I = \{v_2\}$*, we have* $S' = \{v_1, v_2, v_5, v_6\}$*. This result is valid and thus is added into the result set* $\mathcal{N}_S$*. When* $I = \{v_4\}$*, another result* $S' = \{v_4, v_5, v_6\}$ *is further put into* $\mathcal{N}_S$*. Finally, the algorithm terminates and returns* $\mathcal{N}_S$*. Note that an s-defective clique* $\{v_3, v_5, v_6\}$ *is ignored by the algorithm, because it is definitely not the child node of S based on Lemma 4.*

Result ID	Vertices
S_1	v_1, v_2, v_3, v_5
S_2	v_1, v_2, v_4, v_5
S_3	v_1, v_2, v_5, v_6
S_4	v_2, v_3, v_4, v_5
S_5	v_3, v_5, v_6
S_6	v_4, v_5, v_6



(a) The graph-like struture (b) Enumeration tree

Figure 2: A running example for enumerating all maximal 1**-defective cliques using reverse search.**

The overall algorithm. Equipping with the graph-like structure and the technique for computing all possible child nodes of an s-defective clique, we can devise the reverse search algorithm for enumerating all maximal s-defective cliques. The detailed algorithm is outlined in Algorithm 3. In Algorithm 3, it first labels each vertex in V with v_1 to v_n to ensure the lexicographical order of each maximal s-defective clique in G (line 1). Then, the algorithm determines the root node with Definition 3 (line 2) and invokes the DFS enumeration procedure *Enum* (line 3).

In *Enum*, it first determines an output order of maximal s-defective cliques (line 10 and line 17), which is determined by the depth of the current recursion (the main purpose of this trick is to reduce the output delay between consecutive solutions as used in [41]). Then, the procedure iteratively selects a vertex $u \in V$, satisfying that $u \notin S$ and $u > \pi_S$, to continue the computations (lines 11-16). Specifically, it first invokes Algorithm 2 to compute all possible child nodes of S (which may not be maximal) (line 12) and then calls the *Extend* function to maximize them (line 14). After that, the procedure checks each possible child node R whether it is really a child of S. If so, we continue the recursion with such a child node (line 16). Finally, The procedure terminates until all maximal s-defective cliques have been visited. An illustrative example for enumerating all maximal s-defective cliques of a graph G is shown in Fig. 2.

Example 5. Given a graph G shown in Fig. 1(a) and a parameter $s = 1$*, the graph-like structure for each maximal* 1*-defective clique of G can be determined by definitions, which is shown in Fig. 2(a). The reverse search starts from the root node* $S_1 = \{v_1, v_2, v_3, v_5\}$*, and computes all possible child nodes of* S_1 *using Algorithm 2. For the first possible child node* $S_2 = \{v_1, v_2, v_4, v_5\}$*, it can be seen that the parent of* S_2 *is exactly* S_1*. Then, the algorithm continues the recursive call with* S_2 *and finds the possible child nodes of* S_2*. Since* $\pi_{S_2} = v_4$*, two possible child nodes* $S_3 = \{v_1, v_2, v_5, v_6\}$ *and* $S_6 = \{v_4, v_5, v_6\}$ *of* S_2 *are detected, where only the parent of* S_6 *is* S_2*. The algorithm further processes the recursive call with* S_6*. Since no possible child node of* S_6 *is found, the algorithm backtracks to* S_1 *and finds the second possible child node* S_3*. The next recursive call with* S_3 *is also processed since the parent of* S_3 *is* S_1*. Finally, the algorithm terminates until the remaining two child nodes* $S_4 = \{v_2, v_3, v_4, v_5\}$ *and* $S_5 = \{v_3, v_5, v_6\}$ *of* S_1 *are found. It can bee seen that the parent node of* S_6 *is not* S_1*, thus* S_6 *is ignored in the recursive call with* S_1*.*

3.3 Correctness Analysis

In this subsection, we analyze the correctness of Algorithm 3. Below, we first give several useful lemmas which will be used to prove the correctness of Algorithm 3.

Lemma 1. The function $Extend(S)$ *in Algorithm 3 returns the lexicographically smallest maximal s-defective clique containing S.*

PROOF. Let A be the set returned by $Extend(S)$. Note that A must be maximal. This is because given any maximal s-defective clique S' containing S, for each $v \in S'$, $S'_{<v} \cup S$ is clearly an s-defective clique and v is the next vertex to be added. Thus, we only need to show that A is lexicographically smallest. Suppose on the contrary that there is an s-defective clique B containing S such that $B < A$. Let v be the smallest vertex in $B \setminus A$. Then, we have $B_{<v} \subset A$. Since the s-defective clique satisfies the hereditary property, we can derive that $B_{<v} \cup S$ is an s-defective clique and the vertex v can be the next vertex to be added. However, since $v \notin A$, this means that the vertex v cannot be used to expand $Extend(B_{<v} \cup S)$, which is a contradiction. □

Lemma 2. For any two s-defective cliques A and B with $A \subseteq B$, we have $Extend(A) \leq Extend(B)$.

PROOF. Let C_A and C_B be the set of all s-defective cliques containing A and B respectively. Then, we have $C_B \subseteq C_A$. Since $Extend(A)$ and $Extend(B)$ return the lexicographically smallest one from C_A and C_B respectively, we have $Extend(A) \leq Extend(B)$. □

Lemma 3. Given two maximal s-defective cliques A and B, if A is the parent of B, we have $\pi_B \in B \setminus A$ and $B_{<\pi_B} \subset A$.

PROOF. If A is the parent of B, we can easily derive that $B_{<\pi_B} \subset A$ by Definition 5. Note that $B_{<\pi_B} \neq A$, because $B_{<\pi_B}$ is not maximal while A is the maximal. Moreover, if $\pi_B \in A$, we can obtain that $A = Extend(B_{<\pi_B}) = Extend(B_{\leq\pi_B}) = B$, which is a contradiction. □

Lemma 4. Given two maximal s-defective cliques A and B, if $\pi_B \in A$ or $B_{<\pi_B} \not\subset A$, A must not be the parent of B.

PROOF. If $\pi_B \in A$, we can derive that $B_{<\pi_B} \not\subset A$, otherwise $A = Extend(B_{\leq\pi_B}) = B$. We only need to prove that if $B_{<\pi_B} \not\subset A$, then A is not the parent of B. Based on the fact that $B_{<\pi_B} \subseteq Extend(B_{<\pi_B})$, we have $Extend(B_{<\pi_B}) \neq A$ if $B_{<\pi_B} \not\subset A$, thus the lemma is established. □

Lemma 5. Given two maximal s-defective cliques A and B, if A is the parent of B, then $B_{\leq v}$ is a maximal s-defective clique in $G(A_{<v} \cup \{v\})$, where $v = \pi_B$.

PROOF. Based on Lemma 3, if A is the parent of B, we have $B_{<v} \subseteq A_{<v}$. Moreover, based on the fact that $B = Extend(B_{\leq v})$, there is no any vertex u in V such that $u < v$ can be added into $B_{\leq v}$ to form a larger s-defective clique, otherwise v is not the critical vertex of B. This indicates that any vertex in $A_{<v} \setminus B_{<v}$ cannot be added to $B_{\leq v}$. □

Lemma 6. Given two maximal s-defective cliques A and B, if A is the parent of B, then we have $A < B$ and $\pi_A < \pi_B$.

PROOF. Note that $\pi_A < \pi_B$ is obvious, since $A = Extend(B_{<\pi_B})$ and π_A must be the vertex no greater than the largest vertex in $B_{<\pi_B}$. Thus, we only need to prove $A < B$. Since $A = Extend(B_{<\pi_B})$ and $B = Extend(B_{\leq\pi_B})$, we have $A \leq B$ based on Lemma 2. Moreover, π_B is the critical vertex of B, which means that B cannot be equal to A. If not, a smaller vertex in B can be the critical vertex. Thus, we have $A < B$. □

Lemma 7. Let C be the set of all maximal s-defective cliques. For each non-root $S \in C$, the parent of S is unique.

PROOF. Suppose that there are two parents A and B of S. Based on Definition 5, we have $A = Extend(S_{<v})$ and $B = Extend(S_{<v})$, where $v = \pi_S$. Since $Extend(S_{<v})$ returns the lexicographically smallest maximal s-defective clique containing $S_{<v}$, thus we have $A = B$, which is contradiction. □

Based on Lemma 6 and Lemma 7, it is easy to see that the parent relationships for the maximal s-defective cliques of G form a spanning tree. Below, we prove that Algorithm 3 exactly generates such a spanning tree.

THEOREM 3.3. *Let $\mathcal{N}_S$ be the set returned by Algorithm 2 with parameters $S_{<u}$ and u, where S is an s-defective clique and u is a vertex in V with $u > \pi_S$ and $u \notin S$. Then, for each child node B of S with $u = \pi_B$, we have $B_{\leq u} \in \mathcal{N}_S$.*

PROOF. According to Lemma 5, given any child node B of S with $u = \pi_B$, there must be a maximal s-defective clique in $G(S_{<u} \cup \{u\})$ corresponding to $B_{\leq u}$. Thus, we only need to prove that Algorithm 2 returns all maximal s-defective cliques containing u in $G(S_{<u} \cup \{u\})$. In the algorithm, we define that X and P are the sets of vertices in $S_{<u}$ that are adjacent and non-adjacent to u, respectively. Since each vertex in the s-defective clique has at most s non-neighbors, there are at most s vertices in X that can form an s-defective clique with u. Given an arbitrary maximal s-defective clique A containing u in $G(S_{<u} \cup \{u\})$ and let $I = A \setminus (P \cup \{u\})$. We have $|I| \leq s$ and $I \subseteq X$. Let $S' = I \cup P \cup \{u\}$. Then, we have $A \subseteq S'$. Note that the number of missing edges in $G(S')$ is at most $s + |I|$, since $I \cup P$ is an s-defective clique and all vertices in P are the neighbors of u. Moreover, since A is maximal in $G(S_{<u} \cup \{u\})$, we have $|S' \setminus A| = |P \setminus A| \leq |I|$. This is because for each vertex w in $S' \setminus A$, there must be a vertex in A that is not adjacent to w; otherwise, the vertex w can be used to expand A. □

THEOREM 3.4. *Algorithm 3 correctly outputs all maximal s-defective cliques of G.*

PROOF. It is easy to see that $Extend(\{v_1\})$ yields the root maximal s-defective clique of G. Then, we prove that *Enum* can recursively visit each non-root maximal s-defective clique of G. To prove this, we only need to show that all possible child nodes of S are obtained and each child node of S is identified only once, where S is the maximal s-defective clique invoked by *Enum*. Based on Lemma 6, the critical vertex of each child node of S must be larger than π_S, which indicates that the algorithm only needs to iteratively select a vertex u from V with $u > \pi_S$ as the critical vertex (line 11). Moreover, by Theorem 3.3, each $B_{\leq u}$ is included in the result returned by Algorithm 2, where B is the maximal s-defective clique with $\pi_B = u$. Thus, all possible child nodes of S can be determined (lines 11-14). It remains to prove that (i) line 15 correctly identifies the child node of S, and (ii) the same child node of S cannot be invoked multiple times by *Enum*. Let us first consider the case (i). For each R (the possible child node) in line 14, if $\pi_R = u$, it is easy to see that R is the child node of S by Definition 5. For case (ii), if R is the child node of S but $\pi_R \neq u$, we can see that such a set R cannot be invoked by *Enum*. The reason is as follows. On the one hand, $\pi_R > u$ is impossible, since each $w \in S'$ is no larger than u. On the other hand, if $\pi_R < u$, there must have $R < S'$ since $\pi_R \notin S$ based on Lemma 3. □

3.4 Complexity Analysis

Here we analyze the time and the space complexity of Algorithm 3.

THEOREM 3.5. *The time complexity of $Extend(S)$ is $O(n\triangle)$, where $\triangle$ is the size of the maximum s-defective clique of G.*

PROOF. Given a vertex v, the time used to check whether v can be added to S is $O(|S|)$, since it takes constant time to determine whether an vertex $u \in S$ is a neighbor of v using an array. Note that $|S|$ is bounded by $\triangle$ and there are at most $|V|$ vertices to be checked. Thus, the time complexity of $Extend(S)$ is $O(n\triangle)$. □

THEOREM 3.6. *The time complexity of Algorithm 2 is $O(\frac{\triangle^{2s+2}}{4^s})$.*

PROOF. Suppose that the two input parameters of Algorithm 2 are S and u. Let $x_1 = |X|$ and $x_2 = |P|$, then we have $x_1 + x_2 = |X| + |P| = |S| \leq \triangle$. We can observe that there are at most x_1^s possible choices for I in X, since each subset I of X has at most s vertices. Moreover, in each $R = I \cup P \cup \{u\}$, at most $|I|$ vertices can be removed from P. Thus, the total number of iterations of Algorithm 2 to generate maximal s-defective cliques containing u in $G(S \cup \{u\})$ is at most $(x_1x_2)^s$. It is easy to see that $x_1x_2 \leq \frac{\triangle^2}{4}$, since $x_1+x_2 \leq \triangle$. In addition, the time overheads for detecting whether an s-defective clique is maximal in $G(S \cup \{u\})$ is bounded by $O(\triangle^2)$. Putting it all together, the time complexity of Algorithm 2 is $O(\frac{\triangle^{2s+2}}{4^s})$. □

THEOREM 3.7. *Algorithm 3 is a polynomial-delay algorithm, and its time delay to output two consecutive results is $O(n^2\frac{\triangle^{2s+1}}{4^s})$.*

PROOF. It is easy to see that the time cost of each recursive call is bounded by $O(n * P(s) + n|\mathcal{N}_S|n\triangle)$, where $P(s)$ denotes the cost of Algorithm 2. Based on Theorem 3.6, we have $P(s) = O(\frac{\triangle^{2s+2}}{4^s})$ and $|\mathcal{N}_S| = O(\frac{\triangle^{2s}}{4^s})$. Thus, the time complexity of each recursive call in Algorithm 3 is bounded by $O(n\frac{\triangle^{2s+1}}{4^s}(n+\triangle))$. Since $\triangle \leq n$, the time complexity of each recursive call is $O(n^2\frac{\triangle^{2s+1}}{4^s})$. As we analyzed before, Algorithm 3 yields a spanning tree of all maximal s-defective cliques. This means that the time complexity of Algorithm 3 is $O(Kn^2\frac{\triangle^{2s+1}}{4^s})$, where K is the number of maximal s-defective cliques of a graph G. Moreover, we also adjust the output order of solutions. When the depth of a recursive call is odd, we output the result before processing sub-recursions; and when the depth of the recursive call is even, the result is output after finishing the sub-recursions. Based on the results in [41], such a trick can reduce the time delay to the time consumption of one recursive call. Therefore, the time delay of Algorithm 3 is $O(n^2\frac{\triangle^{2s+1}}{4^s})$. □

THEOREM 3.8. *The space complexity of Algorithm 3 is $O(n\triangle)$.*

PROOF. Since Algorithm 3 adopts a DFS strategy, its space complexity is bounded by the maximum depth multiplied by the maximum space usage of one recursive call. Based on Lemma 6, we can derive that maximum depth is bounded by $O(n)$. Moreover, in each recursive call, we observe that the set $\mathcal{N}_S$ may lead to a very large space consumption. To reduce this, once we get a result by Algorithm 2, we immediately execute lines 13-15. This shows that there is no need to use the set $\mathcal{N}_S$ to store all results generated by Algorithm 2, resulting in that Algorithm 2 only consumes $O(\triangle)$ space. In addition, *Extend* consumes $O(\triangle)$ space to store a maximal s-defective clique. Thus, we obtain that the space complexity of Algorithm 3 is $O(n\triangle)$. □

4 NEW PIVOT-BASED ALGORITHMS

Although the proposed reverse search algorithm has a nice theoretical property, its practical performance may be not very well, as each maximal s-defective clique has too many neighbors. In this section, we propose several branch-and-bound enumeration algorithms based on a novel pivoting technique. Interestingly, we show that the time complexity of all the proposed pivot-based enumeration algorithms can be bounded by $O(n\alpha_s^n)$, where α_s is a real number strictly less than 2. For instance, if $s = 0$, $s = 1$ and $s = 2$, we have $\alpha_0 = 1.618$, $\alpha_1 = 1.839$, and $\alpha_2 = 1.928$, respectively.

In the rest of this paper, we will focus more on the problem (2) as defined in Section 2, i.e., enumerating all s-defective cliques with size q no less than $s+2$. This is because small s-defective cliques are often no practical use in real-world applications. Moreover, as shown in Section 2, the diameter of a maximal s-defective clique with size $q \geq s+2$ is no larger than 2, thus it is more desirable to represent a densely-connected community. It is important to note that although we focus on problem (2), all the proposed branch-and-bound enumeration techniques can also be easily adapted to enumerate all maximal s-defective cliques (i.e., for problem (1)). Below, we first present a basic branch-and-bound enumeration algorithm, which forms a basis for developing our advanced techniques.

4.1 A Basic Branch-and-Bound Algorithm

The main idea of the basic branch-and-bound algorithm is that each s-defective clique either contains a particular vertex v in G or does not contain v. Thus, given a graph G, the original problem can be divided into two sub-problems. The first sub-problem is to enumerate all maximal s-defective cliques containing a vertex v in G; and another one is to enumerate all maximal s-defective cliques excluding v. Such a method can be recursively applied to enumerate all maximal s-defective cliques, and the pseudocode is shown in Algorithm 4.

In Algorithm 4, it invokes the *BranchEnum* procedure to enumerate all maximal s-defective cliques. This procedure admits three parameters: S, C, and X, where S is a partial s-defective clique, C is the candidate set in which each vertex is used to expand S, and X is the exclusion set containing all vertices that have been processed form C. Initially, the sets S and X are the empty sets while C is set to V. Then, in each recursive call, the algorithm first checks if S is maximal (lines 3-5). If not, it selects a vertex v from C to perform two sub-recursive calls. The first one is the enumeration of all maximal s-defective cliques containing $S \cup \{v\}$ (line 9); and the other one is invoked to enumerate all maximal s-defective cliques that contain S but not v (line 10). Note that before invoking the first recursive call, the sets C and X need to be updated to ensure that $S \cup \{v, u\}$ forms an s-defective for each u in C and X (lines 7-8). Whenever the set C is empty, the algorithm terminates.

It is easy to verify that Algorithm 4 can correctly enumerate all maximal s-defective cliques of G. The worst-case time complexity of Algorithm 4 is $O(n\triangle 2^n)$, which is analyzed in Theorem 4.1.

THEOREM 4.1. *Algorithm 4 outputs all maximal s-defective cliques of G in $O(n\triangle 2^n)$ time.*

Algorithm 4: The basic branch-and-bound algorithm.

Input: The graph G and two parameters s and $q \geq s + 2$.
Output: All relatively-large maximal s-defective cliques of G.

```
1  BranchEnum(∅, V, ∅);
2  Function: BranchEnum(S, C, X)
3      if C = ∅ then
4          if X = ∅ s.t. |C| ≥ q then Output S;
5          return;
6      Select a vertex v from C, and let S' ← S ∪ {v};
7      C' ← {u ∈ C \ {v} | S' ∪ {u} is an s-defective clique};
8      X' ← {u ∈ X | S' ∪ {u} is an s-defective clique};
9      BranchEnum(S', C', X');
10     BranchEnum(S, C \ {v}, X ∪ {v});
```

Proof. Clearly, the time complexity of Algorithm 4 is the sum of the time taken by each recursive call in the algorithm. Since the time spend of each recursive call is bounded by $O(n\triangle)$, it only needs to determine the maximum number of recursive calls in Algorithm 4. Given a graph with size n, we denote by $T(n)$ the maximum number of recursive calls in Algorithm 4. Then, we have $T(n) \leq T(n-x) + T(n-1)$, where $T(n-x)$ and $T(n-1)$ correspond to the sub-recursive calls in line 8 and line 9, respectively. In the worst-case, we have $x = 1$, as the candidate set can be reduced by at least 1. Thus, we further obtain that $T(n) \leq T(n-1)+T(n-1) = 2T(n-1) = 2^n T(0)$. Clearly, the maximum number of recursive calls is bounded by 2^n, which indicates that the worst-case time complexity of Algorithm 4 is bounded by $O(n\triangle 2^n)$. □

It can be seen that the basic algorithm is very inefficient because there are a large number of branches that produces non-maximal s-defective cliques. In the following, we will propose several efficient techniques to reduce the unnecessary computations.

4.2 A Novel Pivot-based Enumeration Algorithm

To speed up the basic branch and bound algorithm, the key point is to determine the enumeration branches that definitely produce non-maximal s-defective cliques. We observe that only the sub-recursive calls for enumerating all maximal s-defective cliques that excludes v may generate non-maximal s-defective cliques. The reason is as follows. Suppose that C is the candidate set and a maximal s-defective clique $A \subseteq C$ containing v is generated by the first sub-recursive call (i.e., the sub-recursion for enumerating all maximal s-defective cliques containing v). It is easy to see that the set $A \setminus \{v\} \subseteq C$ will be included in the candidate set of the second sub-recursive call (the sub-recursion for enumerating all maximal s-defective cliques excluding v). Clearly, a non-maximal s-defective clique $A \setminus \{v\}$ will be explored by the second sub-recursive call which results in unnecessary computations. To optimize the second sub-recursive call, we propose a novel pivoting technique which is described as follows.

The pivoting technique. The main idea of this technique is based on the fact that for any maximal s-defective clique B that does not contain v, there must be a vertex $u \in B$ such that $u \notin A$, where A is an arbitrary s-defective clique containing v. Therefore, given a candidate set $C \setminus \{v\}$ with $A \subseteq C$, if we select the vertex u ($u \notin A$) to expand the current s-defective clique S in the sub-recursive call to enumerate all maximal s-defective cliques excluding v, then all s-defective cliques contained in $A \setminus \{v\}$ definitely cannot be generated by such a sub-recursive call. This is because any maximal s-defective clique generated by the sub-recursive call must contain a vertex u, and the set $A \setminus \{v\}$ does not contain the vertex u. Moreover, we further notice that any vertex in $A \setminus \{v\}$ is not necessary to be used to expand S (in the sub-recursive call that enumerates all maximal s-defective cliques excluding v). Since for any maximal s-defective clique in $C \setminus \{v\}$, there must be a vertex in $C \setminus A$ that can be used to expand to obtain it. This indicates that all maximal s-defective cliques that exclude v can be generated by only expanding the vertices in $C \setminus A$. Based on this idea, we develop a novel pivoting technique for enumerating all maximal s-defective cliques, which is shown in the following lemma.

Lemma 8. Given three sets S, C and X in a recursive call, let v be a vertex in C with $S \subseteq N_v(G)$. If the first sub-recursive call is used to enumerate all maximal s-defective cliques containing v, then all vertices in $C \cap N_v(G)$ are no need to expand S in the other sub-recursive call.

Proof. Note that the vertex v satisfies that $S \subseteq N_v(G)$. Thus, for any maximal s-defective clique in $S \cup C$ that contains S but not v, it must contain a vertex in $C \setminus N_v(G)$. Suppose on the contrary that there is a maximal s-defective clique B ($S \subseteq B$ and $v \notin B$) in $S \cup C$ with $(B \setminus S) \subseteq N_v(G)$. Then, we can derive that all vertices in B are the neighbors of v. Since B is an s-defective clique, $B \cup \{v\}$ is also an s-defective clique. As a consequence, B is non-maximal, which is a contradiction. Based on our previous analysis, if there is a vertex in $C \setminus N_v(G)$ for an arbitrary maximal s-defective clique excluding v, then it must be generated by a sub-recursive call that expand S with only vertices in $C \setminus N_v(G)$. Thus, the lemma is established. □

We note that directly applying the pivoting technique in each recursive call may not significantly improve the efficiency. For example, let us reconsider the graph G shown in Fig. 1(a). Let $S = \emptyset$, $C = \{v_1, v_2, ..., v_6\}$, and $X = \emptyset$ be the three input sets of the recursive call to enumerate all maximal 1-defective cliques. If we select the vertex v_2 as the pivot vertex, the first sub-recursive call with $S' = \{v_2\}$ and $C' = \{v_1, v_3, ..., v_6\}$ is invoked to enumerate all maximal 1-defective clique containing v_2. Then, based on our pivoting technique (Lemma 8), the second sub-recursive call with $S = \emptyset$ and $C = \{v_1, v_3, ..., v_6\}$ only selects $v_6 \in C \setminus N_{v_2}(G) = \{v_6\}$ as the pivot vertex and then enumerates all maximal 1-defective cliques containing v_6. We observe that all maximal 1-defective cliques have been generated so far. However, if we further perform the pivoting technique, the vertices in $C \setminus N_{v_6}(G) = \{v_3, v_4\}$ will be used to expand $S = \emptyset$ in the sub-recursive call that is invoked with $S = \emptyset$ and $C = \{v_1, v_3, v_4, v_5\}$ to enumerate the maximal 1-defective cliques, which incurs redundant computations.

Pivot-based branching rule. Interestingly, we find that only a part of recursive calls in the branch-and-bound algorithm is required to perform the pivoting technique. Let the top recursive call be the root branch. For each recursive call, we refer to the sub-recursive call that enumerates maximal s-defective cliques containing a particular vertex in the candidate set as the first-child, and the other sub-recursive call as the second-child. Then, we propose the following *pivot-based*

Algorithm 5: The pivot-based branch-and-bound algorithm.

Input: The graph G and two parameters s and $q \geq s + 2$.
Output: All relatively-large maximal s-defective cliques of G.

1 $PivotEnum(\emptyset, 0, \{(v, 0) | v \in V\}, \emptyset, \emptyset)$;
2 **Function:** $PivotEnum(S, r, C_1, C_2, X)$
3 **if** $C_1 = \emptyset$ **then**
4 **if** $C_2 \cup X = \emptyset$ *s.t.* $|S| \geq q$ **then** Output S;
5 **return**;
6 Select a pivot element (v, c_v) from C_1 such that c_v is maximum among all element in C_1;
7 **if** $c_v = 0$ *s.t.* $C_2 = \emptyset$ **then**
8 $C_2 \leftarrow \{(u, c_u) \in C_1 | u \in N_v(G)\}$;
9 $C_1 \leftarrow \{(u, c_u) \in C_1 | u \notin N_v(G)\}$;
10 $C_1' \leftarrow UpdateSet(S \cup \{v\}, r + c_v, v, C_1 \cup C_2)$;
11 $X' \leftarrow UpdateSet(S \cup \{v\}, r + c_v, v, X)$;
12 $PivotEnum(S \cup \{v\}, r + v_c, C_1', \emptyset, X')$;
13 $PivotEnum(S, r, C_1 \setminus \{(v, c_v)\}, C_2, X \cup \{(v, c_v)\})$;
14 **Function:** $UpdateSet(S, r, v, C)$
15 $C' \leftarrow \emptyset$;
16 **foreach** $(u, c_u) \in C$ **do**
17 **if** $u \notin N_v(G)$ **then** $c_u \leftarrow c_u + 1$;
18 **if** $v \neq u$ *s.t.* $r + c_u \leq s$ **then**
19 $C' \leftarrow C' \cup \{(u, c_u)\}$;
20 **return** C';

branching rule: in the branch-and-bound algorithm, only the root branch and the first-child of each recursive call require to perform the pivoting technique, while for the second-child of each recursive call, we inherit the pivoting results (i.e., the candidate vertices that can be used to expand) from its parent branch to continue the computation.

Example 6. Given a graph G shown in Fig. 1(a). Let the parameter $s = 1$. In the root branch, if v_2 is selected as the pivot vertex, its second-child with $S = \emptyset$ and $C = \{v_1, v_3, ..., v_6\}$ can only select v_6 to continue the recursive calls (because only v_6 is a non-neighbor of the pivot vertex v_2). For a new second-child with $S = \emptyset$ and $C = \{v_1, v_3, v_4, v_5\}$, it inherits its parent's pivoting results based on our branching rule, resulting in that no vertex can be used to expand the current s-defective clique (because its parent can only select v_6 to expand, but $v_6 \notin C$ in the current recursion), and thus the recursion terminates.

The pivot-based branching algorithm. With the proposed pivot-based branching rule, we then develop a novel pivot-based branch-and-bound algorithm. Unlike the basic branch-and-bound algorithm, we need to partition the candidate set into two disjoint sets C_1 and C_2. The benefits of doing this are that (i) when performing the pivot-based technique in a recursion, C_1 and C_2 are the candidate sets that need and do not need to be used to expand S, respectively; (ii) it can also be used to distinguish whether the current recursion is the first or second-child branch, i.e., if C_2 is not empty, it must be the second-child branch. The detailed pseudocode is shown in Algorithm 5.

In Algorithm 5, it invokes the *PivotEnum* procedure to perform recursive calls, which requires five parameters: S, r, C_1, C_2, and X, where the parameters C_1 and C_2 are the two disjoint candidate sets

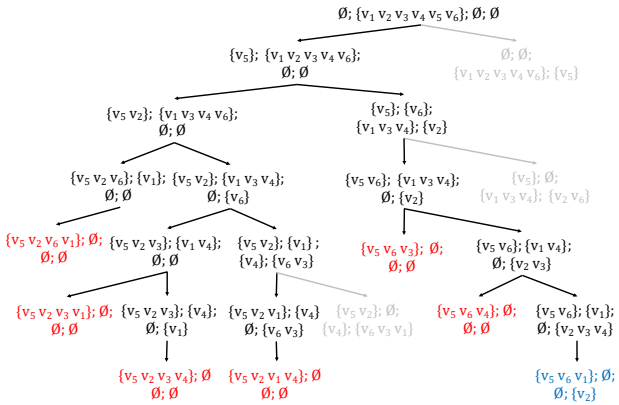


Figure 3: The enumeration tree of our pivot-based enumeration algorithm with $s = 1$ (The sets used in each recursive call correspond to the current 1-defective clique S, candidate set C_1, candidate set C_2, and excluded set X from left to right).

respectively, and r is a parameter to record the number of missing edges in $G(S)$. Initially, the sets S, C_2, and X are set to empty, while C_1 contains all vertices in V (line 1). Note that each element of the three sets C_1, C_2, and X is a pair (v, c_v), where v is the vertex and c_v is the number of non-neighbors of v in S. Here the purpose of using such a *pair representation* is to improve the efficiency for updating the sets C_1, C_2, and X. Specifically, when a vertex v is added to S, for each (u, c_u) in C_1, C_2, and X, we can determine whether u can form an s-defective clique with $S \cup \{v\}$ by simply checking whether u and v are adjacent, since c_u has recorded the number of non-neighbors of u in S. Thus, the overall updating time is linear by using this trick, which is detailed in the procedure $UpdateSet$ (lines 14-20).

In each recursive call, if $C_1 \cup C_2 \cup X$ is empty, the algorithm outputs S as a result (line 4). Otherwise, the algorithm recursively performs the branch-and-bound procedure if $C_1 \neq \emptyset$ (lines 6-13). More specifically, it first picks a pivot element (v, c_v) in C_1, where c_v is maximum among all elements in C_1 (line 6). If such a pivot vertex v satisfies $S \subseteq N_v(G)$ and also $C_2 = \emptyset$ (lines 7), the algorithm makes use of Lemma 8 to partition the original C_1 into two new candidate sets C_1 and C_2 such that all vertices in C_1 are the neighbors of v and C_2 does not contain any neighbors of v (lines 8-9). Otherwise, the original sets C_1 and C_2 are directly used. The algorithm then obtains the sets $C_1' \subset (C_1 \cup C_2)$ and $X' \subseteq X$ by invoking the $UpdateSet$ procedure, and invokes the first sub-recursive call to expand S with v (lines 10-12). Note that in this sub-recursive call, all candidate vertices are included in C_1', while C_2' is empty. This is because only the second sub-recursive call may produce redundant computations. After that, the algorithm moves the pivot vertex (v, c_v) from C_1 to X and makes use of the second sub-recursive call to continue the recursions until the set C_1 is empty (line 3 and line 13). Below, we give an example to illustrate the basic idea of this algorithm.

Example 7. Given a graph G shown in Fig. 1(a) and a parameter $s = 1$. Algorithm 5 first initializes the sets S, C_2, and X to empty, and the set C_1 to V. In the root branch, we assume that v_5 is the pivot vertex. The algorithm then divides the candidate set into $C_1 = \emptyset$ and $C_2 = \{v_1, ..., v_4, v_6\}$, since all vertices in $V \setminus \{v_5\}$ are the neighbors of v_5. Thus, the second-child with $S = \emptyset$, $C_1 = \emptyset$, and

$C_2 = \{v_1, ..., v_4, v_6\}$ will terminate immediately because $C_1 = \emptyset$. For the first-child with $S = \{v_5\}$ and $C_1 = \{v_1, ..., v_4, v_6\}$, another pivot vertex v_2 is further selected, which leads to that its candidate sets can be partitioned as $C_1 = \{v_6\}$ and $C_2 = \{v_1, v_3, v_4, v_5\}$. Subsequently, this recursion further generates two sub-recursive calls to continue the branch-and-bound procedure. For a new second-child, it can only expand $S = \{v_5\}$ with v_6, since C_1 contains only v_6. Such a second-child is finished until all maximal s-defective cliques containing $\{v_5, v_6\}$ are obtained, and the whole algorithm terminates when all vertices in C_1 for each recursion are computed. The complete enumeration tree of Algorithm 5 is shown in Fig. 3.

4.3 Correctness and Complexity Analysis

Here we analyze the correctness and complexity of Algorithm 5.

THEOREM 4.2. *Algorithm 5 outputs all maximal s-defective cliques of G exactly once, and no non-maximal s-defective clique is output.*

PROOF. First, by Lemma 8, it is easy to see that all maximal s-defective cliques are output by Algorithm 5. This is because every maximal s-defective clique either contains the pivot vertex v or contain at least one non-neighbor of v. The proposed pivoting technique applied in each recursion exactly captures this property, thus no maximal s-defective clique will be missed by our algorithm. Second, let S be an arbitrary maximal s-defective clique of G. When S is first output in a recursive call, there must be a vertex v in S is pushed into the exclusion set X, which results in that S cannot be output by the remaining recursive calls. As a consequence, any maximal s-defective clique is output only once by Algorithm 5. Third, suppose on the contrary that A is a non-maximal s-defective clique output by Algorithm 5. Then, A must be maximal in $G(S \cup C_1 \cup C_2 \cup X)$, where $S \subseteq A$, C_1, C_2, and X are the parameters invoked by a certain recursive call. Since A is not maximal in G, there must exist a vertex v in $V \setminus A$ such that $v \notin C_1 \cup C_2 \cup X$ but $S \cup \{v\}$ is an s-defective clique, which leads to a contradiction. Putting all it together, the theorem is established. □

THEOREM 4.3. *Given a graph G with n vertices, the time complexity of Algorithm 5 is bounded by $O(n\alpha_s^n)$, where α_s is the largest real root of function $x^{s+3} - 2x^{s+2} + 1 = 0$ and it is strictly smaller than 2. In particular, when s = 0, 1, and 2, we have $\alpha_s = 1.618, 1.839$, and 1.928, respectively.*

PROOF. Before proving this, we first give a theoretical result of a linear recurrence [19]. Let $T(n)$ be the maximum number of leaves in a search tree executed by a recursive algorithm with an input graph G having n vertices. Assume that each recursive branch can produce at most $r \geq 2$ sub-branches and for each sub-branch $i \leq r$, it can reduce the size of the current branch by at least $a_i > 0$. Then, a linear recurrence is obtained, as shown in the following inequality

$$T(n) \leq T(n - a_1) + T(n - a_2) + ... + T(n - a_r). \tag{1}$$

Based on the results in [19], the maximum size of $T(n)$ is bounded by $O(\alpha^n)$, where α is the maximum real root of $x^n - \sum_{i=1}^{r} x^{n-a_i} = 0$. Here α is also called the branching factor of the recurrence.

Next, we analyze the recurrence of Algorithm 5. Since the algorithm adopts a binary search procedure, then we have the following recurrence for Algorithm 5

$$T(n) \leq T(n - a_1) + T(n - a_2). \tag{2}$$

We observe that both sub-recursive calls in Algorithm 5 can reduce the size of the candidate sets by at least 1, which implies $a_1 = 1$ and $a_2 = 1$ in the worst case. Thus, the size of $T(n)$ in Eq. (2) is bound by $O^*(2^n) = O(f(n)2^n)$, where $f(n)$ is a polynomial function of n. However, a tighter bound can be obtained by an in-depth analysis. Let the current s-defective clique S and the candidate set C be the input parameters of a recursion. Denote by $N_S(C)$ the common neighbors of S in C, i.e., $N_S(C) = \{v \in C | S \subseteq N_v(G)\}$. In each recursion, we use $\overline{d}_S$ to denote $|C \setminus N_S(C)|$. The branching rules of Algorithm 5 can be divided in three cases.

(1) If $\overline{d}_S > s$, a pivot vertex v in $C \setminus N_S(C)$ is immediately used for expanding S, which leads to the number of missing edges in the current s-defective clique that increases by at least 1. Moreover, in the sub-recursive call that enumerates the maximal s-defective cliques containing $S \cup \{v\}$, another pivot vertex u in $C \setminus N_{S\cup\{v\}}(C)$ is also selected. Based on Definition 1, there are at most s vertices in $C \setminus N_S(C)$ can be added to S. We can observe that for the recursion with current s-defective clique $S \cup S'$ satisfying $|S'| = s$ and $S' \subset (C \setminus N_S(C))$, only the common neighbors of S can be included in the candidate set. This implies that the number of leaves of a recursion $T(n - s)$, expanding $S \cup S'$, can be corresponding to $T(n - \overline{d}_S)$. Thus, we have the following recurrence

$$T(n) \leq \sum_{i=1}^{s} T(n - i) + T(n - \overline{d}_S). \tag{3}$$

Based on $\overline{d}_S > s$, the minimum size of $\overline{d}_S$ is $s + 1$.

(2) If $\overline{d}_S = 0$, let v be the pivot vertex in C. Then, the candidate set is partitioned as two disjoint subsets $C_1 = C \cap N_v(G)$ and $C_2 = C \setminus N_v(G)$ based on Lemma 8. If $d_v(C) < |C| - s$, in a recursion with the current s-defective clique $S \cup \{v\}$, we observe that $\overline{d}_{S\cup\{v\}} > s$, which corresponds to the case (1). So, combining with Eq. (3), we have

$$T(n) \leq \sum_{i=1}^{s+1} T(n - i) + T(n - \overline{d}_{S\cup\{v\}}), \tag{4}$$

where $\overline{d}_{S\cup\{v\}} \geq s + 2$ because $d_v(C) < |C| - s$. Otherwise, if $d_v(C) \geq |C| - s$, there are at most $|C_1| \leq s$ vertices available to expand S in a recursion that enumerates all maximal s-defective cliques excluding v with the pivot-based technique. Then, we have

$$T(n) \leq \sum_{i=1}^{\overline{d}_v+1} T(n - i). \tag{5}$$

Here $\overline{d}_v \leq s$ denotes the number of non-neighbors of the pivot vertex v in C, i.e., $\overline{d}_v = |C \setminus N_v(G)|$ in each recursion.

(3) If $\overline{d}_S > 0$ and $\overline{d}_S \leq s$, all vertices in $C \setminus N_S(C)$ are used to expand S. After that, in the recursion with the current s-defective clique $S \cup S'$ with $S' = C \setminus N_S(C)$, each vertex in $C \cap N_S(C)$ can be adjacent to all vertices in $S \cup S'$ in the worst case, which means that the recurrence of this sub-recursion corresponds to case (2), because $\overline{d}_{S\cup S'} = 0$. Note that if $\overline{d}_{S\cup S'} \neq 0$, we can

obtain a recurrence of Eq. (3). Combining with case (2), in the worst case, we have

$$T(n) \leq \sum_{i=1}^{\overline{d}_S} T(n-i) + \sum_{i=\overline{d}_S+1}^{s+1} T(n-i) + T(n-\overline{d}_H). \quad (6)$$

Here $H = S \cup S' \cup \{u\}$, and u is a pivot vertex in the recursion with the current s-defective clique $S \cup S'$, which implies that $\overline{d}_H \geq s+2$.

With the above analysis, in the worst case, the recurrence of Algorithm 5 is corresponding to $T(n) \leq \sum_{i=1}^{s+2} T(n-i)$. As a result, the number of recursive calls of the pivot-based algorithm is bound by $O(\alpha_s^n)$, where α_s is the maximum real root of $x^{s+3} - 2x^{s+2} + 1 = 0$. Note that Algorithm 5 yields a binary search tree, so the total number of non-leaves is less than the number of leaves. In addition, in each recursion, the time complexity is bounded by $O(n)$, since the time overhead of lines 7-10 is linear to the size of the candidate set. Putting all it together, the theorem is established. □

THEOREM 4.4. *The space complexity of Algorithm 5 is $O(n\triangle)$, where $\triangle$ is the size of the maximum s-defective clique of G.*

PROOF. Initially, the candidate set is set as V, so that each recursive call uses no more than $O(n)$ space. Then, we notice that each second-child in the algorithm can directly use its parent datas (i.e, the sets S, C_1, C_2 and X), which means that all second-recursive calls of Algorithm 5 does not consume any space. As a result, the space usage is dominated by the maximum depth of the first-child branch (Algorithm 5 forms a depth-first search tree). Since each time the first-child is invoked, there is always a vertex in the candidate set C_1 used to expand current s-defective clique S, which means that its maximum depth can be bounded by $O(\triangle)$. Thus, Algorithm 5 uses $O(n\triangle)$ space.

□

Remark. Note that based on Lemma 8, only the vertex $v \in C_1$ satisfying $S \subseteq N_v(G)$ is considered as the pivot vertex. If the conditions in line 7 of Algorithm 5 hold in a recursive call, then every vertex v in C_1 satisfies $S \subseteq N_v(G)$ and can be used as a pivot vertex. To achieve a good efficiency, we can re-select a vertex that has the most number of neighbors in the candidate set C_1 as the pivot vertex.

4.4 Optimization Techniques

In this subsection, we propose several optimization techniques to further improve the efficiency of the proposed pivot-based algorithm. The first optimization is a degeneracy ordering for enumerating all maximal s-defective cliques. With the help of this technique, the time complexity of the proposed pivot-based algorithm can be further improved to $O(n^{s+2}\alpha_s^{\delta})$. Here δ is the degeneracy of a given graph G [29], which is often very small in real-world graphs [18, 28]. The other optimizations are aimed to reduce the size of the candidate set in enumerating all relatively-large maximal s-defective cliques. Below, we first introduces the concept of the degeneracy ordering.

Definition 6 (Degeneracy ordering). Given a graph G, the degeneracy ordering of vertices in V is an ordering of $v_1, v_2, ..., v_n$ such that for each v_i, its degree is minimum in $G(\{v_i, ..., v_n\})$.

Algorithm 6: The improved pivot-based algorithm.

Input: The graph G and two parameters s and $q \geq s+2$.
Output: All relatively-large maximal s-defective cliques of G.

1 Let $v_1, v_2, ..., v_n$ be the degeneracy ordering of vertices in V;
2 **for** $i = 1$ *to* n **do**
3 $\quad C \leftarrow V_{>v_i} \cap N_{v_i}(G)$; $\overline{C} \leftarrow V_{>v_i} \setminus N_{v_i}(G)$; $X \leftarrow V_{<v_i}$;
4 $\quad$ **foreach** $I \subseteq \overline{C}$ *s.t.* $|I| \leq s$ **do**
5 $\quad\quad I \leftarrow I \cup \{v_i\}$; $s' \leftarrow$ the number of missing edges in $G(I)$;
6 $\quad\quad$ **if** $s' \leq s$ **then**
7 $\quad\quad\quad$ Obtain $C' \subseteq \{(u, c_u) | u \in C\}$ and $X' \subseteq \{(u, c_u) | u \in \overline{C} \cup X \setminus I\}$ such that for each $(u, c_u) \in C' \cup X'$, $c_u = |I \setminus N_u(G)|$ and $s' + c_u \leq s$;
8 $\quad\quad\quad$ $PivotEnum(I, s', C', \emptyset, X')$;

To obtain the degeneracy ordering of the vertices in a graph G, we can use a peeling technique to iteratively remove the lowest-degree vertex from G [3]. Such a vertex-removal ordering is the degeneracy ordering, which can be computed in $O(m+n)$ time [3].

The key idea of our improved pivot-based algorithm is based on an observation that given a vertex $v \in V$, each maximal s-defective clique including v contains at most s non-neighbors of v. This indicates that to obtain all maximal s-defective cliques including v, it only needs to enumerate all maximal s-defective cliques containing $I \cup \{v\}$, for each I satisfying $|I| \leq s$ and $I \subseteq \overline{N}_v(G)$. Here the candidate set of $I \cup \{v\}$ must be a subset included in $N_v(G)$. Thus, we obtain an improved algorithm which is shown in Algorithm 6.

In Algorithm 6, it first computes the degeneracy ordering of the vertices in V (line 1). Then, for each vertex v_i in V, it computes all maximal s-defective cliques containing v_i (lines 2-8). Before performing recursive calls, the algorithm partitions the task into several sub-tasks based on our previous observation (lines 4-7). In particular, the algorithm first divides the candidate set into two disjoint sets C and $\overline{C}$, where C ($\overline{C}$) includes all neighbors (non-neighbors) of v_i that occur after v_i in the degeneracy ordering (line 3). In line 3, the set $V_{>v_i}$ is defined as $\{v_j \in V | j > i\}$. Next, it selects each subset I of $\overline{C}$ with $|I| \leq s$ and enumerates all maximal s-defective cliques that contain $I \cup \{v_i\}$ (lines 4-8). Note that the candidate set of the recursion with $I \cup \{v_i\}$ only contains the neighbors of v_i, since for any maximal s-defective clique S containing v_i, there always exists a recursion with $I \cup \{v_i\}$ satisfying $I \subset \overline{C}$ and $I = S \setminus N_{v_i}(G)$. Finally, the algorithm terminates when all vertices in V have been processed.

THEOREM 4.5. *The time and space complexity of Algorithm 6 are $O(n^{s+2}\alpha_s^{\delta})$ and $O(n * min(\triangle, \delta))$, respectively, where α_s is the largest real root of $x^{s+3} - 2x^{s+2} + 1 = 0$ which is strictly smaller than 2, and δ is the degeneracy of the graph G.*

PROOF. For the time complexity, it is easy to see that line 8 takes $O(n\alpha_s^{|C'|})$ time to finish the computation by Theorem 4.3. Note that the size of C' is no larger than δ based on the degeneracy ordering. Since there are at most $O(n^{s+1})$ times to invoke the procedure in line 8, the total time complexity of Algorithm 6 is bounded by $O(n^{s+2}\alpha_s^{\delta})$. For the space complexity, the maximum depth of the first-recursive calls in Algorithm 6 is bounded by $O(min(\triangle, \delta))$ since $|C'| \leq \delta$. Thus, the space complexity of Algorithm 6 is $O(n * min(\triangle, \delta))$. □

Optimization techniques for the problem (2). Here we further propose several additional optimization techniques for enumerating all maximal s-defective cliques with size no less than q ($q \geq s+2$).

Lemma 9. Given any s-defective clique S with the size of q ($q = |S|$), for each pair of vertices u and v in S, we have:

- $|N_v(S) \cap N_u(S)| \geq q - s - 2$ *if* $(u, v) \in E$.
- $|N_v(S) \cap N_u(S)| \geq q - s - 1$ *if* $(u, v) \notin E$.

Proof. Denote by $\overline{N}_v(S) = S \setminus N_v(S)$ for each $v \in S$, where $v \in \overline{N}_v(S)$. Based on Definition 1, we have $|\overline{N}_v(S)| \leq s+1$ for each $v \in S$. Moreover, let s_v be the size of $\overline{N}_v(S)$ for each $v \in S$. We then obtain that $s_v + s_u \leq s+2$ for each $(u, v) \in E_S$ since $|\overline{E}_S| \leq s$. Thus, we have $|N_u(S) \cap N_u(S)| \geq |S| - (s_v + s_u) \geq q - s - 2$. For each pair of vertices u and v in S with $(u, v) \notin E$, we can derive that $s_v + s_u \leq s+1$. This means that $|N_u(S) \cap N_u(S)| \geq q - s - 1$. □

Before performing the recursive procedure to enumerate the maximal s-defective cliques with size no less than q ($q \geq s+2$), we can make use of Lemma 9 to remove unnecessary vertices in the candidate set. The details are as follows. Let C be the candidate set and $S = \{v\}$ be the partial result, we first partition the set C into two subsets C_1 and C_2 such that $C_1 = C \cap N_v(G)$ and $C_2 = C \setminus N_v(G)$. Then, we iteratively remove the vertices in C_1 whose degree in $G(C_1)$ is less than $q - s - 2$. Let $C_1' \subseteq C_1$ be the remaining vertices satisfying $d_u(C_1') \geq q - s - 2$ for each u in C_1'. For each vertex u in C_2, we remove it from C_2 if the vertex satisfies $|N_u(G) \cap C_1'| < q - s - 1$. After that, all the remaining vertices form the new candidate set of a recursive call. Clearly, such a pruning process can be done in linear time with respect to the number of edges in $G(C)$.

Interestingly, we can derive a more general result based on Lemma 9 as shown in the following lemma.

Lemma 10. Given any s-defective clique S with the size of q ($q = |S|$), for each subset $R \subseteq S$, we refer to $N_R(S)$ as the number of common neighbors of R in S, then we have

$$|N_R(S)| \geq q - |R| - s + r, \tag{7}$$

where r is the number of missing edges in $G(R)$.

Proof. For any subset R of S, there are at most $s - r$ vertices in $S \setminus R$ that has at least one non-neighbors in R. That is to say, the size of $A \subseteq (S \setminus R)$ is at most $s - r$ such that $R \setminus N_v(G) \neq \emptyset$ for each $v \in A$. This indicates that the size of $N_R(S)$ must be no less than $|S \setminus R| - (s - r)$. As a consequence, we can obtain $|N_R(S)| \geq |S \setminus R| - (s - r) = q - |R| - s + r$. □

Based on Lemma 10, an early termination criteria can be further used in each recursive call of our pivot-based algorithm. Specifically, in each recursive call, there always exists a vertex v in the candidate set C_1 selected to expand the current s-defective clique S. Before expanding S with v, we first compute the common neighbors of $S \cup \{v\}$ in the candidate set. Then, we make use of Lemma 10 to determine whether the sub-recursive call with $S \cup \{v\}$ needs to be computed. If not, we can early terminate the computation. Note that in Algorithm 5 and Algorithm 6, we have already known the common neighbors of $S \cup \{v\}$ using the $UpdateSet$ procedure. Therefore, such an early termination trick can be implemented in constant time in each recursive call.

Algorithm 7: The general pivot-based algorithm.

Input: A graph G.
Output: All maximal $\mathcal{P}$-subgraphs of G.

```
1  GenEnum(R, V, ∅, X);
2  Function: GenEnum(S, C1, C2, X)
3      if C1 = ∅ then
4          if C2 ∪ X = ∅ then Output S;
5          return;
6      Select a pivot vertex v from C1; C ← C1 ∪ C2 \ {v};
7      Obtain the sets C' ⊆ C and and X' ⊆ X such that for each
         u ∈ C' ∪ X', S ∪ {v, u} is a P-subgraph;
8      GenEnum(S ∪ {v}, C', ∅, X');
9      if C2 = ∅ then
10         Partition C into two disjoint subsets C1' and C2' such that all
             vertices in C2' are ignored when expanding S; C1 ← C1',
             C2 ← C2';
11     GenEnum(S, C1 \ {v}, C2, X ∪ {v});
```

5 A GENERAL PIVOTING PARADIGM

In this section, we generalize our pivot-based algorithm to enumerate all maximal subgraphs that satisfy the hereditary property. Given a subgraph $G(S)$ of G that has a property $\mathcal{P}$, if all subgraphs of $G(S)$ also have the property $\mathcal{P}$, we call $G(S)$ satisfies the hereditary property. For convenience, we refer to a subgraph as a $\mathcal{P}$-subgraph if it satisfies the hereditary property. Clearly, the basic branch-and-bound algorithm can enumerate all maximal $\mathcal{P}$-subgraphs of G. Such an algorithm, however, is inefficient because its worst-case time complexity is $O(f(n)2^n)$, where $f(n)$ is a polynomial function with respect to (w.r.t.) n. To improve the efficiency, we present a general pivot-based branch-and-bound algorithm which is shown in Algorithm 7.

Similar to our pivot-based algorithm for enumerating all maximal s-defective cliques, Algorithm 7 also needs two disjoint candidate sets C_1 and C_2, and it only expands the current $\mathcal{P}$-subgraph S using the vertices in C_1 (line 1 and line 6). Specifically, in each recursion, the algorithm selects a pivot v from C_1 to expand S, and then recursively enumerates all maximal $\mathcal{P}$-subgraphs that contains v (lines 7-8). To enumerate all the maximal $\mathcal{P}$-subgraphs excluding v, the algorithm first checks whether $C_2 = \emptyset$. If so, the algorithm partitions the current candidate set $C = C_1 \cup C_2 \setminus \{v\}$ into two new candidate sets C_1' and C_2' such that each vertex in C_2' is no need to expand S (lines 9-10). After that, the algorithm invokes the recursive procedure to enumerate all maximal $\mathcal{P}$-subgraphs excluding v (line 11).

The remaining technical issue needed to be addressed in Algorithm 7 is how to partition the candidate set C into two disjoint subsets C_1' and C_2' in line 10. Below, we propose a general partition approach to solve this issue.

Lemma 11. In each recursion, we let S be the current $\mathcal{P}$-subgraph and C be the candidate set. Suppose that A is an arbitrary maximal $\mathcal{P}$-subgraph containing $S \cup \{v\}$ that has already been identified by the algorithm, where $A \subseteq C \cup S \cup \{v\}$. Then, based on A, we can obtain a valid partition such that $C_2 = A \setminus (S \cup \{v\})$ and $C_1 = C \setminus C_2$.

PROOF. Since A is a maximal $\mathcal{P}$-subgraph containing $S \cup \{v\}$, we can obtain that for any maximal $\mathcal{P}$-subgraph $B \supseteq S$ in $G(S \cup C)$ with $v \notin B$, there must be a vertex $u \in B$ such that $u \in C \setminus A$. Then, if we select the vertex $u \in C \setminus A$ to expand S to enumerate the maximal $\mathcal{P}$-subgraphs that exclude v, the maximal $\mathcal{P}$-subgraph B must be obtained by the algorithm. This means that for any maximal $\mathcal{P}$-subgraph $B \supseteq S$ in $G(S \cup C \setminus \{v\})$, it must be generated by expanding S only with a vertex u in $C \setminus A$, since there always exists a vertex $u \in B$ such that $u \in C \setminus A$. □

By Lemma 11, it is easy to construct a partition if we have a maximal $\mathcal{P}$-subgraph containing $S \cup \{v\}$. So, the question is how to obtain such a subgraph in each recursion? In effect, Algorithm 7 can obtain such a subgraph *for free*. That is, the result returned by the recursive call in line 8 is exactly a maximal $\mathcal{P}$-subgraph that contains $S \cup \{v\}$. More interestingly, we prove that the time complexity of such a general pivot-based algorithm can be bounded by $O(f(n)\alpha^n)$, where α is a real number no larger than 2.

THEOREM 5.1. *Let p be the maximum size of C_1 in Algorithm 7. Then, the time complexity of Algorithm 7 is bounded by $O(f(n)\alpha^n)$, where $f(n)$ is a polynomial function w.r.t. n, and α is the maximum real root of $x^{p+1} - 2x^p + 1 = 0$, which is no larger than 2.*

PROOF. The proof is very similar to the proof of Theorem 4.3 by establishing a recursive relation $T(n) \leq \sum_{i=1}^{p} T(n-i)$, thus we omit the details for brevity. □

6 EXPERIMENTS

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

6.1 Experimental Setup

Algorithms. We implement four algorithms, denoted by RSA, Basic, Pivot, and Pivot+, to enumerate all maximal s-defective cliques, where RSA is the polynomial-delay algorithm shown in Algorithm 3, Basic is the basic branch-and-bound algorithm presented in Algorithm 4, Pivot is the proposed pivot-based branch-and-bound algorithm presented in Algorithm 5, and Pivot+ is the improved Pivot algorithm shown in Algorithm 6. Since there is no existing algorithms that can enumerate all maximal s-defective cliques, we use Basic as a baseline for comparison.

To enumerating relatively-large maximal s-defective cliques (with size no less than a given threshold q), we also implement two additional algorithms, called Pivot2 and Pivot2+. Pivot2 is a variant of Pivot which initializes the candidate set with 2-hop neighbors of a vertex v to enumerate maximal s-defective cliques containing v. Pivot2+ is an improved Pivot2 algorithm equipped with the optimization techniques shown in Lemma 9 and Lemma 10. Note that both Pivot2 and Pivot2+ are equipped with the degeneracy ordering technique. All algorithms are implemented in C++, and tested on a PC with one 2.2 GHz AMD CPU and 128GB memory running CentOS 7.6.

Datasets. We use two classes of datasets in the experiments: (1) the real-world graph dataset, and (2) the DIMACS benchmark dataset.

Table 1: Real-world graph datasets.

Datasets	n	m	$d_{\max}$	δ
scc-enron	146	9828	145	119
scc-rt-lolgop	273	4510	249	41
web-google	1299	2773	59	17
web-edu	3031	6474	104	29
ca-GrQc	4158	13422	81	43
web-spam	4767	37375	477	35
dblp-2010	226,413	1,432,920	238	74
ca-citeseer	227,320	1,628,268	1,372	86
wikipedia2009	1,864,433	9,014,630	2,624	66
flixster	2,523,386	15,837,602	1,474	68
socfb-B-anon	2,937,612	41,919,708	4,356	63

The real-world graph dataset contains 11 real-world networks, including collaboration networks, web graphs, social networks, scientific computing networks, and so on. The detailed statistics of these 11 real-world graphs are shown in Table 1, where d_{max} and δ denote the maximum degree and the degeneracy of the graph respectively. The DIMACS dataset contains 14 graphs which are the well-known benchmark graphs and widely used for evaluating the performance of different clique enumeration algorithms [11, 24, 42]. The detailed statistics of the DIMACS benchmark graphs are described in Table 3. Note that we use 6 small real-world graphs (the first six datasets in Table 1) and the DIMACS benchmark dataset to evaluate the performance of different algorithms for enumerating all maximal s-defective cliques, because both RSA and Basic are often too expensive to enumerate all results on large graphs. We use 5 large graphs (the last five datasets in Table 1) to evaluate the performance of different algorithms for enumerating relatively-large maximal s-defective cliques. All datasets can be downloaded from https://networkrepository.com/index.php.

Parameters. For all algorithms, we set the parameter s to be an integer falling in the interval $[1, 4]$ with a default value of 2. To enumerate relatively-large maximal s-defective cliques, we set the threshold parameter q to be an integer falling in the interval $[8, 20]$ with a default value of 12. We will study the effect of our algorithms with varying these two parameters.

6.2 Efficiency Testing

Exp-1: Results of enumerating all maximal s-defective cliques. In this experiment, we evaluate the performance of various algorithms to enumerate all maximal s-defective cliques. Table 3 and Table 2 show the runtime of RSA, Basic, Pivot, and Pivot+, on 14 DIMACS benchmark graphs and 6 small real-world graphs, respectively. The column #Nums in Table 3 and Table 2 denotes the number of maximal s-defective cliques in each graph with a specified parameter s. Note that we set the runtime of an algorithm to "INF" if it cannot terminate within 24 hours; and in column #Nums, "—" denotes that the number of maximal s-defective cliques in the graph is unclear. As can be seen, both Pivot and Pivot+ substantially outperform Basic. This result indicates that the proposed pivoting technique indeed plays a crucial role in pruning unnecessary computations of the enumeration procedure. More specifically, both Pivot and Pivot+ can be more than two orders of magnitude faster than Basic and RSA, which

Table 2: Running time of various algorithms on small real-world graphs (in seconds).

Datasets (n, m)	s	#Nums	RSA	Basic	Pivot	Pivot+	s	#Nums	RSA	Basic	Pivot	Pivot+
scc-enron	1	797	0.83	INF	**0.01**	0.47	3	614477	16811.20	INF	**0.22**	98.71
(146, 9828)	2	23563	116.41	INF	**0.04**	7.52	4	14614052	INF	INF	**2.18**	1062.73
scc-rt-lolgop	1	32725	4.02	INF	**0.01**	0.08	3	10645790	26075.29	INF	**1.32**	12.84
(273, 4510)	2	677261	466.1	INF	**0.08**	1.19	4	92934096	INF	INF	**10.09**	98.33
web-google	1	840519	85.63	3.49	0.73	**0.37**	3	368459600	INF	2777.89	305.72	**96.33**
(1299, 2773)	2	3559549	610.87	10.57	3.07	**1.72**	4	383858895	INF	3772.05	**340.16**	411.99
web-edu	1	4585535	1320.73	178.63	9.00	**3.25**	3	4648863524	INF	67828.74	9404.94	**2397.92**
(3031, 6474)	2	19425029	8085.40	241.86	36.63	**19.42**	4	4782150346	INF	INF	**9445.79**	11414.93
ca-GrQc	1	8631208	3012.07	INF	23.53	**7.86**	3	12119124523	INF	INF	33316.87	**8452.04**
(4158, 13422)	2	55474221	35055.38	INF	96.18	**50.52**	4	13450361630	INF	INF	**33560.66**	40650.56
web-spam	1	12579786	9334.69	98.96	35.16	**13.55**	3	18622643486	INF	INF	57014.37	**15403.21**
(4767, 37375)	2	184214200	INF	557.83	165.25	**108.17**	4	27068757394	INF	INF	**62245.90**	75138.01

Table 3: Running time of various algorithms on DIMACS graphs (in seconds).

Datasets (n, m)	s	#Nums	RSA	Basic	Pivot	Pivot+	s	#Nums	RSA	Basic	Pivot	Pivot+
brock200-2	1	9136255	443.71	22.43	5.70	**5.23**	3	708240243	INF	3607.93	329.05	**311.57**
(200, 9876)	2	97771482	10855.17	204.69	52.40	**48.90**	4	3925035986	INF	19254.09	1705.81	**1576.59**
brock200-4	1	490579863	34199.47	5054.66	456.30	**400.21**	3	54231210046	INF	INF	36223.94	**33551.67**
(200, 13089)	2	6249448243	INF	50942.17	4946.73	**4508.97**	4	—	INF	INF	INF	INF
c-fat200-2	1	16683	0.43	11.54	0.01	**0.00**	3	5036269	303.82	101.45	0.69	**0.65**
(200, 3235)	2	514378	20.40	19.45	0.10	**0.09**	4	31320769	3017.51	291.13	**3.62**	3.71
c-fat200-5	1	11434	1.37	INF	0.27	**0.14**	3	21125784	27401.87	INF	**10.78**	24.85
(200, 8473)	2	804813	393.90	INF	**1.74**	2.76	4	382282733	INF	INF	**59.95**	199.30
c-fat500-2	1	115651	4.95	212.24	0.12	**0.05**	3	54544685	6625.82	1441.31	11.52	**8.04**
(500, 9139)	2	4147354	106.10	285.53	0.76	**0.64**	4	315932832	61910.35	3931.06	45.47	**43.81**
c-fat500-5	1	101575	6.44	INF	1.39	**0.45**	3	233282147	INF	INF	**79.83**	101.65
(500, 23191)	2	8932296	339.78	INF	10.04	**9.37**	4	4114443303	INF	INF	**579.90**	1044.31
c-fat500-10	1	78131	60.39	INF	19.75	**6.26**	3	703374943	INF	INF	**2343.98**	4828.51
(500, 46627)	2	12617010	68633.97	INF	**225.21**	261.64	4	26893143878	INF	INF	**10480.94**	74025.94
Erdos981	1	99644	3.85	0.20	0.03	**0.02**	3	14736791	908.63	35.23	4.83	**2.27**
(445, 1381)	2	585822	37.73	0.60	0.16	**0.12**	4	17610021	1572.85	51.44	**6.75**	8.13
Erdos982	1	16908236	7993.86	142.02	62.84	**30.19**	3	32769884964	INF	INF	**30694.53**	INF
(5816, 9505)	2	54829408	42964.93	420.04	255.30	**130.20**	4	—	INF	INF	INF	INF
geom1	1	18954444	9510.26	166.73	75.11	**35.93**	3	38924146334	INF	INF	**38682.72**	INF
(6158, 11898)	2	72974355	60968.64	554.73	303.80	**153.75**	4	—	INF	INF	INF	INF
geom2	1	2148378	360.19	11.34	2.91	**1.44**	3	1485909377	INF	16107.70	1960.68	**564.86**
(2074, 3939)	2	8081903	2281.75	30.77	12.09	**6.54**	4	1510257369	INF	21552.36	**2111.91**	2558.94
hamming6-2	1	21468654	287.26	20079.28	**17.20**	25.16	3	1236781442	56606.33	INF	**789.47**	1110.12
(64, 1824)	2	192222338	5070.63	INF	**140.23**	201.43	4	6384044002	INF	INF	**3592.23**	4888.49
johnson8-4-4	1	2336490	32.46	8.38	**1.21**	1.43	3	133830510	5778.80	778.87	**50.77**	59.96
(70, 1855)	2	21883710	597.50	59.66	**9.39**	11.19	4	617991780	37552.01	3292.93	**222.32**	256.25
MANN_a9	1	9033660	38.25	275.13	**2.13**	3.61	3	315114132	3411.30	16728.72	**72.15**	123.37
(45, 918)	2	66409560	460.22	1596.76	**15.15**	26.17	4	1102044069	18983.18	53312.57	**264.71**	431.75

demonstrates the high-efficiency of our pivot-based algorithms. For instance, on ca-GrQc, Pivot and Pivot+ take only 23.53 and 7.86 seconds respectively to compute all maximal 1-defective cliques, while RSA takes 3012.07 seconds and Basic even cannot finish the computations within 24 hours.

When comparing the two pivot-based algorithms Pivot and Pivot+, we can observe that if s is no larger than 3, Pivot+ is often faster than Pivot on most datasets. This result confirms that the optimization techniques developed in Section 4.4 is very effective when s is small. When s getting larger, Pivot+ may generate more non-maximal s-defective cliques in the recursive procedure (Line 8 of Algorithm 6), thus degrading the performance of the algorithm. In practice, we suggest to use Pivot+ to enumerate all maximal s-defective cliques of a graph if $s \leq 3$, otherwise it is better to use Pivot.

We can also observe that RSA works well when s is small, but it is often very costly when $s \geq 3$. Moreover, we can see that for small dense graphs, RSA is often much faster than Basic when $s \leq 2$. For example, on scc-enron, RSA can output all maximal 1-defective cliques using only 0.83 seconds, while Basic cannot terminate within 24 hours. This result indicates that RSA is not only with theoretical interests, but it also works well in small real-world graphs given that $s \leq 2$. In addition, since RSA can output two consecutive results within polynomial time, it might be useful for the applications that only requires a part of results.

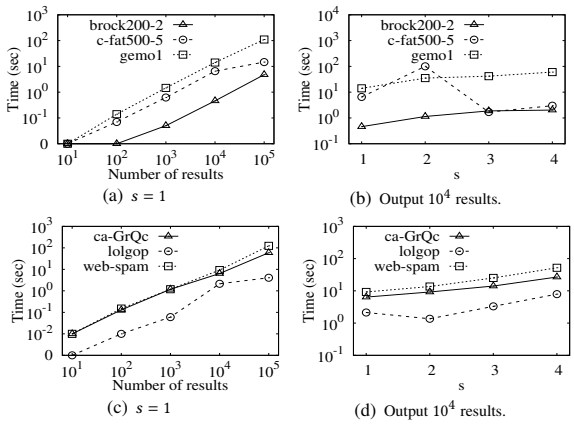


Figure 4: Delay of RSA on DIMACS and real-world graphs.

Exp-2: Time delay of RSA. In this experiment, we further evaluate the delay performance of RSA. Fig. 4 shows the runtime of RSA on DIMAC and small real-world graphs with varying s and varying the number of results. The results on other datasets are consistent. From Fig. 4, we can see that the runtime of RSA scales linearly with respect to (w.r.t.) the number of results, which confirms our polynomial-delay time complexity analysis of RSA in Section 3.4. Moreover, when specifying the desired number of output results (e.g 10^4 results), the runtime of RSA increases very smoothly as s increases on most datasets. This result further confirm that RSA can be very useful for the applications that only need to obtain a part of results even when $s > 2$.

Exp-3: Results of enumerating all relatively-large maximal s-defective cliques. Here we evaluate the efficiency of different algorithms to enumerate all relatively-large maximal s-defective cliques. Fig. 5 shows the results of Basic, Pivot2, and Pivot2+ on 5 large real-world graphs with varying s and q. Note that for a fair comparison, in this experiment, we also make use of the Property 2 to reduce the candidate set in the Basic algorithm. That is to say, the only difference between Basic and Pivot2 in this experiment is whether the proposed pivoting technique is used or not. From Fig. 5, we can observe that Pivot2+ consistently outperforms the other algorithms on all datasets. Specifically, under most parameter settings, Pivot2+ can be more than two orders of magnitude faster than Pivot2 and Basic. For instance, On socfb-B-anon, when $s = 1$ and $q = 20$, Pivot2+ and Pivot2 take 16.35 and 6468.2 seconds respectively, while Basic cannot finish the computation within 24 hours. This result indicates that the proposed pivoting technique is indeed very effective in improving the performance of enumerating maximal s-defective cliques. Moreover, when comparing Pivot2 and Basic, we can see that Basic cannot handle most large real-world graphs, while Pivot2 is still very efficient to process all graphs with most parameter settings. This result indicate that the proposed pivoting technique indeed plays an important role in reducing unnecessary computations.

Exp-4: The number of maximal s-defective cliques. In this experiment, we show the number of maximal s-defective cliques on each dataset by varying s. The results are shown in Fig. 7. Note that here we abbreviate the name of each dataset to reduce the space usage in the figures. As can be seen, the number of maximal s-defective cliques increases dramatically as s increases. This result suggests that the problem of enumerating maximal s-defective clique becomes more difficult when s getting larger. Indeed, when s increases, the runtime of all algorithms increase as shown in our previous experiments. However, as shown in Fig. 5, our pivot-based algorithm Pivot2+ is still very efficient even for relatively-large s (e.g., $s = 4$). For example, on socfb-B-anon, when $q = 20$, Pivot2+ takes only 16.35 and 189.52 seconds for $s = 1$ and $s = 4$ respectively, while the number of maximal 4-defective cliques is 470 times more than that of maximal 1-defective cliques as shown in Fig. 7.

Exp-5: Scalability testing. To evaluate the scalability of the proposed pivot-based algorithms, we generate eight subgraphs by randomly sampling 20-80% vertices or edges from the largest dataset socfb-B-anon. Similar results can also be observed on the other datasets. We run our algorithms on these subgraphs and report their runtime. Fig. 6 depicts the results of Pivot2 and Pivot2+ with varying s. As can be seen, the runtime of the proposed pivot-based algorithms increase smoothly as $|V|$ or $|E|$ increases. When selecting different value of s, our algorithms also exhibits a similar trend. This result indicates that the proposed pivot-based algorithms scale well on real-world graphs. In addition, Pivot2+ always shows excellent performance in all parameter settings compared to Pivot2, which further demonstrates that Pivot2+ is capable of handling large real-world graphs.

6.3 Effectiveness Testing

In this subsection, we evaluate the effectiveness of our solutions. To this end, we compare the s-defective clique model with the k-plex model which is a well-known relaxed clique model and is also closely-related to the s-defective clique model. Specifically, a subset C of V is called a k-plex if every vertex in $G(C)$ has at least $|C| - k$ neighbors. When $k = 1$, the k-plex is a clique which corresponds to the case of $s = 0$ for the s-defective clique. Moreover, it is easy to check that any s-defective clique is also an $(s + 1)$-plex. Therefore, the k-plex model is a very good baseline for our experiments.

Exp-6: The number of different relaxed cliques on large real-world graphs. Here we first compare the difference in the number of maximal s-defective cliques and the maximal $(s + 1)$-plexes on various large real-world graphs. Fig. 8 shows the results on each dataset with varying s. Note that the vertical coordinate in the figure represents the ratio of the number of maximal $(s + 1)$-plexes to the number of maximal s-defective cliques. "Unk." denotes that the number of maximal $(s + 1)$-plexes cannot be obtained within 24 hours by the state-of-the-art k-plex enumeration algorithms [45, 50]. From Fig. 8, we can see that the number of maximal $(s+1)$-plexes is consistently larger than the number of maximal s-defective cliques on each dataset with varying s. Moreover, with an increasing s, the gap between the number of maximal $(s+1)$-plexes and the number of maximal s-defective cliques becomes larger. This results suggest that the problem of enumerating all maximal k-plexes is more difficult than the problem of enumerating all maximal s-defective cliques.

Exp-7: Runtime of enumerating different relaxed cliques. In this experiment, we compare the runtime of our pivot-based algorithm (Pivot2+) and that of Plex (the state-of-the-art maximal $(s+1)$-plexes enumeration algorithms in [45, 50]). Fig. 9 shows the results on each

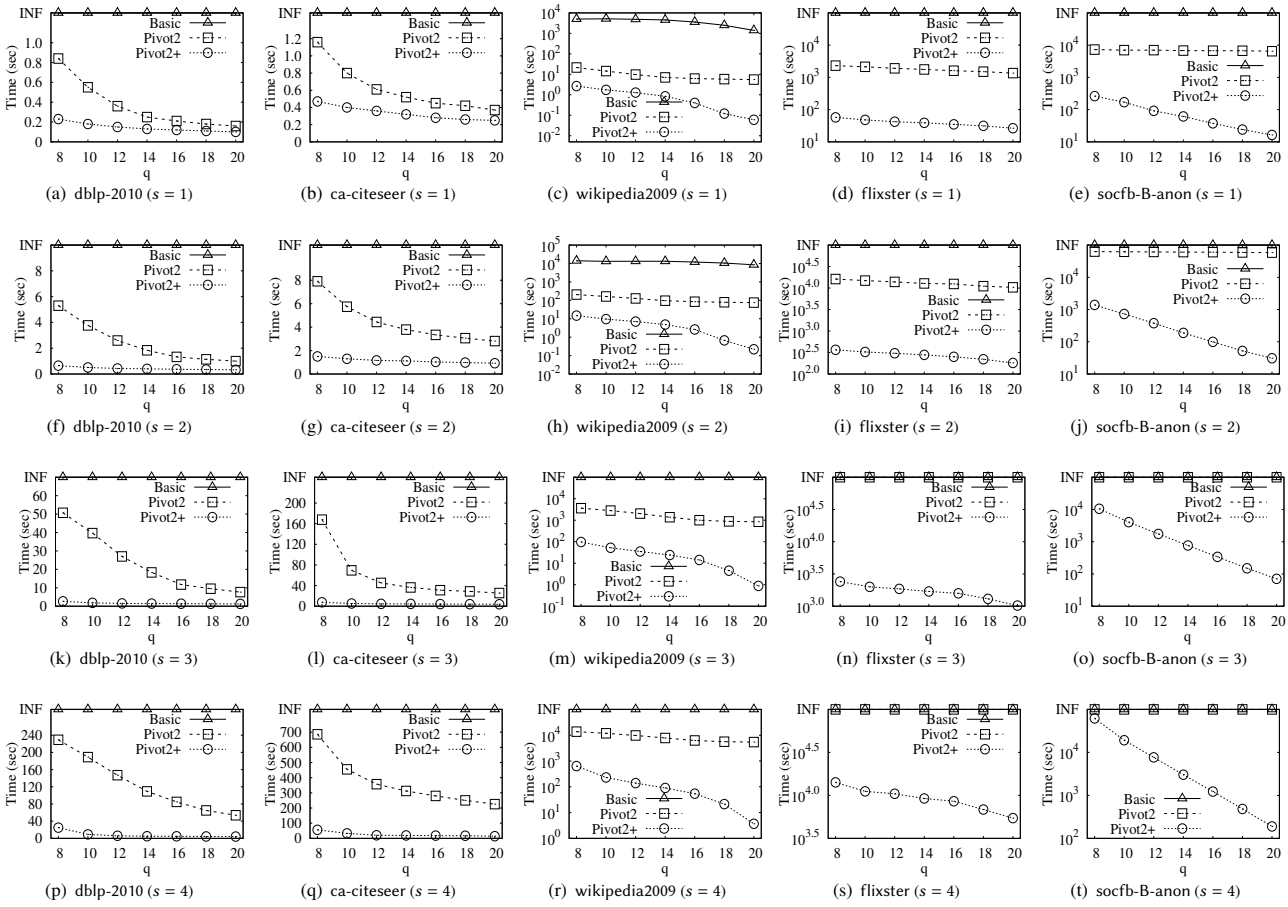


Figure 5: Runtime of different algorithms with varying s and q on large real-world graphs.

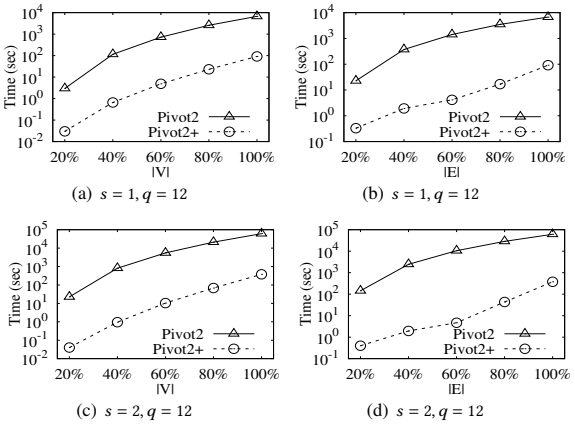


Figure 6: Scalability testing (socfb-B-anon).

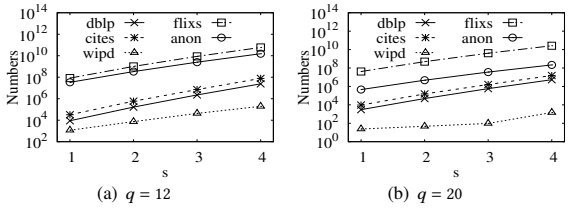


Figure 7: The number of s-defective cliques on different datasets with varying s and q

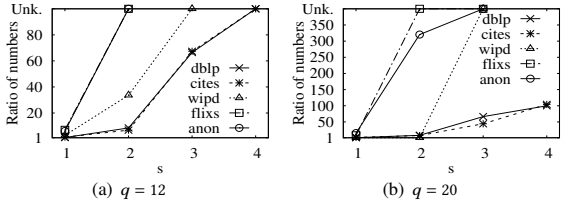


Figure 8: The ratios of the number of maximal $(s+1)$-plexes to the number of maximal s-defective cliques on various datasets with the same parameters.

dataset with varying s. The vertical coordinate in the figures represents the ratios of the runtime of Plex to the runtime of Pivot2+. "Unk." denotes that the Plex algorithm cannot complete the computations within 24 hours. From Fig. 9, we can observe that Pivot2+ exhibits much better performance than Plex on all datasets. More specifically, Pivot2+ can be more than two orders of magnitude faster than Plex on most parameter settings, and the gap between the runtime of Plex and Pivot2+ increases dramatically as s increases.

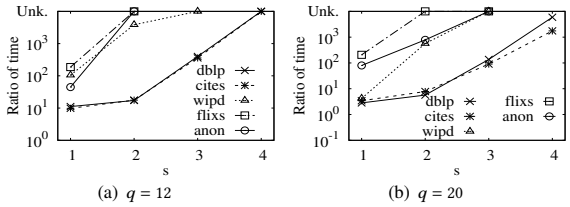


Figure 9: The ratios of the runtime of enumerating maximal $(s+1)$-plexes to the runtime of enumerating maximal s-defective cliques on various datasets with the same parameters.

This results confirm that the s-defective clique model is indeed easier to compute than the k-plex model.

Exp-8: Protein interaction prediction. In this experiment, we make use of the s-defective clique model to identify implicit protein interactions on a protein-protein interaction (PPI) network denoted by CORE [22]. Our solution is mainly based on an observation that the missing edges in a maximal s-defective clique can be considered as implicit interactions. First, we invoke the proposed algorithms to compute all maximal s-defective cliques with size no less than 10. Then, we add the missing edges of each detected maximal s-defective clique and test whether the added edges are the true positives. The $(s+1)$-plex model is also evaluated as a baseline method to compare the effectiveness of s-defective clique model. Table 4 shows the prediction accuracy of two different models (i.e, the ratio of the number of true positive edges to all added edges) on CORE [22], which contains 2708 vertices and 7123 edges and the ground truth protein interactions can be determined by the MIPS protein database [22]. From Table 4, we observe that the s-defective clique model is often more accurate than the $(s+1)$-plex model with varying s (except for $s = 1$, and in this case the s-defective clique model is also comparable to the k-plex model). Moreover, with the increase of s, the accuracy gap between the s-defective clique model and the $(s+1)$-plex model tends to be larger. This result demonstrate the high effectiveness of our solutions for protein interaction prediction in PPI networks.

Exp-9: Collaboration relationship prediction on DBLP. Here we evaluate the effectiveness of the s-defective clique model to predict the collaboration relationship on DBLP. To this end, we use the DBLP collaboration network before 2011, and evaluate the prediction accuracy using the full DBLP dataset. Fig. 10 shows the results of Prof. Jiawei Han's communities which are generated by the s-defective clique model and the $(s+1)$-plex model. As can be seen, the s-defective clique model can exactly predict a new collaboration relationship (Prof. Charu C. Aggarwal and Prof. Xifeng Yan in Fig. 10(a)) with $s = 1$, while the $(s+1)$-plex model will introduce a false-positive collaboration relationship (Prof. Jian Pei and Prof. Xiaohui Gu in Fig. 10(b)). Moreover, when increasing s, (i.e, $s = 2$), the result obtained by the 3-plex model (Fig. 10(c)) yields more false positive relationships compared to the 2-defective clique model (Fig. 10(b)). This result further confirm the high effectiveness of the s-defective clique model for implicit interaction prediction in real-world graphs.

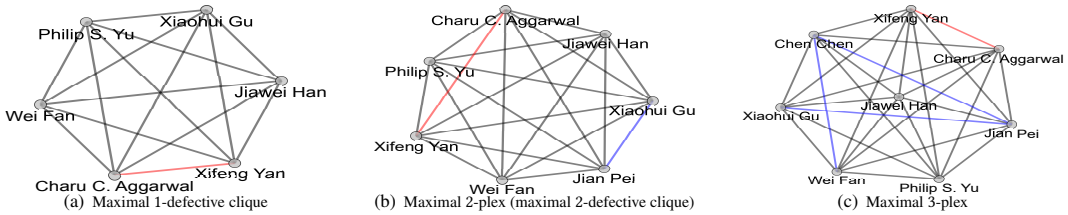
Table 4: Prediction accuracy on a PPI network (CORE).

s	$(s+1)$-plexes		s-defective cliques	
	#Nums.	Accuracy	#Nums.	Accuracy
1	206	**0.621**	160	0.6
2	673	0.513	569	**0.614**
3	2192	0.382	1390	**0.608**
4	29246	0.182	3530	**0.465**

7 RELATED WORKS

Maximal clique enumeration. The maximal clique enumeration problem had been well studied in recent decades [9, 10, 16, 18, 35, 43]. Perhaps, the most impressive technique is the Bron-Kerbosch (BK) algorithm [9], which is a backtracking enumeration algorithm with a pivoting technique. Following this work, many interesting variants of the BK algorithms were proposed. For example, Tomita et al. [43] proposed an $O(3^{n/3})$ BK algorithm with a careful-designed pivoting technique. Such an algorithm was shown to be optimal, as there exists an graph having $O(3^{n/3})$ maximal cliques in the worst case [33]. Eppstein et al. [18] presented an improved BK algorithm with the time complexity closely related to the degeneracy of the graph [29]. In addition, a refined pivot-selection technique [35] and a dominated-set based technique [10] were also developed to improve the BK algorithm when processing dense graphs. For sparse graphs, however, the bit-parallelism techniques were proposed to improve the BK algorithm [16, 39]. Recently, the maximal clique model was also generalized to other types of graphs. Notable examples include maximal cliques on uncertain graphs [27, 34, 51], maximal bipartite cliques on bipartite graphs [1, 25, 48], maximal cliques on temporal graphs [21, 38], and maximal signed cliques on signed graphs [13, 26]. Unlike all these mentioned maximal clique algorithms, this paper focuses mainly on the maximal s-defective clique model and developing efficient techniques to enumerate all maximal s-defective cliques.

Maximal relaxed clique enumerations. In real-world applications, using the clique model to mine cohesive subgraphs may be overly strict due to the presence of noises in graph datasets. Many relaxed clique models have been developed as alternatives. Except for the s-defective clique, several representative relaxed clique models include k-plex [6, 15, 40, 50], s-bundle [36, 49], s-clique [5, 32], and γ-quasi-clique [31, 37]. Existing algorithms for enumerating all these relaxed cliques can be classified into two categories. The first category of algorithms are to convert the original problem to the problem of enumerating all maximal relaxed cliques in each almost-satisfying subgraphs [14] if the relaxed clique model admits the hereditary property, such as the techniques presented in [5, 6]. The other category of algorithms are based on the set enumeration technique or the branch-and-bound technique to check whether each subset of a given graph is the maximal relaxed clique [15, 31, 37, 49, 50]. In addition, some of these models were also extended to different types of graphs. For instance, [47] developed an algorithm to enumerate all maximal k-plexes in bipartite graphs. [30] studied the problem of finding stable quasi-cliques in temporal graphs. In this work, we focus on developing efficient solutions with nontrivial time complexity guarantees to enumerate all and relatively-large s-defective cliques.



(a) Maximal 1-defective clique (b) Maximal 2-plex (maximal 2-defective clique) (c) Maximal 3-plex

Figure 10: Collaboration relationship prediction on DBLP (red and blue edges are the true and false positive collaborators respectively).

8 CONCLUSION

In this paper, we systematically investigate the maximal s-defective clique enumeration problem, and propose several novel and efficient algorithms to solve this problem. The first proposed algorithm is an output-sensitive algorithm based on a carefully-developed reverse search technique, which can return any consecutive solutions within polynomial time. The second proposed algorithm is a more practical algorithm based on a branch-and-bound enumeration and a novel pivoting technique. We prove that the time complexity of our pivot-based branch-and-bound algorithm can break the $O(2^n)$ worst-case enumeration complexity. We also develop several new pruning techniques to further improve the efficiency of our branch-and-bound algorithm. Additionally, we also extend our pivot-based branch-and-bound algorithm to enumerate all maximal subgraphs that satisfy a hereditary property. Finally, the results of comprehensive experiments demonstrate the efficiency, effectiveness, and scalability of the proposed approaches.

REFERENCES

[1] Aman Abidi, Rui Zhou, Lu Chen, and Chengfei Liu. 2020. Pivot-based Maximal Biclique Enumeration. In *IJCAI*. 3558–3564.

[2] David Avis and Komei Fukuda. 1996. Reverse Search for Enumeration. *Discret. Appl. Math.* 65, 1-3 (1996), 21–46.

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

[4] Punam Bedi and Chhavi Sharma. 2016. Community detection in social networks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 6, 3 (2016), 115–135.

[5] Rachel Behar and Sara Cohen. 2018. Finding All Maximal Connected s-Cliques in Social Networks. In *EDBT*. 61–72.

[6] Devora Berlowitz, Sara Cohen, and Benny Kimelfeld. 2015. Efficient Enumeration of Maximal k-Plexes. In *SIGMOD*. 431–444.

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

[10] Frédéric Cazals and Chinmay Karande. 2006. *Reporting maximal cliques: new insights into an old problem*. Ph. D. Dissertation. INRIA.

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

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

[13] Zi Chen, Long Yuan, Xuemin Lin, Lu Qin, and Jianye Yang. 2020. Efficient Maximal Balanced Clique Enumeration in Signed Networks. In *WWW*. 339–349.

[14] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. 2008. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *J. Comput. Syst. Sci.* 74, 7 (2008), 1147–1159.

[15] 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 *KDD*. 1272–1281.

[16] Naga Shailaja Dasari, Desh Ranjan, and Mohammad Zubair. 2014. pbitMCE: A bit-based approach for maximal clique enumeration on multicore processors. In *ICPADS*. 478–485.

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

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

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

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

[21] Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. 2016. Enumerating maximal cliques in temporal graphs. In *ASONAM*. 337–344.

[22] George Kollios, Michalis Potamias, and Evimaria Terzi. 2013. Clustering Large Probabilistic Graphs. *IEEE Trans. Knowl. Data Eng.* 25, 2 (2013), 325–336.

[23] Andrea Lancichinetti and Santo Fortunato. 2009. Community detection algorithms: a comparative analysis. *Physical review E.* 80, 5 (2009), 056117.

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

[25] Jinyan Li, Guimei Liu, Haiquan Li, and Limsoon Wong. 2007. Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A One-to-One Correspondence and Mining Algorithms. *IEEE Trans. Knowl. Data Eng.* 19, 12 (2007), 1625–1637.

[26] Rong-Hua Li, Qiangqiang Dai, Lu Qin, Guoren Wang, Xiaokui Xiao, Jeffrey Xu Yu, and Shaojie Qiao. 2018. Efficient Signed Clique Search in Signed Networks. In *ICDE*. 245–256.

[27] Rong-Hua Li, Qiangqiang Dai, Guoren Wang, Zhong Ming, Lu Qin, and Jeffrey Xu Yu. 2019. Improved Algorithms for Maximal Clique Search in Uncertain Networks. In *ICDE*. 1178–1189.

[28] Rong-Hua Li, Qiushuo Song, Xiaokui Xiao, Lu Qin, Guoren Wang, Jeffrey Xu Yu, and Rui Mao. 2022. I/O-Efficient Algorithms for Degeneracy Computation on Massive Networks. *IEEE Trans. Knowl. Data Eng.* 34, 7 (2022), 3335–3348.

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

[30] Longlong Lin, Pingpeng Yuan, Rong-Hua Li, Jifei Wang, Ling Liu, and Hai Jin. 2022. Mining Stable Quasi-Cliques on Temporal Networks. *IEEE Trans. Syst. Man Cybern. Syst.* 52, 6 (2022), 3731–3745.

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

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

[33] John W. Moon and Leo Moser. 1965. On cliques in graphs. *Israel journal of Mathematics* 3, 1 (1965), 23–28.

[34] Arko Mukherjee, Pan Xu, and Srikanta Tirthapura. 2017. Enumeration of Maximal Cliques from an Uncertain Graph. *IEEE Trans. Knowl. Data Eng.* 29, 3 (2017), 543–555.

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

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

[37] Jian Pei, Daxin Jiang, and Aidong Zhang. 2005. On mining cross-graph quasi-cliques. In *KDD*. 228–238.

[38] Hongchao Qin, Rong-Hua Li, Guoren Wang, Lu Qin, Yurong Cheng, and Ye Yuan. 2019. Mining Periodic Cliques in Temporal Networks. In *ICDE*. 1130–1141.

[39] Pablo San Segundo, Jorge Artieda, and Darren Strash. 2018. Efficiently enumerating all maximal cliques with bit-parallelism. *Comput. Oper. Res.* 92 (2018), 37–46.

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

[41] Uno Takeaki. 2003. *Two general methods to reduce delay and change of enumeration algorithms*. Technical Report. Technical Report.

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

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

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

[45] Zhengren Wang, Yi Zhou, Mingyu Xiao, and Bakhadyr Khoussainov. 2022. Listing Maximal k-Plexes in Large Real-World Graphs. In *WWW*.

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

[47] Kaiqiang Yu, Cheng Long, Shengxin Liu, and Da Yan. 2022. Efficient Algorithms for Maximal k-Biplex Enumeration. In *SIGMOD*. 860–873.

[48] Yun Zhang, Charles A. Phillips, Gary L. Rogers, Erich J. Baker, Elissa J. Chesler, and Michael A. Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinform.* 15 (2014), 110.

[49] Yi Zhou, Weibo Lin, Jin-Kao Hao, Mingyu Xiao, and Yan Jin. 2022. An effective branch-and-bound algorithm for the maximum s-bundle problem. *Eur. J. Oper. Res.* 297, 1 (2022), 27–39.

[50] Yi Zhou, Jingwei Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. 2020. Enumerating Maximal k-Plexes with Worst-Case Time Guarantee. In *AAAI*. 2442–2449.

[51] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. 2010. Finding top-k maximal cliques in an uncertain graph. In *ICDE*. 649–652.