Efficient Maximal Directed Plex Enumeration with Worst-Case Time Guarantee

Yukai Sun[†], Shuohao Gao[†], Kaiqiang Yu[‡], Shengxin Liu[†], Cheng Long[‡], Xun Zhou[†]

[†]*Harbin Institute of Technology, Shenzhen, China*

[‡]*Nanyang Technological University, Singapore*

{220810130@stu., 200111201@stu., sxliu@, zhouxun2023@}hit.edu.cn, {kaiqiang002@e., c.long@}ntu.edu.sg

***Abstract*—Directed graphs are essential for modeling real-world networks, where edge directionality encodes critical information. Cohesive subgraphs in directed graphs, such as directed (k, ℓ)-plexes (DPlexes), are widely used in applications including community detection, anomaly detection, and functional group discovery. A DPlex is defined as a subgraph in which each vertex disconnects at most k vertices and is meanwhile not pointed to by at most ℓ vertices. However, enumerating all maximal DPlexes is computationally challenging, as finding even one maximum DPlex is NP-hard. Furthermore, existing algorithms for undirected graphs cannot be directly applied to directed graphs due to the complexities introduced by edge directionality. To address these challenges, we propose an efficient branch-and-bound algorithm, `DPEnum`, which incorporates two key techniques: (1) a pivot technique based on a new concept, *out-degeneracy* (δ), which quantifies sparseness in directed graphs and satisfies $\delta < n$, where n is the number of vertices; and (2) a preprocessing technique that removes *reducible vertices* and directly identifies several maximal DPlexes. `DPEnum` achieves a worst-case time complexity of $O^*(\alpha_k^\delta)$, where O^* suppresses the polynomial factors and $\alpha_k < 2$, outperforming the naive $O^*(2^n)$. Additionally, we propose `LDPEnum` for enumerating large maximal DPlexes with at least s vertices, which uses the upper-bounding technique to prune search branches. Extensive experiments on real-world datasets show that our algorithms achieve up to four orders of magnitude speedup compared to the baseline.**

***Index Terms*—Cohesive subgraph mining, directed graphs, directed (k, ℓ)-plex**

I. Introduction

Directed graphs, a fundamental concept in graph theory, have significant applications in the field of network data analysis. For example, in certain social media networks, such as Twitter, TikTok, and Facebook, the "following" relationship between a user and a content creator can be abstracted as a directed edge pointing from the user to the content creator [30]. Similarly, in networks formed by the Internet (i.e., the World Wide Web), a web page can navigate to another through a "hyperlink", and such relationships can also be represented as directed edges in a directed graph [2]. These examples highlight how directed graphs provide a powerful framework for modeling and analyzing real-world network structures.

Within the domain of graph data mining, cohesive subgraphs are often regarded as carriers of rich information and are considered to hold significant research value [9], [21], [22], [29], [35], [52]. For instance, in community search tasks within social relationship networks, a cohesive subgraph represents a group of users who are closely connected, forming a highly dense community. However, real-world graphs are typically vast and sparse, making the extraction of cohesive subgraphs a challenging computational task. When analyzing such graph networks using cohesive subgraph models, directly applying existing models designed for undirected graphs may overlook *edge directions*, potentially leading to the loss of critical information. To address this limitation, it is necessary to adopt cohesive subgraph models specifically designed for directed graphs. Examples of such directed cohesive subgraph models include directed cliques [40], directed (k, ℓ)-plexes [25], directed (k, ℓ)-cores [23], [26], and directed quasi-cliques [27]. These models are natural extensions of their counterparts in undirected graphs, such as cliques [8], [44], k-plexes [3], [42], k-cores [4], [41], and quasi-cliques [33], [53]. Among these directed cohesive models, the directed (k, ℓ)-plexes (DPlexes) have been shown to outperform others in a case study on community search presented in [25].

In this paper, we focus on the maximal DPlexes enumeration (MDPE) problem in directed graphs. Specifically, a directed (k, ℓ)-plex is defined as a subgraph where each vertex disconnects at most k vertices and is not pointed to by at most ℓ vertices, where k and ℓ are two small positive integers. Furthermore, a DPlex g is considered maximal if there is no other DPlex that properly contains g. Finding all maximal DPlexes has a wide range of potential applications across various fields. In biological networks, maximal DPlexes can uncover biologically relevant functional groups, such as protein complexes or gene modules, providing insights into cellular functions and mechanisms [28]. In social network analysis, maximal DPlexes can support community detection by identifying tightly-knit groups with shared properties, offering insights into social dynamics, influence patterns, and collaborative structures [27]. Additionally, maximal DPlexes can be applied in anomaly detection, aiding in fraud detection and network security [48]. Furthermore, a case study on community search [25] demonstrates that DPlexes outperform other directed cohesive subgraph structures, such as directed core, truss, clique, and quasi-clique.

However, despite its practical significance, the enumeration of maximal DPlexes poses substantial theoretical challenges, as finding even one maximum DPlex – which is also a maximal DPlex in the graph – is NP-hard [3], [25]. Although

several algorithms exist for enumerating maximal k-plexes in undirected graphs [6], [13], [19], [46], [57], these algorithms cannot be easily adapted to our problem because they do not account for the directionality of edges. For instance, most k-plexes enumeration algorithms rely on the degeneracy ordering, which is defined by repeatedly removing a vertex of minimum degree in the remaining subgraph. However, defining a degeneracy ordering in directed graphs is not straightforward, as each vertex has both in- and out-degrees.

In this paper, we first introduce `BaselineBK`, a baseline algorithm based on the Bron-Kerbosch framework, which has been widely adopted as the foundation for numerous enumeration algorithms [7], [18], [19], [46], [53], [57]. However, `BaselineBK` is not practically efficient due to the large number of search branches and the vast size of the search space. To address these limitations, we propose an efficient branch-and-bound algorithm, `DPEnum`, which incorporates novel pivot and preprocessing techniques to accelerate enumeration. Specifically, our pivot technique is based on a new concept, *out-degeneracy* (denoted by δ), which measures sparseness in directed graphs, extends the concept of degeneracy from undirected graphs, and is formally proven to satisfy $\delta < n$, where n is the number of vertices in the graph. With the pivot technique, `DPEnum` achieves the worst-case time complexity of $O^*(\alpha_k^\delta)$, where O^* suppresses the polynomial factors and $\alpha_k < 2$, thereby improving upon the trivial time complexity of $O^*(2^n)$ in terms of *both the base and the exponent*. Additionally, we propose a preprocessing technique in `DPEnum`, which is designed to (1) preemptively remove vertices with relatively few neighbors, referred to as *reducible vertices*, and (2) directly identify several maximal DPlexes that contain these vertices. This step reduces redundant computations and accelerates the recursive enumeration process.

We also introduce the problem of enumerating all *large* maximal DPlexes with at least s vertices in a given graph. To address this problem, we propose an efficient algorithm, `LDPEnum`, which incorporates the upper bounding technique. This technique utilizes multiple methods to estimate the upper bound in each branch of the enumeration and compares it with the size restriction s to prune branches that cannot produce valid DPlexes. Our main contributions are as follows:

- We introduce the *out-degeneracy* (denoted by δ), which measures the sparseness of directed graphs. (Section II)
- We present a Bron-Kerbosch based method `BaselineBK` as our baseline. (Section III)
- We propose a new branch-and-bound algorithm `DPEnum` for the maximal DPlexes enumeration problem. `DPEnum` leverages the pivot technique, resulting in a time complexity of $O^*(\alpha_k^\delta)$, where $\alpha_k < 2$. Moreover, `DPEnum` incorporates a novel preprocessing technique that directly identifies several maximal DPlexes prior to enumeration. (Sections IV and V)
- We also study the problem of enumerating large maximal DPlexes and propose an efficient algorithm `LDPEnum`, which includes the upper bounding technique. (Section VI)
- We conduct extensive experiments on real database to verify the efficiency of both `DPEnum` and `LDPEnum`. Compared to the baseline `BaselineBK`, `DPEnum` is up to three orders of magnitudes faster, while `LDPEnum` achieves speeds up to four orders of magnitudes faster. (Section VII)

II. Preliminaries

Let $G = (V, E)$ be a *directed* graph (without parallel edges and self-loops), where the vertex set V has $|V| = n$ vertices and the edge set E has $|E| = m$ edges. For a directed edge $(u, v) \in E$, the direction of the edge is from u to v, i.e., $u \to v$; we say that u is the *in-neighbor* of v and v is the *out-neighbor* of u. For a vertex v, the sets of in-neighbors and out-neighbors in G are denoted by $N_G^{in}(v)$ and $N_G^{out}(v)$, respectively. Subsequently, the in-degree and out-degree of vertex v in G are defined as $d_G^{in}(v) = |N_G^{in}(v)|$ and $d_G^{out}(v) = |N_G^{out}(v)|$, respectively. Following [25], we utilize the concept of *pseudo-degree* $pd_G(v)$ for vertex v in G, i.e., $pd_G(v) = \min(d_G^{out}(v) + k, d_G^{in}(v) + \ell)$. Given a vertex set S, we use $G[S]$ to describe the induced subgraph, where the induced vertex set is S and the edge set is $\{(u, v) \mid (u, v) \in E \text{ and } u, v \in S\}$. Moreover, for a subgraph g, we use $V(g)$ and $E(g)$ to denote its vertex set and edge set, respectively.

In this paper, we focus on the following cohesive subgraph.

Definition 1 (Directed (k, ℓ)-Plex [25])**.** *A subgraph g is said to be a* directed (k, ℓ)-plex (DPlex) *if each vertex $v \in V(g)$ disconnects (including itself) at most k out-neighbors and at most ℓ in-neighbors in g, i.e., $d_g^{out}(v) \geq |V(g)| - k$ and $d_g^{in}(v) \geq |V(g)| - \ell$. In other words, $pd_g(v) \geq |V(g)|$ for each vertex $v \in V(g)$.*

Moreover, a DPlex g in G is said to be *maximal* if there does not exist another DPlex g' in G such that $V(g) \subset V(g')$.

We are ready to define the problem we study in this paper.

Problem 1. *Given a directed graph $G = (V, E)$ and two positive integers k and ℓ, the* Maximal Directed (k, ℓ)-Plex Enumeration Problem (MDPE) *aims to find all maximal directed (k, ℓ)-plex in G.*

Furthermore, we also consider the problem variant of listing large maximal directed (k, ℓ)-Plex.

Problem 2. *Given a directed graph $G = (V, E)$ and three positive integers k, ℓ and s, the* Large Maximal Directed (k, ℓ)-Plex Enumeration Problem (LMDPE) *aims to find all maximal directed (k, ℓ)-plex of size at least $s \geq k + \ell - 1$ in G.*

A. Useful Properties of DPlex

We first show two useful properties of DPlex, which has been proved in [25]. The first one is the hereditary property.

Property 1 ([25])**.** *If g is a directed (k, ℓ)-plex, any subgraphs $g' \subseteq g$ is a directed (k, ℓ)-plex.*

The next property shows that when a DPlex is sufficiently large, it is strongly connected and its diameter is at most 2.

Property 2 ([25])**.** *If g is a directed (k, ℓ)-plex with $|V(g)| > k + \ell - 2$, then the diameter of g does not exceed 2, i.e., $\forall u, v \in V(g)$, the distance from u to v is at most 2.*

Besides, in this paper, we focus on the case with $k \leq \ell$. This is because the case with $k \geq \ell$ can be converted to the case with $k \leq \ell$ via the following property.

Property 3. *Given a directed graph $G = (V, E)$ and two integers k and ℓ, if $G[S]$ is a directed (k, ℓ)-plex, then $G'[S]$ is a directed (ℓ, k)-plex in the transpose graph $G' = (V, E')$ of G, where a directed edge $(v, u) \in E'$ if and only if $(u, v) \in E$.*

The proof of the above property is obvious due to the definition of the transpose graph. Moreover, Property 3 shows a bijection relation between a directed (k, ℓ)-plex in G and a directed (ℓ, k)-plex in the transpose graph G'.

B. A New Measurement in Directed Graphs: Out-Degeneracy

In undirected graphs, a useful notation that measures the sparseness is *degeneracy*. In particular, the degeneracy of an undirected graph is δ, meaning that δ is the smallest number such that every induced subgraph of this graph has a vertex of degree at most δ [4], [8]. However, it is not straightforward to define the degeneracy in directed graphs since each vertex has both in-degree and out-degree. In this paper, we instead introduce a new measurement, called *out-degeneracy*, of the sparseness in directed graphs by using only out-degree.

Definition 2 (Out-Degeneracy)**.** *Given a directed graph G, the* out-degeneracy *$\delta^{out}(G)$ of G (hereafter abbreviated as δ) is an integer that satisfies the following two conditions:*

- *There exists a subgraph g_1 of G such that $\forall v \in V(g_1)$, $d^{out}_{g_1}(v) \geq \delta$;*
- *There does not exist a subgraph g_2 of G such that $\forall v \in V(g_2)$, $d^{out}_{g_2}(v) \geq \delta + 1$.*

Remark that one can similarly define the *in-degeneracy* by using in-degree. Since we focus on the case of $k \leq \ell$ in this paper, the constraint on out-edges is stricter than on in-edges. Thus, we will use the out-degeneracy for designing algorithms and analyzing time complexity. Note also that the out-degeneracy is a measurement of a given directed graph, independent of the parameters k and ℓ of DPlex. The properties related to the out-degeneracy are summarized as follows.

Property 4. $\delta < \sqrt{m}$.

The proof is provided in Appendix of [1]. We remark that, in large real graphs, δ is usually much smaller than n, which can be verified in Table I in Section VII. Based on this property, we will design algorithms that achieve better theoretical time complexity by optimizing the exponential term from n to δ.

The size of the largest directed (k, ℓ)-plex is also related to the out-degeneracy of the graph.

Property 5. *The size of the largest directed (k, ℓ)-plex is at most $\delta + k$.*

The proof is provided in Appendix of [1]. Moreover, it is easy to compute the out-degeneracy of a graph by iteratively removing the vertex with the smallest out-degree from the current graph. This algorithm can be completed in linear time of $O(n+m)$, where we can use n lists with the i-th list storing all vertices with an out-degree of exactly i ($0 \leq i \leq n-1$). Another related concept is the *out-degeneracy ordering*, which arranges the vertex set V as $v_1, v_2, \ldots, v_n$ such that v_i has the smallest out-degree in the subgraph $G[v_i, v_{i+1}, \ldots, v_n]$. The out-degeneracy ordering is not unique. It is easy to see that the sequence of vertex deletion $v_1, v_2, \ldots, v_n$ for computing out-degeneracy exactly matches the definition of out-degeneracy ordering, and we have $|N^{out}_G(v_i) \cap \{v_i, v_{i+1}, \ldots, v_n\}| \leq \delta$.

Algorithm 1: `BaselineBK`

Input: A directed graph G and two integers k, ℓ
Output: All directed (k, ℓ)-plexes in G

1 `BK-Rec`$(G, \emptyset, V(G), \emptyset, k, \ell)$;

Procedure: `BK-Rec`(G, S, C, X, k, ℓ)
2 $C \leftarrow \{u \in C \mid G[S \cup \{u\}]$ is a DPlex$\}$;
3 $X \leftarrow \{u \in X \mid G[S \cup \{u\}]$ is a DPlex$\}$;
4 **if** $C = \emptyset$ **then**
5 **if** $X = \emptyset$ **then**
6 Output $G[S]$;
7 **return**;
8 $u \leftarrow$ select a vertex from C;
9 `BK-Rec`$(G, S \cup \{u\}, C \setminus \{u\}, X, k, \ell)$;
10 `BK-Rec`$(G, S, C \setminus \{u\}, X \cup \{u\}, k, \ell)$;

III. The Bron-Kerbosch based Baseline Algorithm

The Bron-Kerbosch algorithm, which is designed for the maximal clique enumeration problem [7], is utilized in many existing enumeration algorithms [18], [19], [46], [53], [57]. Following the same fashion, we introduce the Bron-Kerbosch based algorithm `BaselineBK` as our baseline in this section.

Our baseline `BaselineBK` is summarized in Algorithm 1. Specifically, `BaselineBK` invokes the recursive process `BK-Rec` in Line 1. The recursive process of `BK-Rec` (Lines 2-10) is equipped with three disjoint vertex sets: 1) the selected vertex set S, which guarantees that $G[S]$ is a DPlex; 2) the candidate vertex set C, where the vertices in C could be selected into S to form a larger DPlex; 3) the excluded vertex set X, where the vertices are used to avoid non-maximal DPlexes although they can also be selected into S to form a larger DPlex. The rationale of `BK-Rec` is that a triplet (S, C, X) corresponds to a branch (or a sub-problem). This branch finds all maximal DPlexes that include the selected vertex set S in $G[S \cup C]$; the excluded vertex set X can be used to guarantee the found DPlexes are also maximal in G. In each branch, we select a vertex $u \in C$ in Line 8 and generate two sub-branches: one includes vertex u (corresponding to sub-branch $(G, S \cup \{u\}, C \setminus \{u\}, X)$) in Line 9; another excludes vertex u (corresponding to subbranch $(G, S, C \setminus \{u\}, X \cup \{u\})$) in Line 10. For each branch of `BK-Rec`, we first remove redundant vertices using degree-based pruning techniques; specifically, vertices in C and X with $pd_{g[S \cup \{v\}]}(v) < |S| + 1$ are removed, based on the definition and hereditary property of DPlex (Lines 2-3). When the candidate vertex set C is

empty, the algorithm outputs a maximal DPlex in Line 6 if the excluded vertex set X is empty in Line 5; then, the recursion will stop as the selected vertex set S cannot be further expanded (Lines 4-7). Clearly, by invoking BK-Rec with $(S = \emptyset, C = V(G), X = \emptyset)$ in Line 1, BaselineBK correctly outputs all maximal DPlexes in G.

Remarks. **First**, it is easy to see that the time complexity of BaselineBK is $O^*(2^n)$ since BK-Rec recursively visits all the vertex subsets in the worst case. **Second**, BaselineBK can be easily adapted to solve the problem of large maximal DPlex enumeration (LMDPE) by including a size check of "**if** $|S| + |C| \geq s$" before Line 4 of Algorithm 1.

Limitations of the Baseline. Despite BaselineBK can correctly return all the DPlexes for a given directed graph, it may not be efficient due to the following reasons.

- **Numerous search branches.** BaselineBK makes use of a basic binary branching strategy to generate branches. Thus, it is necessary to consider more advanced branching strategies that could lead to better theoretical time complexity and practical performance. Moreover, to reduce the number of search branches, BaselineBK uses the hereditary property of DPlex to shrink the candidate vertex set in each recursion and the size constraint to avoid unnecessary computations (for the LMDPE problem). However, these methods are still limited and do not fully address the efficiency issue. Thus, it is essential to develop effective techniques (e.g., upper bounding method) to further boost performance.
- **Huge search space.** The time complexity of BaselineBK depends on the size of the graph being searched. The search space of BaselineBK is the entire input graph, which includes several special vertices (e.g., those with limited degree) that can be used to directly return the maximal DPlex. Thus, to avoid redundant computations in the exhaustive search, it is crucial to design effective preprocessing techniques to accelerate the enumeration process.

IV. Our Enumeration Algorithm: DPEnum

In this section, we present our algorithm DPEnum for enumerating all maximal DPlexes in Section IV.A. Moreover, we show the time complexity of DPEnum is $O^*(\alpha_k^\delta)$, where α_k is strictly smaller than 2 and δ is strictly smaller than n, in Section IV.B. We remark that this new time complexity breaks the trivial bound of $O^*(2^n)$ in terms of *both the base and the exponent*. Below, we elaborate on the details.

A. Algorithm Description

Our algorithm DPEnum is shown in Algorithm 2. In particular, DPEnum has three technical components.

Preprocessing. In Line 1 of Algorithm 2, we first invoke a preprocessing method DPEnum-Pre for directly removing some specific vertices from the graph and outputting several maximal Dlexes related to these vertices before the enumeration process. Note that the more removed vertices the smaller the graph used in the enumeration process; the more outputted maximal DPlexes the less returned in the enumeration process. Thus, DPEnum aims to *quickly* identify the set of specific vertices and output the related DPlexes. The details of DPEnum-Pre will be discussed in Section V.

Algorithm 2: DPEnum

Input: A directed graph G, and two integers k, ℓ
Output: All directed (k, ℓ)-plexes in G

1 $G \leftarrow$DPEnum-Pre(G, k, ℓ);
2 Sort V as $v_1, v_2, ..., v_n$ according to the out-degeneracy ordering;
3 **for** $i = 1$ *to* n **do**
4 $\quad g \leftarrow G[\{v_i, v_{i+1}, ..., v_n\}]$;
5 $\quad$ **foreach** $P \subseteq V(g) \setminus (\{v_i\} \cup N_g^{out}(v_i))$ *satisfying* $|P| \leq k - 1$ **do**
6 $\quad\quad S \leftarrow P \cup \{v_i\}$;
7 $\quad\quad C \leftarrow \{u \in N_g^{out}(v_i) \mid G[S \cup \{u\}]$ is a DPlex$\}$;
8 $\quad\quad X \leftarrow \{u \in V(G) \setminus (S \cup C) \mid G[S \cup \{u\}]$ is a DPlex$\}$;
9 $\quad\quad$ DPEnum-Pivot(g, S, C, X, k, ℓ);

Divide-and-Conquer. In Lines 2-4 of Algorithm 2, we make use of the classic *divide-and-conquer* technique, which has been widely used for finding cohesive subgraphs [10], [11], [53], [54]. The goal of the divide-and-conquer technique is to divide the input graph into several smaller ones and run an enumeration algorithm on each of them. Specifically, given an ordering of vertices in V, i.e., $\langle v_1, v_2, ..., v_n \rangle$, we can divide the problem into n sub-problems. The i-th $(1 \leq i \leq n)$ sub-problem corresponds to finding all DPlexes that includes v_i and excludes $\{v_1, v_2, ..., v_{i-1}\}$ from G. It is easy to see that solving n sub-problems solves the original problem since every DPlex in G must be found in one of the sub-problems. We remark that we adopt the *out-degeneracy ordering*, which is discussed in Section II.B, in Line 2 of Algorithm 2. This is because (1) the out-degeneracy ordering can be computed in linear time $O(n + m)$ by iteratively removing the vertex with smallest out-degree in the current graph; and (2) with the out-degeneracy ordering, each selected vertex v_i in the i-th sub-problem has at most δ out-neighbors in $g = G[\{v_i, v_{i+1}, ..., v_n\}]$, i.e., $d_g^{out}(v_i) \leq \delta$.

Decomposed Enumeration with Pivot. By applying the divide-and-conquer technique, the i-th sub-problem requires to find all DPlexes that includes v_i and excludes $\{v_1, v_2, ..., v_{i-1}\}$ from G. In this step, we aim to decompose the i-th sub-problem into several tasks, each of which can be solved by a pivot-based Bron-Kerbosch algorithm. Specifically, for the i-th sub-problem, we focus on the subgraph $g = G[\{v_i, v_{i+1}, ..., v_n\}]$ constructed in Line 4 of Algorithm 2. We first observe that any DPlex containing vertex v_i contains at most $k - 1$ vertices that are not out-neighbors of v_i (since otherwise the requirement of DPlex would be violated). Thus, in Line 5, we enumerate the vertex set P with at most $k - 1$ vertices from $V(g) \setminus (\{v_i\} \cup N_g^{out}(v_i))$. Based on P, we then decompose the i-th sub-problem into several

Algorithm 3: DPEnum-Pivot(G, S, C, X, k, ℓ)

Output: All directed (k, ℓ)-plexes in $G[S \cup C]$

1 $C \leftarrow \{u \in C \mid G[S \cup \{u\}]$ is a DPlex$\}$;
2 $X \leftarrow \{u \in X \mid G[S \cup \{u\}]$ is a DPlex$\}$;
3 $M \leftarrow \arg\min_{v \in S \cup C} pd_{G[S \cup C]}(v)$;
4 $u \leftarrow \arg\min_{v \in M} pd_{G[S \cup \{v\}]}(v)$;
5 **if** $pd_{G[S \cup C]}(u) \geq |S \cup C|$ **then**
6 **if** $G[S \cup C \cup \{w\}]$ *is not a DPlex* $\forall w \in X$ **then**
7 Output $G[S \cup C]$;
8 **return**;
9 **if** $u \in S$ **then**
10 **if** $pd_{G[S]}(u) = d^{out}_{G[S]}(u) + k$ **then**
11 $u_{new} \leftarrow \arg\min_{v \in C \setminus N^{out}_G(u)} pd_{G[S \cup \{v\}]}(v)$;
12 **else**
13 $u_{new} \leftarrow \arg\min_{v \in C \setminus N^{in}_G(u)} pd_{G[S \cup \{v\}]}(v)$;
14 $u \leftarrow u_{new}$;
15 DPEnum-Pivot$(G, S \cup \{u\}, C \setminus \{u\}, X, k, \ell)$;
16 DPEnum-Pivot$(G, S, C \setminus \{u\}, X \cup \{u\}, k, \ell)$;

tasks related to P, where each task corresponds to an instance of the Bron-Kerbosch algorithm that aims to find all maximal DPlexes that must include $P \cup \{v_i\}$ and exclude $V(g) \setminus (\{v_i\} \cup N^{out}_g(v_i) \cup P)$. Specifically, we initialize the three disjoint vertex sets S, C, and X in Lines 6-8, where the selected vertex set $S = P \cup \{v_i\}$, the candidate vertex set $C = \{u \in N^{out}_g(v_i) \mid G[S \cup \{u\}]$ is a DPlex$\}$, and the excluded vertex set $X = \{u \in V(G) \setminus (S \cup C) \mid G[S \cup \{u\}]$ is a DPlex$\}$. Then, we recursively invoke DPEnum-Pivot with specified S, C, and X in Line 9 of Algorithm 2. The correctness of the above decomposition is easy to verify.

The pseudo-code of DPEnum-Pivot is shown in Algorithm 3. The main idea of DPEnum-Pivot follows the Bron-Kerbosch based baseline BaselineBK and the core of DPEnum-Pivot is to select a *pivot* vertex to be considered in the recursion of Lines 15-16 of Algorithm 3 (note that one recursion includes vertex u from C into S; the other recursion includes vertex u from C into X). Specifically, the selection criteria of the pivot vertex in Lines 3-4 are: (1) the vertex u has the smallest pseudo-degree in $G[S \cup C]$, i.e., $u \in M$ (Line 3); and (2) For all vertices in M, we select the pivot vertex u that is the "least adjacent to" S, i.e., u has the smallest value of $pd_{G[S \cup \{u\}]}(u)$ for all vertices in M (Line 4). With this selected pivot vertex u, we can easily derive that if $pd_{G[S \cup C]}(u) \geq |S \cup C|$, $G[S \cup C]$ is already a DPlex. We then check whether $G[S \cup C]$ is maximal (Line 6) by checking whether $G[S \cup C \cup \{w\}]$ is a DPlex for all vertices $w \in X$; and output $G[S \cup C]$ is a maximal DPlex (Line 7). Note that we can stop this recursion in Line 8 since $G[S \cup C]$ is already a DPlex. We next consider the case where the pivot vertex u (selected in Lines 3-4) is in S; in such a case, we need to re-select the pivot vertex for our recursion. Specifically, in Lines 10-11, if we have $pd_{G[S]}(u) = d^{out}_{G[S]}(u) + k$, i.e., $d^{out}_{G[S]}(u) + k \leq d^{in}_{G[S]}(u) + \ell$, the re-selected vertex u_{new} has smallest value of $pd_{G[S \cup \{u_{new}\}]}(u_{new})$ among all the vertices that are not out-neighbors of u in C; otherwise, in Lines 12-13, the re-selected vertex u_{new} has smallest value of $pd_{G[S \cup \{u_{new}\}]}(u_{new})$ among all the vertices that are not in-neighbors of u in C. The reason for this re-selection criterion will become clear in our time complexity analysis below.

B. Time Complexity Analysis

We show the time complexity of our proposed algorithm DPEnum (Algorithm 2) in the following.

Theorem 1. *Given a directed graph G and two positive integers k and ℓ with $k \leq \ell$, Algorithm 2 finds all maximal DPlexes in $O(n^{k+2} + mn^k \alpha^{\delta}_k)$, where $\alpha_k < 2$ is the largest real root of the equation $x^{k+2} - 2x^{k+1} + 1 = 0$. For example, when $k = 1$, 2, and 3, $\alpha_k = 1.6181$, 1.8393, and 1.9276, respectively.*

Proof. First, we can observe that Lines 1-2 of DPEnum (Algorithm 2) need $O(m + n \log n + n^{k+2}) = O(n^{k+2})$ time as $k \geq 1$ (the time complexity of DPEnum-Pre is $O(n^{k+2})$ and will be analyzed in the following section). Then, we know that DPEnum invokes at most $O(n^k)$ times of DPEnum-Pivot in Line 9 (as there are n iterations in Line 3 and each iteration has at most n^{k-1} calls to DPEnum-Pivot in Lines 5-9). Besides, each call to DPEnum-Pivot requires additional $O(m)$ time since we need to construct g in Line 4, compute P in Line 5, and initialize S, C, and X in Lines 6-8.

We next focus on the time complexity of DPEnum-Pivot (Algorithm 3). We can see that Lines 1-14 of Algorithm 3 can be solved in $O(m)$ time, since (1) Lines 1, 2, and 6 can be done in $O(m)$; and (2) computing and operating the arrays of $pd(\cdot)$ in Lines 3, 4, 11, and 13 can be done in $O(m)$. We then consider the maximum number of recursions (Lines 15-16) of DPEnum-Pivot invoked in Line 9 of Algorithm 2. Let $L(z)$ be the maximum number of recursions that can be generated in DPEnum-Pivot(g, S, C, X, k, ℓ) where $z = |C|$. We note that in each recursion (either Line 15 or 16), the size of the candidate vertex set z is decreased by 1. Thus, we have

$$L(z) \leq 2 \times L(z - 1). \tag{1}$$

Clearly, directly solving the recurrence in Equation (5) gives the maximum number of recursions of $O(2^n)$. To obtain a tighter analysis, we pay attention to the *pivot* vertex u selected in Lines 3-4 of Algorithm 3. Based on the pivot vertex u, we have the following two scenarios.

Scenario 1: $u \in S$**.** In this case, Algorithm 3 re-selects a new pivot vertex u_{new} based on u in Lines 9-14. An important observation is that in the sub-recursion of DPEnum-Pivot$(G, S \cup \{u_{new}\}, C \setminus \{u_{new}\}, X, k, \ell)$ (Line 15), the pivot vertex selected in Lines 3-4 of the sub-recursion is identical to the pivot selected in Lines 3-4 of the recursion of DPEnum-Pivot(G, S, C, X, k, ℓ). This is because both values of $pd_{G[S \cup C]}(u)$ and $pd_{G[S \cup \{u\}]}(u)$ do not increase when we include u_{new} into S in Line 15 of the recursion of DPEnum-Pivot(G, S, C, X, k, ℓ). u_{new} will be selected as

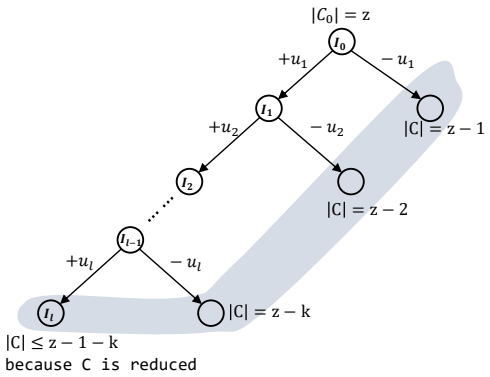


Fig. 1: An example of (sub-)recursions

the pivot vertex again in the subsequent sub-recursions until (1) the recursion is terminated (i.e., $|C| = 0$) or (2) the candidate set C is shrinked by at least 1 in Line 1 of Algorithm 3, i.e., there exists at least one vertex in C that cannot form a DPlex with S. Note that u_{new} is selected based on the vertex u. Moreover, as shown in Lines 11 and 13, u_{new} is the non-out-neighbor or non-in-neighbor of u. Thus, u_{new} can be selected at most a number of $pd_{G[S]}(u) - |S|$ times based on the same vertex u, since otherwise C is shrinked by at least 1 in Line 1. Moreover, we have $pd_{G[S]}(u) - |S| = \min\{d^{out}_{G[S]}(u) + k - |S|, d^{in}_{G[S]}(u) + \ell - |S|\} \leq \min\{k-1, \ell-1\} = k-1$, u_{new} can be selected at most $k-1$ times. See Figure 1 for an example. Thus, we conclude that

$$L(z) \leq L(z-k) + \sum_{i=1}^{k-1} L(z-i). \tag{2}$$

We solve the recurrence in Equation (6) to derive $L(z) = O(\gamma_k^z)$, where γ_k is the largest real root of the equation $x^{k+1} - 2x^k + 1 = 0$, by the analytical methods in Section 2.1 of [34].

Scenario 2: $u \in C$. In this case, the pivot vertex u of `DPEnum-Pivot`(G, S, C, X, k, ℓ) is pushed into set S to invoke the sub-recursion `DPEnum-Pivot`$(G, S \cup \{u\}, C \setminus \{u\}, X, k, \ell)$. We remark that both values of $pd_{G[S\cup C]}(u)$ and $pd_{G[S\cup\{u\}]}(u)$ do not increase when we include u into S in Line 15 of the recursion of `DPEnum-Pivot`(G, S, C, X, k, ℓ). This implies that the pivot vertex selected in Lines 3-4 of the sub-recursion of `DPEnum-Pivot`$(G, S \cup \{u\}, C \setminus \{u_{new}\}, X, k, \ell)$ is identical to the pivot selected in Lines 3-4 of the recursion of `DPEnum-Pivot`(G, S, C, X, k, ℓ). Then, we have the case where the pivot vertex u is in set S, and we have the identical situation in Scenario 1. Following the discussion in Scenario 1, we can conclude that

$$L(z) \leq L(z-1) + L(z-k-1) + \sum_{i=1}^{k-1} L(z-i-1). \tag{3}$$

We solve the recurrence (Equation (3)) to derive $L(z) = O(\alpha_k^z)$, where α_k is the largest real root of $x^{k+2} - 2x^{k+1} + 1 = 0$, by the analytical methods in Section 2.1 of [34].

Combing the above two scenarios, the maximum number of recursions is the largest one between $O(\gamma_k^z)$ and $O(\alpha_k^z)$, which corresponds to $O(\alpha_k^z)$. We then focus on bounding the value of z, i.e., the size of the candidate vertex set. We note that in Algorithm 2, we utilize the technique of divide-and-conquer in Lines 2-4. Since we use the out-degeneracy ordering, we can easily see that $z = |C| \leq \delta$ (as $C \subseteq N^{out}_g(v_i)$ for each v_i). The proof is thus complete. □

V. Our Preprocessing Techniques

In this section, we present our proposed preprocessing techniques for the MDPE problem. Specifically, the objective of preprocessing is to *quickly* identify and reduce some specific vertices from the graph and output a set of (but not necessarily all) maximal DPlexes related to these vertices before the enumeration, in order to reduce redundant computations in the enumeration and thus accelerate the algorithm.

Our preprocessing techniques rely on a type of *reducible* vertices, which is defined as follows.

Definition 3. *Given a directed graph $G = (V, E)$ and two integers k and ℓ with $k \leq \ell$, a vertex $v \in V$ is said to be* reducible *if v does not belong to any directed (k, ℓ)-plex of size at least $k + 1$.*

From Definition 3, we know that a reducible vertex can only be in a directed (k, ℓ)-plex of size at most k, which corresponds to a trivial directed (k, ℓ)-plex, since any subgraph of at most k vertices is a directed (k, ℓ)-plex. Based on the reducible vertex, we have the following lemma.

Lemma 1. *Given a directed graph $G = (V, E)$, two integers k and ℓ with $k \leq \ell$, and a reducible vertex v, all maximal DPlexes that contain v can be obtained by enumerating all the vertex sets $S \subseteq V \setminus \{v\}$ that satisfy $|S| = k - 1$ and outputting $G[S \cup \{v\}]$. Moreover, v can be safely reduced without affecting the correctness of other maximal DPlexes.*

Proof. It is easy to see that each outputted subgraph is a maximal DPlex by the definitions of DPlex (Definition 1) and reducible vertices (Definition 3). □

By Lemma 1, we can identify a set of reducible vertices. Then, (1) we can remove reducible vertices from the graph and shrink the size of graph with polynomial time; (2) and we only need to enumerate subsets of V without checking whether a found DPlex is maximal. These processes can reduce redundant calculations so as to accelerate the whole enumeration algorithm. In the following, we show how to identify some reducible vertices through the following lemmas.

Lemma 2. *Given a directed graph G and two integers k and ℓ with $k \leq \ell$, if vertex $v \in V$ satisfies the following:*

- $d^{out}_G(v) = 0$, *or*
- $d^{in}_G(v) = 0$ *and* $k = \ell$,

then v is a reducible vertex.

Proof. It is easy to see that a vertex v satisfying the above conditions cannot exist in any DPlex with at least $k+1$ vertices due to the definition of DPlex (Definition 1). □

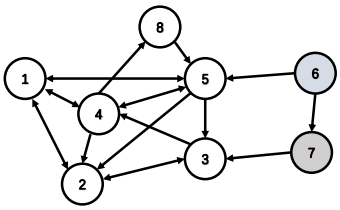


Fig. 2: An example of reducible vertices ((k,ℓ)=(2,2))

We can generalize the above lemma to the following lemma.

Lemma 3. *Given a directed graph G and two integers k and ℓ with $k \leq \ell$, if vertex $v \in V$ satisfies the following:*

- *any directed walk of length exactly $k+1$* starting from v *does not contain the same vertex twice (does not contain cycles), i.e, the walk $(v_1, v_2, ..., v_{k+1})$ where $v_1 = v$ and $v_i \neq v_j$ for all $i \neq j$, or*
- *when $k = \ell$, any directed walk of length exactly $k+1$* ending at v *does not contain the same vertex twice (does not contain cycles), i.e, the walk $(v_1, v_2, ..., v_{k+1})$ where $v_{k+1} = v$ and $v_i \neq v_j$ for all $i \neq j$,*

then v is a reducible vertex.

The proof can be found in Appendix of [1]. Moreover, we can identify more reducible vertices as stated in the following.

Lemma 4. *Given a directed graph G and two integers k and ℓ with $k = \ell \leq 3$, if vertex $v \in V$ satisfies the following:*

- *$v \notin N^{out}_{\leq k+1}(v)$, where $N^{out}_{\leq k+1}(v)$ denotes the set of vertices that are reached by a directed walk of length at most $k+1$ starting from v,*

then v is a reducible vertex.

The proof can be found in Appendix of [1].

An Example of Reducible Vertices. We use an example graph of 7 vertices with $(k, \ell) = (2, 2)$ to show the reducible vertices in Figure 2. First, as vertex 6 has no in-neighbor, which satisfies Lemma 2, vertex 6 is clearly reducible. Next, we can check that vertex 7 is contained in any cycle of length at most 3, which implies that vertex 7 is reducible base on Lemma 4. Moreover, vertex 3 is part of a cycle $(3 \rightarrow 4 \rightarrow 5 \rightarrow 3)$ of length 3, thus we cannot determine whether vertex 3 is reducible or not by applying Lemmas 2, 3, and 4.

Remarks. **First**, Lemma 2 is utilized for handling those vertices without out-neighbors or in-neighbors, while Lemma 3 is a generalization of Lemma 2. For specific values of k and ℓ, we can use Lemma 4 for identifying more reducible vertices. **Second**, Lemmas 2, 3, and 4 are not applicable for undirected graphs, even though we can transform an undirected graph into a directed graph by using bidirectional edges to replace edges in the undirected graph. This is because Lemmas 2, 3, and 4 are derived based on the properties of directed graphs.

Our Preprocessing Algorithm: `DPEnum-Pre`. We summarize our preprocessing algorithm `DPEnum-Pre` in Algorithm 4. In Line 1, we initialize a queue Q to store the reducible vertices to be considered. Then, Lines 2-4 iteratively

Algorithm 4: `DPEnum-Pre`

Input: A directed graph G and two integers k, ℓ
Output: Several maximal DPlexes and a reduced graph G

1 Initialize a queue $Q \leftarrow \emptyset$;
2 **foreach** $v \in V$ **do**
3 **if** v *satisfies Lemma 2, 3, or 4* **then**
4 $Q \leftarrow Q \cup \{v\}$;
5 **while** $Q \neq \emptyset$ *and* $|V| > k$ **do**
6 $u \leftarrow$ pop a vertex from Q;
7 **foreach** $S \subset V \setminus \{u\}$ *satisfying* $|S| = k-1$ **do**
8 Output $G[S \cup \{u\}]$;
9 Remove u from G;
10 **foreach** $v \in V$ *and* $v \notin Q$ **do**
11 Update $d^{out}_G(v)$ and $d^{in}_G(v)$;
12 **if** v *satisfies Lemma 2, 3, or 4* **then**
13 $Q \leftarrow Q \cup \{v\}$;
14 **if** $|V| = k$ **then** Output G;
15 **return** G;

check whether a vertex $v \in V$ satisfies Lemmas 2, 3, and 4 and push reducible vertices into Q. We next iteratively process reducible vertices and output related maximal DPlexes in Lines 5-13 if Q is not empty and $|V| > k$. Specifically, given a vertex u popped from Q (Line 6), we output maximal DPlexes that contain u (Lines 7-8) and remove u from the current graph G (Line 9) according to Lemma 1. We then update the degrees of the remaining vertices in V and check whether there exist reducible vertices by Lemmas 2, 3, and 4 (Lines 10-13). In Line 14, we output the current graph as a maximal DPlex if we have $|V| = k$. Finally, we return the reduced graph (which removes those reducible vertices and their edges) in Line 15.

Time Complexity. The correctness of Algorithm 4 is easy to verify. We next focus on the time complexity. In Lines 2-4, we check each vertex in V using Lemmas 2, 3, and 4. Specifically, for Lemma 2, we simply check whether the in-degree and out-degree of each vertex is equal to 0; thus, Lemma 2 can be done in $O(n)$ for all vertices in V. In Lemmas 3 and 4, we need to check all walks (either starting from v or ending at v) of length exactly $k+1$, which requires $O(n^k)$ time. Thus, the total time complexity in Lines 2-4 is $O(n \cdot (n + n^k)) = O(n^{k+1})$. Then, in Lines 5-13, each vertex will be pushed into Q at most once; thus, the while loop is performed at most n times. For each while loop, Lines 7-8 use $O(n^{k-1})$ time to output all the solutions containing u; Lines 10-13 are the same as Lines 2-4, which cost $O(n^{k+1})$. Thus, the total time complexity of Algorithm 4 is $O(n^{k+1} + n^{k+2}) = O(n^{k+2})$.

VI. Large Maximal DPlexes Enumeration

To enumerate large maximal DPlexes, i.e., maximal DPlexes of size at least s ($s \geq k+\ell-1$), our proposed `DPEnum` in Section IV can be adapted to solve the LMDPE problem by simply

filtering out those maximal DPlexes with size smaller than s. However, this straightforward modified algorithm is inefficient in practice (as verified in our experiments in Section VII). In fact, we can make use of the size constraint of s to design novel upper bounding techniques (Section VI.A) such that the number of search branches is reduced. With the newly developed upper-bound-based pruning techniques and other speed-up techniques, our final large maximal DPlexes enumeration algorithm `LDPEnum` is summarized in Section VI.B.

A. Upper Bounding Techniques

We discuss upper bounding techniques in this section. In particular, we aim to design several methods used for estimating the upper bound ub of the maximum DPlex containing the selected vertex set S during the enumeration. It is clear that once the estimated ub is no larger than the size constraint of s, i.e., $ub < s$, we can safely prune the current branch that associates with the selected vertex set S. Thus, a tighter upper bound can potentially prune more branches.

Our upper bound is shown as follows.

Lemma 5. *Given a selected vertex set S (where $G[S]$ corresponds to a DPlex) and a candidate vertex set C, the upper bound of the size of the maximum DPlex containing S in $G[S \cup C]$ is $\min_{v \in S} pd_{S \cup C}(v)$.*

Proof. By the definition of DPlex (Definition 1), we have $pd_g(v) \geq |V(g)|$ for each vertex v in a DPlex g. Then, for each vertex $v \in S$, v can be contained in a larger DPlex with size of at most $pd_{S \cup C}(v)$. Thus, the upper bound of the size of the maximum DPlex containing S in $G[S \cup C]$ is clearly $\min_{v \in S} pd_{S \cup C}(v)$. □

The upper bound can be refined using the following lemma.

Lemma 6. *Given a directed graph $G = (V, E)$, a DPlex g exists if and only if there are at least $|V(g)|$ vertices $v \in V$ in G such that $pd_G(v) \geq |V(g)|$.*

Proof. It is obvious that if a DPlex g of G exists, then for each vertex $v \in V(g)$, we have $pd_G(v) \geq pd_g(v) \geq |V(g)|$, as stated in Definition 1. The proof is complete. □

Based on Lemma 6, given a computed upper bound ub, we can check whether the number of vertices in the graph with a pseudo-degree larger than ub exceeds ub. If not, our estimated upper bound can be decremented by 1.

Algorithm. We summarize our proposed upper bound computation algorithm `LDPEnum-UB` in Algorithm 5. In Line 1, the upper bound ub is initially set to $|S \cup C|$. Then, we use Lemma 5 to compute an upper bound in Lines 2-3 and Lemma 6 to refine the computed upper bound in Lines 4-5. Finally, we return ub in Line 6.

The time complexity of Algorithm 5 is analyzed as follows. We first assume we already have $pd_{G[S \cup C]}(\cdot)$; otherwise, we need $O(E(G))$ time to compute. Then, the number of iterations for Lines 2-3 is $|S|$ and each iteration costs $O(1)$ time. For Lines 4-5, we can know the number of iterations is at most $|C|$ and each iteration requires $O(|S|+|C|)$. Thus, the total time complexity of Algorithm 5 is $O(|S| + |C| \cdot (|S| + |C|)) = O(|C| \cdot (|S| + |C|))$.

Algorithm 5: `LDPEnum-UB`(G, S, C, k, ℓ)

Input: A directed graph G, two vertex sets S, C, and two integers k, ℓ

Output: An upper bound of the size of the maximum DPlex containing S in $G[S \cup C]$

```
1 ub ← |S ∪ C|;
2 foreach v ∈ S do
3     ub ← min{ub, pd_{G[S∪C]}(v)};
4 while ub > |{v ∈ S ∪ C | pd_{G[S∪C]}(v) ≥ ub}| do
5     ub ← ub − 1;
6 return ub;
```

Remark. We first note that the following lemma can be used to refine the candidate set C.

Lemma 7 ([25]). *Given a directed graph $G = (V, E)$ and the size constraint s, a vertex v cannot be included in a directed (k, ℓ)-plex of size greater than s if $pd_G(v) \leq s$.*

Specifically, we refine the candidate set C into a smaller one C' by iteratively removing from C those vertices for which $pd_{G[S \cup C](v)} < s$ holds. As some vertices are removed from C, a tighter upper bound can be derived by Lemma 5, formally,

$$\min_{v \in S} pd_{G[S \cup C']}(v) \leq \min_{v \in S} pd_{G[S \cup C]}(v). \tag{4}$$

We observe that this new upper bound is smaller than $|S \cup C'|$. Thus, our proposed upper bounds are necessary. Furthermore, as shown in our experiments (Table IV), the variant that applies Lemma 7 without relying on our proposed upper bounds is, on average, over 53% slower than our final algorithm.

B. Our Large Maximal DPlexes Enumeration Algorithm: `LDPEnum`

We now show our final large maximal DPlexes enumeration algorithm `LDPEnum`. Basically, `LDPEnum` shares similarities with our algorithm `DPEnum` introduced in Section IV, which implements the techniques of divide-and-conquer and decomposed enumeration with pivot as in Algorithms 2 and 3 for the MDPE problem. For completeness, we present our `LDPEnum` in Algorithm 6 (which invokes `LDPEnum-Pivot` in Appendix of our technical report [1]), and highlight the newly incorporated speed-up techniques in the following.

First, `LDPEnum` does not invoke `DPEnum-Pre` since `DPEnum-Pre` mainly focuses on directly outputting maximal DPlexes that are of size at most k, while `LDPEnum` requires the outputted DPlexes that are of size at least $s \geq k + \ell - 1$. Note that, with $k \leq \ell$, the only exception is when $k = \ell = 1$ occurs, where a directed $(1, 1)$-plex corresponds to a directed clique that is not the main focus of this paper.

Second, in Line 1 of Algorithm 6, `LDPEnum` invokes `GraphReduction` using the size constraint s, which is based on the following lemma.

TABLE I: Statistics of Datasets

ID	Dataset	$\|V\|$	$\|E\|$	Density	δ	d^{in}_{max}	d^{out}_{max}	$d^{out}_{avg}(d^{in}_{avg})$
G1	jazz	198	5,484	0.140594	29	100	100	27.70
G2	email-Eu-core	1,005	24,929	0.024706	26	333	211	24.80
G3	political-blogs	1,224	33,430	0.022332	36	351	351	27.31
G4	OTC	5,881	35,592	0.001029	18	763	535	6.05
G5	Wiki-Vote	7,115	103,689	0.002049	15	893	457	14.57
G6	wikipedia-link-als	29,511	759,789	0.000872	96	1,093	5,484	25.75
G7	soc-Epinions	75,888	508,837	0.000088	42	1,801	3,035	6.71
G8	MathOverflow	88,580	227,991	0.000029	48	1,848	968	2.57
G9	wikipedia-link-new	100,573	1,808,548	0.000179	374	3,764	31,736	17.98
G10	web-NotreDame	325,729	1,469,679	0.000014	151	3,444	10,721	4.51
G11	Amazon0601	403,394	3,387,388	0.000021	10	10	2,751	8.40
G12	higgs-twitter	456,626	14,855,819	0.000071	90	1,259	51,386	32.53
G13	web-Google	916,428	5,105,039	0.000005	32	456	6,326	5.57
G14	web-baidu-baike	2,141,300	6,726,387	0.000004	27	2,477	97,848	8.23
G15	cit-Patents	3,774,768	16,518,947	0.000001	1	770	779	4.38

Algorithm 6: `LDPEnum`

Input: A directed graph G, the size constraint s, and two integers k, ℓ

Output: All DPlexes of size at least s in G

1 $G \leftarrow$ `GraphReduction`(G, k, ℓ, s);
2 Sort V as $v_1, v_2, ..., v_n$ according to the out-degeneracy ordering;
3 **for** $i = 1$ *to* n **do**
4 $g \leftarrow G[\{v_i, v_2, ..., v_n\}]$;
5 Reduce g using Property 2 and Lemma 8;
6 **foreach** $P \subseteq V(g) \setminus (\{v_i\} \cup N^{out}_g(v_i))$ *satisfying* $|P| \leq k-1$ **do**
7 $S \leftarrow P \cup \{v_i\}$;
8 $C \leftarrow \{u \in N^{out}_g(v_i) \mid G[S \cup \{u\}]$ is a DPlex$\}$;
9 $X \leftarrow \{u \in V(G) \setminus (S \cup C) \mid G[S \cup \{u\}]$ is a DPlex$\}$;
10 `LDPEnum-Pivot`(g, S, C, X, k, ℓ, s);

Lemma 8 ([25]). *Given a directed graph $G = (V, E)$ and the size constraint s, we consider the following two cases:*

- *(1) Both edges (u, v) and (v, u) are in E: if $|N^{in}_G(u) \cap N^{in}_G(v)| \leq s - 2\ell$ or $|N^{out}_G(u) \cap N^{out}_G(v)| \leq s - 2k$, and*
- *(2) Exactly one of edges (u, v) and (v, u) exists in E: if $|N^{in}_G(u) \cap N^{in}_G(v)| \leq s - 2\ell + 1$ or $|N^{out}_G(u) \cap N^{out}_G(v)| \leq s - 2k + 1$,*

then vertices u and v cannot be included in a directed (k, ℓ)-plex of size greater than s at the same time.

By using Lemmas 7 and 8, `GraphReduction` (as shown in Appendix of [1]) removes unqualified vertices and edges using a method similar to `CTCP` for undirected graphs [10].

Third, in Line 5 of Algorithm 6, `LDPEnum` reduces the graph based on Property 2 and Lemma 8. That is, we check whether each vertex v in $V(g)$ (1) is strongly connected to v_i and has a distance within 2 by Property 2, and (2) has enough number of common in-neighbors and out-neighbors with v_i by Lemma 8. We then reduce g by removing those related unpromising vertices.

Fourth, compared with `DPEnum-Pivot`, `LDPEnum-Pivot` additionally incorporates the upper bounding techniques (in Section VI.A) for pruning unnecessary branches. Thus, the pseudo-code of `LDPEnum-Pivot` (shown in the appendix of our technical report [1]) is similar to that of `DPEnum-Pivot` (Algorithm 3). Specifically, we compute a tighter upper bound ub using `LDPEnum-UB`, based on Lemmas 5 and 6, and compare it with the size constraint s.

Time Complexity. Compared to `DPEnum`, `LDPEnum` mainly incorporates `GraphReduction` and `LDPEnum-UB`. These techniques can be applied in polynomial time, and thus do not affect the overall time complexity. Thus, the worst-case time complexity of `LDPEnum` is equivalent to that of `DPEnum`.

VII. Experimental Evaluations

Setup. We conduct experimental evaluations by comparing our proposed `DPEnum` (Algorithm 2) and `LDPEnum` (Algorithm 6) with the baseline `BaselineBK` (Algorithm 1) for both the MDPE and LMDPE problems, respectively. All algorithms are coded in C++, compiled with g++ -O3, and run on a Linux server equipped with an Intel(R) Core(TM) i5-10500 CPU and 256GB memory. We set the time limit as 24 hours (i.e., 86,400 seconds) and use **INF** to indicate the time exceeds the limit. Our source codes can be found in [1].

Datasets. Following previous studies on cohesive directed subgraphs [23], [25], we select 15 real-world datasets from the website (https://law.di.unimi.it/datasets.php) and SNAP (http://snap.stanford.edu). We summarize the dataset statistics in Table I, where the density is defined by $|E|/(|V| \cdot (|V|-1))$, d^{in}_{max} and d^{out}_{max} (resp. d^{in}_{avg} and d^{out}_{avg}) denote the maximum (resp. the average) in-degree and out-degree, respectively.

A. Experiments for Enumerating Maximal DPlexes

Exp-1: Comparison between `DPEnum` and the baseline. In this experiment, we compare our `DPEnum` with the baseline `BaselineBK` by using different parameter settings of (k, ℓ). We first show the results with $(k, \ell) = (2, 2)$ in Figure 3. We observe that `DPEnum` can solve the problem for all datasets (note that G12 completes in 83,884 seconds), while `BaselineBK` only solves 5 out of 15 datasets in 24 hours. Moreover, `DPEnum` shows improved efficiency over `BaselineBK` across all datasets. In particular, on G8, `DPEnum` can return all maximal DPlexes in 30.06 seconds, while `BaselineBK` fails to complete the problem within the given 24 hours. This also implies that `DPEnum` runs up to three orders of magnitudes faster than `BaselineBK`.

TABLE II: DPlexes with varying k and ℓ

ID	(k, ℓ)	Running time	# maximal DPlexes
G1	(2, 2)	0.11	35,214
	(3, 3)	6.44	3,602,575
	(4, 4)	298.87	193,056,583
G2	(2, 2)	1.03	747,269
	(2, 3)	1.80	933,532
	(2, 4)	1.95	939,216
	(3, 3)	143.82	202,157,112
	(3, 4)	183.99	214,056,849
	(4, 4)	18,447.40	43,982,393,611
G3	(2, 2)	6.86	3,160,093
	(3, 3)	936.44	525,856,574
G4	(2, 2)	2.69	17,363,311
	(3, 3)	3,710.36	33,912,597,657
G5	(2, 2)	0.44	25,317,234
	(3, 3)	226.25	60,007,000,734

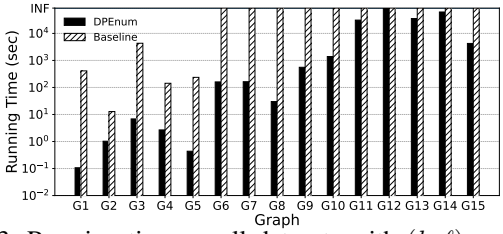


Fig. 3: Running time on all datasets with $(k, \ell) = (2, 2)$

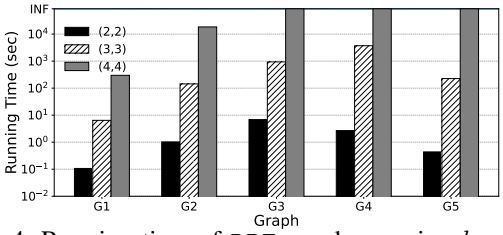


Fig. 4: Running time of DPEnum by varying k and ℓ

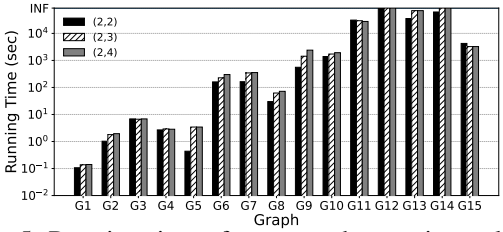


Fig. 5: Running time of DPEnum by varying only ℓ

We next conduct experiments for our DPEnum on G1-G5 by varying k and ℓ, i.e., $(2, 2)$, $(3, 3)$, and $(4, 4)$, in Figure 4. We can observe that with the increasing of k and ℓ, the running time obviously increases. This is because, with a larger value of k and ℓ, the maximal DPlexes become sparser and the number of maximal DPlexes grows rapidly. We also present the running time of DPEnum with varying values of ℓ and a fixed value of $k = 2$ in Figure 5. In most cases, the running time grows when ℓ increases; however, such as G4 and G11, DPEnum runs slower with $(2, 3)$ than with $(2, 4)$. The reason may be due to the fact that the smaller number of maximal DPlexes at $(2, 4)$ compared to $(2, 3)$ in these graphs. For G15, DPEnum exhibits slower performance with $(2, 2)$ compared to $(2, 3)$ and $(2, 4)$, since (1) the preprocessing technique consumes most of the time, and (2) the case where $k = \ell$ requires more techniques compared to when $k \neq \ell$.

In Table II, we also present the number of maximal DPlexes and observe that the trend closely aligns with the running time: as k and ℓ increase, the number of maximal DPlexes grows rapidly. In most cases, the number of maximal DPlexes increases as ℓ increases, while k is held constant. The remaining results for all datasets are provided in the technical report [1].

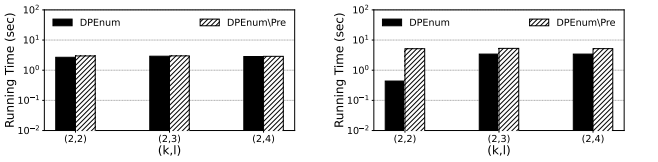
Exp-2: Effect of the preprocessing in DPEnum. We compare DPEnum with DPEnum\Pre, which is DPEnum without using our proposed preprocessing technique (Algorithm 4 in Section V) with $(k, \ell) = (2, 2)$ in Table III. We can observe that DPEnum outperforms DPEnum\Pre in most cases. In particular, DPEnum solves G5 with 0.44 seconds while DPEnum\Pre completes in 5.16 seconds, which implies an 11× speedup. Moreover, DPEnum can solve G12-G15 that DPEnum\Pre cannot in 24 hours. The superiority of DPEnum may be due to the fact that our preprocessing technique can be used to directly return several maximal DPlexes without conducting the enumeration. We also show the results for DPEnum and DPEnum\Pre with different values of (k, ℓ) in Figure 6. We observe that the preprocessing technique is more effective with $(2, 2)$ than with $(2, 3)$ and $(2, 4)$. This is because with $(2, 2)$, we can identify more reducible vertices via Lemmas 2, 3, and 4 as $k = \ell$, which may improve the overall efficiency. These results demonstrate the effectiveness of our preprocessing technique.

Exp-3: Effect of the pivot technique in DPEnum. We compare DPEnum with DPEnum\Piv, which removes the pivot technique in Algorithm 3, i.e., we randomly select a vertex in the candidate set C for enumeration. Comparing the running time in Figure 7(a) and 7(b), we can see that DPEnum performs better, although the improvement is not significant. This is because directed graphs are usually sparse, so selecting the pivot vertex or not always reduces the size of the candidate set. On the other hand, for undirected graphs (note that undirected graphs are special cases of directed graphs) in Figures 7(c) and 7(d), DPEnum performs much better than DPEnum\Piv. Specifically, in Figure 7(c) with $(k, \ell) = (2, 2)$, DPEnum\Piv costs 134.96 seconds, while DPEnum runs only 0.11 seconds, which achieves a speedup of at least 1200×. We defer additional experiments on abolition studies for DPEnum to Appendix of the technical report [1].

Exp-4: Scalability test. To evaluate scalability, we conduct experiments on the two largest graph G14 and G15 with $(k, \ell) = (2, 2)$. We randomly sample 20%–100% of the vertices and compare DPEnum with Baseline. Figure 8 shows the following key trend: The running time decreases significantly as the graph size reduces, mainly due to graph sparsification, which rapidly reduces the number of maximal DPlexes. These results validate the scalability of DPEnum.

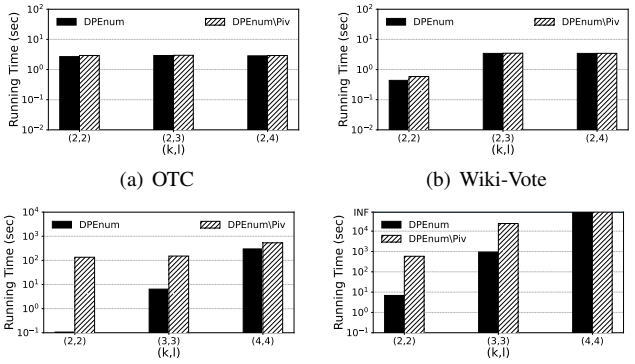
TABLE III: Abolition studies of DPEnum with $(k, \ell) = (2, 2)$

ID	DPEnum	DPEnum\Pre	DPEnum\Piv
G1	**0.11**	0.11	134.96
G2	**1.03**	1.11	2.37
G3	**6.86**	6.86	581.26
G4	**2.69**	2.97	2.93
G5	**0.44**	5.16	0.59
G6	**160.50**	290.48	INF
G7	**163.27**	427.05	178.76
G8	**30.06**	135.18	49.01
G9	**558.56**	2,006.09	INF
G10	**1,395.79**	4,228.34	INF
G11	31,103.86	**27,228.32**	27,343.78
G12	**83,884.37**	INF	INF
G13	35,496.18	INF	**33,014.69**
G14	**62,076.22**	INF	62,271.99
G15	**4259.76**	INF	4263.26



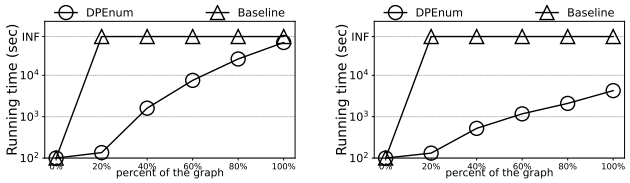
(a) OTC (b) Wiki-Vote

Fig. 6: Runtime comparison between DPEnum and DPEnum\Pre



(a) OTC (b) Wiki-Vote (c) jazz (d) political-blogs

Fig. 7: Runtime comparison between DPEnum and DPEnum\Piv



(a) web-baidu-baike (b) cit-Patents

Fig. 8: Scalability test

B. Experiments for Enumerating Large Maximal DPlexes

Exp-5: Comparison between LDPEnum and the baseline. We first compare the running time between LDPEnum and the baseline BaselineBK in Figure 9 (and defer the additional experiments to Appendix of [1]). From Figure 9, we observe that LDPEnum successfully completes 14 of the datasets, while BaselineBK fails to solve 13 out of 15 datasets within 24 hours. In general, when the graph size increases, the time gap between the two algorithms is more obvious. Moreover, for G5, LDPEnum runs in 7.95 seconds and BaselineBK faces timeout (86,400 seconds). This demonstrates that LDPEnum is up to four orders of magnitudes faster than BaselineBK.

We also compare LDPEnum with DPEnum in terms of running time and the number of found maximal DPlexes, with the detailed comparison is provided in the appendix of [1]. The results show that LDPEnum outperforms DPEnum, which demonstrates the effectiveness of GraphReduction.

Exp-6: Effect of the upper bound in LDPEnum. We compare LDPEnum with LDPEnum\UB, which is LDPEnum without using the upper bounding technique (Algorithm 5) in Section VI.A. Table IV presents the results with $(k, \ell, s) = (3, 3, 5)$. We observe that in general, the technique of upper bound can be used to speed up the whole performance, where the speedup factor is up to 2.74× on G13. We then show the effectiveness of our proposed upper bounding technique via varying (k, ℓ, s) in Figure 10. We can find in Figures 10(a) and 10(b) that our upper bounding method is more effective with larger values of (k, ℓ, s), since a larger value of s implies that a recursion is more likely to be pruned by Lemmas 4 and 5. Moreover, as shown in Figure 10(b) with $(k, \ell, s) = (5, 5, 9)$, the running time of LDPEnum\UB is 18209.58 seconds, whereas LDPEnum completes in only 1395.41 seconds, resulting in a speedup of 13.05×. We also study the effect of applying Lemma 7 to refine candidate sets by comparing LDPEnum to LDPEnum\UB+Lem7, which refines candidate sets using Lemma 7 instead of relying on our proposed upper bounds. From Table IV, we observe that LDPEnum\UB+Lem7 is, on average, over 53% slower than LDPEnum. This is because applying Lemma 7 at each branch iteratively removes vertices and computes $pd(\cdot)$ for each vertex, which incurs significant additional computational costs.

Exp-7: Performance of LDPEnum with large k and ℓ. We evaluate LDPEnum on large values of k and ℓ, with $s = k + \ell + 3$. As shown in Figure 11, the running time increases consistently as k and ℓ increase. In particular, for Wiki-Vote (Figure 11(b)), the running time stabilizes when $k \geq 5$, as the number of output DPlexes approaches zero.

VIII. Related Work

Directed cohesive subgraphs. In the literature, several cohesive subgraphs for directed graphs have been proposed and studied. Based on the k-core model, Giatsidis et al. [26] and Fang et al. [23] studied the directed (k, ℓ)-core, where each vertex in the subgraph has at least k in-neighbors and ℓ out-neighbors. Liu et al. [36] and Tian et al. [43] considered the concept of (k_c, k_f)-truss, which generalizes the k-truss in undirected graphs and distinguishes two different types of triangles in semantics for directed graphs. Guo et al. [27] designed efficient algorithms for the problem related to the directed (γ_1, γ_2)-quasi-clique, where each vertex has a fraction γ_1 of out-neighbors and a fraction γ_2 of in-neighbors in the subgraph. Ma et al. [38], [39] focused on the directed

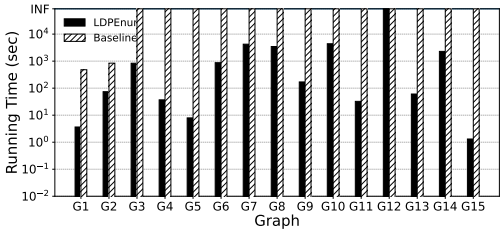


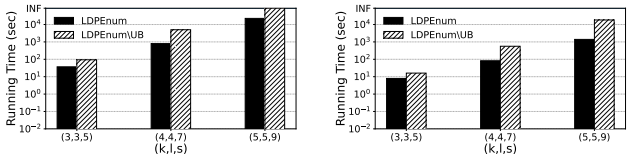
Fig. 9: Running time on all datasets with $(k, \ell, s) = (3, 3, 5)$

TABLE IV: Abolition studies of LDPEnum with $(k, \ell, s) = (3, 3, 5)$

ID	LDPEnum	LDPEnum\UB	LDPEnum\UB+Lem7
G1	**3.66**	3.72	4.37
G2	**74.08**	82.61	97.39
G3	838.8	**735.84**	957.28
G4	**36.97**	93.77	84.83
G5	**7.95**	16.09	12.48
G6	**876.25**	1,661.83	1578.97
G7	**4,245.31**	6,577.25	8259.92
G8	**3,481.79**	4,494.48	6245.57
G9	**170.07**	255.31	301.38
G10	**4,403.9**	5,843.12	7041.24
G11	**32.52**	32.91	32.75
G12	INF	INF	INF
G13	**60.8**	77.62	96.32
G14	**2,296.21**	6,296.84	4541.96
G15	**1.28**	1.29	**1.28**

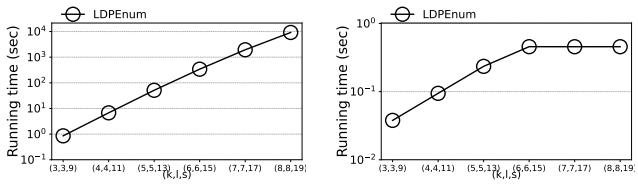
densest subgraph, which refers to a subgraph whose density is the highest. Closest to our paper, Gao et al. [25] introduced the directed (k, ℓ)-plex (DPlex) model and investigated the maximum DPlex problem, which aims to identify the DPlex with the largest size. In our work, we consider the maximal DPlexes enumeration problem, which returns multiple DPlexes rather than a single maximum DPlex. Thus, the algorithm proposed in [25] cannot be directly adapted to solve our problem. We note that all the directed models above define dense subgraphs from different perspectives. However, none of their algorithms can be directly applied to solve our maximal DPlexes enumeration problem. Moreover, as illustrated in [25] via a case study of community search, the DPlex performs better than other directed cohesive subgraph structures including directed core, directed truss, directed clique, and so on.

k-plex. The k-plex is an important cohesive subgraph that has recently been extensively studied in undirected graphs. Many efficient algorithms are proposed for enumerating maximal k-plexes in undirected graphs [6], [13]–[15], [19], [46], [47], [49], [57]. The majority of previous studies is based on the classical Bron-Kerbosch (BK) algorithm with various speed-up techniques. Specifically, Conte et al. [14] generalized the pivoting technique for the BK algorithm, while Zhou et al. [57] and Dai et al. [19] provided more advanced pivoting techniques for the problem. In addition, Wang et al. [46] designed effective reduction methods for shrinking the graph. Very recently, Cheng et al. [13] proposed the state-of-the-art algorithm for the problem by proposing a new pivot selection approach, an effective upper bound, and novel pruning techniques. Another algorithmic framework is based on the *reverse search* [6],



(a) OTC (b) Wiki-Vote

Fig. 10: Runtime comparison between LDPEnum and LDPEnum\UB



(a) OTC (b) Wiki-Vote

Fig. 11: Runtime in LDPEnum with large k and ℓ

which provides a polynomial delay (i.e., the time between any two consecutive solutions is polynomially bounded). However, the reverse search-based algorithm is less efficient than BK-based algorithms as verified in previous work [19], [46]. Moreover, there are recent studies focusing on the maximum k-plex search problem, which aims to obtain the k-plex with the largest size in the graph [10], [11], [24], [31], [32], [45], [51], [56]. These algorithms normally make use of the *branch-and-bound* framework along with reduction and bounding techniques to boost the practical performance. We remark that, as mentioned before, all algorithms for the maximal k-plexes enumeration and the maximum k-plex search problems are unable to address the maximal DPlexes enumeration problem due to their disregard for the directional information of edges. Furthermore, the concept of k-plex is also investigated in various types of graphs, such as bipartite graphs [12], [16], [20], [37], [54], [55], temporal graphs [5], [50], uncertain graphs [17], and so on. For other related problems, we refer to excellent books and surveys [9], [21], [22], [29], [35], [52].

IX. Conclusion

In this paper, we considered the problem of enumerating maximal DPlexes in directed graphs. We proposed our algorithm DPEnum, which leverages the pivot technique for enumeration. By introducing the new concept of out-degeneracy in directed graphs, DPEnum achieves an improved time complexity of $O^*(\alpha_k^\delta)$. We also proposed a novel preprocessing technique for DPEnum that directly identifies several maximal DPlexes without enumeration. We also studied the enumeration of large maximal DPlexes and developed an algorithm LDPEnum, incorporating newly devised upper bounding methods. Our experimental studies demonstrated that both DPEnum and LDPEnum are up to four orders of magnitude faster than the baseline. For future work, we will consider parallel or distributed versions of both DPEnum and LDPEnum.

REFERENCES

[1] https://github.com/sykhhhh/dplex-enumeration-master, Technical Report and Source Code.

[2] R. Albert, H. Jeong, and A.-L. Barabási, "Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.

[3] B. Balasundaram, S. Butenko, and I. V. Hicks, "Clique relaxations in social network analysis: The maximum k-plex problem," *Operations Research*, vol. 59, no. 1, pp. 133–142, 2011.

[4] V. Batagelj and M. Zaveršnik, "An $O(m)$ algorithm for cores decomposition of networks," *CoRR*, vol. cs.DS/0310049, 2003.

[5] M. Bentert, A.-S. Himmel, H. Molter, M. Morik, R. Niedermeier, and R. Saitenmacher, "Listing all maximal k-plexes in temporal graphs," *ACM J. Exp. Algorithmics*, vol. 24, pp. 1–27, 2019.

[6] D. Berlowitz, S. Cohen, and B. Kimelfeld, "Efficient enumeration of maximal k-plexes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2015, pp. 431–444.

[7] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[8] L. Chang, "Efficient maximum clique computation over large sparse graphs," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 529–538.

[9] L. Chang and L. Qin, *Cohesive Subgraph Computation over Large Sparse Graphs*. Springer, 2018.

[10] L. Chang, M. Xu, and D. Strash, "Efficient maximum k-plex computation over large sparse graphs," *Proceedings of the VLDB Endowment*, vol. 16, no. 2, pp. 127–139, 2022.

[11] L. Chang and K. Yao, "Maximum k-plex computation: Theory and practice," *Proceedings of the ACM on Management of Data (SIGMOD)*, vol. 2, no. 1, pp. 1–26, 2024.

[12] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient exact algorithms for maximum balanced biclique search in bipartite graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2021, pp. 248–260.

[13] Q. Cheng, D. Yan, T. Wu, L. Yuan, J. Cheng, Z. Huang, and Y. Zhou, "Efficient enumeration of large maximal k-plexes," in *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2025, pp. 53–65.

[14] A. Conte, T. De Matteis, D. De Sensi, R. Grossi, A. Marino, and L. Versari, "D2K: Scalable community detection in massive networks via small-diameter k-plexes," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1272–1281.

[15] A. Conte, D. Firmani, C. Mordente, M. Patrignani, and R. Torlone, "Fast enumeration of large k-plexes," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 115–124.

[16] Q. Dai, R.-H. Li, D. Cui, M. Liao, Y.-X. Qiu, and G. Wang, "Efficient maximal biplex enumerations with improved worst-case time guarantee," *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, vol. 2, no. 3, pp. 1–26, 2024.

[17] Q. Dai, R.-H. Li, M. Liao, H. Chen, and G. Wang, "Fast maximal clique enumeration on uncertain graphs: A pivot-based approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2022, pp. 2034–2047.

[18] Q. Dai, R.-H. Li, M. Liao, and G. Wang, "Maximal defective clique enumeration," *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, vol. 1, no. 1, pp. 1–26, 2023.

[19] Q. Dai, R.-H. Li, H. Qin, M. Liao, and G. Wang, "Scaling up maximal k-plex enumeration," in *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2022, p. 345–354.

[20] Q. Dai, R.-H. Li, X. Ye, M. Liao, W. Zhang, and G. Wang, "Hereditary cohesive subgraphs enumeration on bipartite graphs: The power of pivot-based approaches," *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, vol. 1, no. 2, pp. 1–26, 2023.

[21] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin, "A survey of community search over big graphs," *The VLDB Journal*, vol. 29, pp. 353–392, 2020.

[22] Y. Fang, K. Wang, X. Lin, and W. Zhang, "Cohesive subgraph search over big heterogeneous information networks: Applications, challenges, and solutions," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2021, pp. 2829–2838.

[23] Y. Fang, Z. Wang, R. Cheng, H. Wang, and J. Hu, "Effective and efficient community search over large directed graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2093–2107, 2018.

[24] J. Gao, J. Chen, M. Yin, R. Chen, and Y. Wang, "An exact algorithm for maximum k-plexes in massive graphs," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 1449–1455.

[25] S. Gao, K. Yu, S. Liu, C. Long, and Z. Qiu, "On searching maximum directed (k, ℓ)-plex," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2024, pp. 2570–2583.

[26] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, "D-cores: Measuring collaboration of directed graphs based on degeneracy," *Knowledge and Information Systems*, vol. 35, no. 2, pp. 311–343, 2013.

[27] G. Guo, D. Yan, L. Yuan, J. Khalil, C. Long, Z. Jiang, and Y. Zhou, "Maximal directed quasi-clique mining," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2022, pp. 1900–1913.

[28] E. Harley, A. Bonner, and N. Goodman, "Uniform integration of genome mapping data using intersection graphs," *Bioinformatics*, vol. 17, no. 6, pp. 487–494, 2001.

[29] X. Huang, L. V. S. Lakshmanan, and J. Xu, *Community Search over Big Graphs*. Morgan & Claypool Publishers, 2019.

[30] A. Java, X. Song, T. Finin, and B. Tseng, "Why we twitter: Understanding microblogging usage and communities," in *Proceedings of the WebKDD and SNA-KDD Workshop on Web Mining and Social Network Analysis*, 2007, pp. 56–65.

[31] H. Jiang, F. Xu, Z. Zheng, B. Wang, and W. Zhou, "A refined upper bound and inprocessing for the maximum k-plex problem," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2023, pp. 5613–5621.

[32] H. Jiang, D. Zhu, Z. Xie, S. Yao, and Z.-H. Fu, "A new upper bound based on vertex partitioning for the maximum k-plex problem." in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021, pp. 1689–1696.

[33] J. Khalil, D. Yan, G. Guo, and L. Yuan, "Parallel mining of large maximal quasi-cliques," *The VLDB Journal*, vol. 31, no. 4, pp. 649–674, 2022.

[34] D. Kratsch and F. Fomin, *Exact Exponential Algorithms*. Springer-Verlag Berlin Heidelberg, 2010.

[35] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal, "A survey of algorithms for dense subgraph discovery," *Managing and Mining Graph Data*, pp. 303–336, 2010.

[36] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Truss-based community search over large directed graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2020, pp. 2183–2197.

[37] W. Luo, K. Li, X. Zhou, Y. Gao, and K. Li, "Maximum biplex search over bipartite graphs," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2022, pp. 898–910.

[38] C. Ma, Y. Fang, R. Cheng, L. V. Lakshmanan, W. Zhang, and X. Lin, "Efficient algorithms for densest subgraph discovery on large directed graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2020, pp. 1051–1066.

[39] ——, "On directed densest subgraph discovery," *ACM Transactions on Database Systems (TODS)*, vol. 46, no. 4, pp. 1–45, 2021.

[40] S. B. Seidman, "Clique-like structures in directed networks," *Journal of Social and Biological Structures*, vol. 3, no. 1, pp. 43–54, 1980.

[41] ——, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269–287, 1983.

[42] S. B. Seidman and B. L. Foster, "A graph-theoretic generalization of the clique concept," *Journal of Mathematical Sociology*, vol. 6, no. 1, pp. 139–154, 1978.

[43] A. Tian, A. Zhou, Y. Wang, and L. Chen, "Maximal D-truss search in dynamic directed graphs," *Proceedings of the VLDB Endowment*, vol. 16, no. 9, pp. 2199–2211, 2023.

[44] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28–42, 2006.

[45] Z. Wang, Y. Zhou, C. Luo, and M. Xiao, "A fast maximum k-plex algorithm parameterized by the degeneracy gap," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2023, pp. 5648–5656.

[46] Z. Wang, Y. Zhou, M. Xiao, and B. Khoussainov, "Listing maximal k-plexes in large real-world graphs," in *Proceedings of the ACM Web Conference (WWW)*, 2022, pp. 1517–1527.

[47] Z. Wang, Q. Chen, B. Hou, B. Suo, Z. Li, W. Pan, and Z. G. Ives, "Parallelizing maximal clique and k-plex enumeration over graph data," *Journal of Parallel and Distributed Computing*, vol. 106, pp. 79–91, 2017.

[48] D. Weiss and G. Warner, "Tracking criminals on facebook: A case study from a digital forensics REU program," 2015.
[49] B. Wu and X. Pei, "A parallel algorithm for enumerating all the maximal k-plexes," in *Proceedings of the International Workshops on Emerging Technologies in Knowledge Discovery and Data Mining (PAKDD Workshop)*, 2007, pp. 476–483.
[50] Y. Wu, R. Sun, X. Wang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "Efficient maximal temporal plex enumeration," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2024.
[51] M. Xiao, W. Lin, Y. Dai, and Y. Zeng, "A fast algorithm to compute maximum k-plexes in social network analysis," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2017, pp. 919–925.
[52] K. Yu and C. Long, "Graph mining meets fake news detection," in *Data Science for Fake News: Surveys and Perspectives*. Springer, 2021, pp. 169–189.
[53] ——, "Fast maximal quasi-clique enumeration: A pruning and branching co-design approach," *Proc. ACM Manag. Data (SIGMOD)*, vol. 1, no. 3, pp. 211:1–211:26, 2023.
[54] ——, "Maximum k-biplex search on bipartite graphs: A symmetric-BK branching approach," *Proceedings of the ACM on Management of Data (SIGMOD)*, vol. 1, no. 1, pp. 1–26, 2023.
[55] K. Yu, C. Long, S. Liu, and D. Yan, "Efficient algorithms for maximal k-biplex enumeration," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2022, p. 860–873.
[56] Y. Zhou, S. Hu, M. Xiao, and Z.-H. Fu, "Improving maximum k-plex solver via second-order reduction and graph color bounding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, no. 14, 2021, pp. 12 453–12 460.
[57] Y. Zhou, J. Xu, Z. Guo, M. Xiao, and Y. Jin, "Enumerating maximal k-plexes with worst-case time guarantee," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020, pp. 2442–2449.

APPENDIX

A. Omitted Proofs

1) Proof of Property 4:

Proof. For a subgraph g_1 in Definition 2, all vertices in g_1 have out-degrees of at least δ. Thus, g_1 contains at least $\delta+1$ vertices, and we can derive:

$$\sum_{v \in V(g_1)} d^{out}_{g_1}(v) \geq \sum_{v \in V(g_1)} \delta \geq (\delta+1)\delta > \delta^2. \quad (5)$$

On the other hand, as g_1 is a subgraph of G, the number of edges in g_1 does not exceed m, i.e.,

$$\sum_{v \in V(g)} d^{out}_{g_1}(v) = |E(g_1)| \leq m. \quad (6)$$

Combining Inequalities (5) and (6), we can get $\delta < \sqrt{m}$. □

2) Proof of Property 5:

Proof. Suppose, to the contrary that, there exists a directed (k,ℓ)-plex g with size of at least $\delta+k+1$. By the definition of DPlex, all vertices in g have out-degrees at least $|V(g)|-k \geq \delta+k+1-k = \delta+1$, which contradicts to the definition of out-degeneracy δ. □

3) Proof of Lemma 3:

Proof. We prove by contradiction. Suppose, to the contrary, that v is not a reducible vertex given the condition stated in the lemma. Then, there must exist a directed (k,ℓ)-plex $G[S]$ containing v with $|S| = k+1$. Due to the definition of DPlex, we know that each vertex $v \in S$ has at least one out-neighbor, which implies that there must exist a cycle in the directed walk of length $k+1$ starting from v in S. This contradicts our condition stated in the lemma. Further, the case of $k = \ell$ can be proved analogously, and the proof is complete. □

4) Proof of Lemma 4:

Proof. We prove by contradiction. Suppose, to the contrary, that v is not a reducible vertex given the condition stated in the lemma. Then, there must exist a directed (k,ℓ)-plex $G[S]$ containing v with $|S| = k+1$. Due to the definition of DPlex, we know that each vertex $v \in S$ has at least one out-neighbor and one in-neighbor. Note that v does not have bidirectional edges; that is, for all $w \in V$, we do not have both (v,w) and (w,v) in E. Otherwise, v would be part of a cycle of length 2, contradicting the condition stated in the lemma. This also implies that we have $k = \ell \geq 2$.

Consider a walk $u \to v \to w$ in S and $u \neq w$. We consider the following two cases. **Case 1: $k = \ell = 2$.** We know that $|S| = 3$. Then, vertex w must have one out-neighbor, which is not v; thus, there is an edge from w to u. However, this implies that v is contained in a cycle of length 3, contradicting the given condition in the lemma. **Case 2: $k = \ell = 3$.** Now, we have $|S| = 4$. We let the other vertex in S be x (besides the vertices u, v, and w). It is clear that we cannot have $w \to u$; otherwise there will be a cycle of length 3. Thus, we must have $w \to x$ and $x \to u$, which form a cycle of length 4. This also contradicts the condition stated in the lemma. □

B. Pseudo-code of `LDPEnum-Pivot`

For completeness, we provide the pseudo-code of `LDPEnum-Pivot` in Algorithm 7 (described in Section VI) as follows. Compared to Algorithm 6, Algorithm 7 mainly incorporates our upper bounding techniques in Lines 2-3.

Algorithm 7: `LDPEnum-Pivot`(G, S, C, X, k, ℓ, s)

Output: All directed (k,ℓ)-plexes of size at least s in $G[S \cup C]$

1 $C \leftarrow \{u \in C \mid G[S \cup \{u\}]$ is a DPlex$\}$;
2 $X \leftarrow \{u \in X \mid G[S \cup \{u\}]$ is a DPlex$\}$;
3 **if** `LDPEnum-UB`$(G, S, C, k, \ell) < s$ **then**
4 **return**;
5 $M \leftarrow \arg\min_{v \in S \cup C} pd_{G[S \cup C]}(v)$;
6 $u \leftarrow \arg\min_{v \in M} pd_{G[S \cup \{v\}]}(v)$;
7 **if** $pd_{G[S \cup C]}(u) \geq |S \cup C|$ **then**
8 **if** $S \cup C \cup \{w\}$ *is not a DPlex* $\forall w \in X$ **then**
9 Output $S \cup C$;
10 **return**;
11 **if** $u \in S$ **then**
12 **if** $pd_{G[S]}(u) = d^{out}_{G[S]}(u) + k$ **then**
13 $u_{new} \leftarrow \arg\min_{v \in C \setminus N^{out}_G(u)} pd_{G[S \cup \{v\}]}(v)$;
14 **else**
15 $u_{new} \leftarrow \arg\min_{v \in C \setminus N^{in}_G(u)} pd_{G[S \cup \{v\}]}(v)$;
16 $u \leftarrow u_{new}$;
17 `LDPEnum-Pivot`$(G, S \cup \{u\}, C \setminus \{u\}, X, k, \ell, s)$;
18 `LDPEnum-Pivot`$(G, S, C \setminus \{u\}, X \cup \{u\}, k, \ell, s)$;

C. Pseudo-code of `GraphReduction`

For completeness, we provide the pseudo-code of `GraphReduction` (Algorithm 8) (described in Section VI). Specifically, `GraphReduction` checks every vertex and edge in the current graph. If a vertex or edge does not satisfy Lemmas 7 or 8, it is removed (Lines 4 and 7). The process continues iteratively until no further reductions can be made to the graph (Lines 1-7).

Algorithm 8: `GraphReduction`

Input: A directed graph G and three integers k, ℓ, s
Output: A reduced graph G

1 **while** G *has been reduced* **do**
2 **foreach** $v \in V$ **do**
3 **if** v *satisfies Lemma 7* **then**
4 Remove vertex v from G;
5 **foreach** $e \in E$ **do**
6 **if** e *satisfies Lemma 8* **then**
7 Remove edge e from G;
8 **return** G;

D. Additional Experiments

We provide additional experiments in the appendix. **Number of maximal DPlexes for all datasets.** We provide the remaining results of maximal DPlexes for G6-G15 with $(k, \ell) = (2, 2)$ in Table V. Note that these graphs cannot be solved by our algorithms within the time limit when $k \geq 3$.

TABLE V: DPlexes for G6-G15

ID	(k, ℓ)	Running time	# maximal DPlexes
G6	(2, 2)	160.50	435,414,605
G7	(2, 2)	163.27	2,881,595,485
G8	(2, 2)	30.06	3,924,932,450
G9	(2, 2)	558.56	5,057,132,607
G10	(2, 2)	1,395.79	53,048,892,778
G11	(2, 2)	31,103.86	81,362,263,929
G12	(2, 2)	83,884.37	106,232,558,444
G13	(2, 2)	35,496.18	499,998,716,533
G14	(2, 2)	62,076.22	2,292,581,228,373
G15	(2, 2)	4,259.76	7,124,434,839,528

Ablation Studies of `DPEnum`. Tables VI and VII present the results of ablation studies for `DPEnum` with $(k, \ell) = (2, 3)$ and $(k, \ell) = (2, 4)$. The results demonstrate that the proposed pivot and preprocessing techniques in `DPEnum` significantly accelerate the algorithm in most cases. Specifically, the effectiveness of the preprocessing technique can be verified by comparing the `DPEnum\Pre` and `DPEnum` shown in Tables VI and VII, where `DPEnum` outperforms `DPEnum\Pre` in most cases and can solve two more graphs G13 and G15 within the time limit. For the pivot technique, a comparison with `DPEnum\Piv` shows that `DPEnum` successfully solves three additional graphs within the time limit. For instance, on G6 with $(k, \ell) = (2, 3)$, `DPEnum` completes in only 227.51 seconds, whereas `DPEnum\Piv` exceeds the 24-hour time limit (86,400 seconds). These results highlight the performance improvements achieved by the proposed techniques.

TABLE VI: Abolition studies of `DPEnum` with $(k, \ell) = (2, 3)$

ID	`DPEnum`	`DPEnum\Pre`	`DPEnum\Piv`
G1	**0.14**	0.14	134.38
G2	1.80	**1.72**	3.88
G3	**6.68**	6.68	577.07
G4	**2.92**	2.98	2.97
G5	**3.43**	5.30	3.47
G6	227.51	**217.52**	INF
G7	**400.25**	431.64	408.46
G8	**62.20**	155.28	116.93
G9	**1,432.33**	1,703.63	INF
G10	**1,724.85**	3,603.65	INF
G11	29,448.98	**26,747.38**	29,065.14
G12	INF	INF	INF
G13	**69,537.62**	INF	71,378.83
G14	INF	INF	INF
G15	**3,249.10**	INF	3,314,62

Comparison between `LDPEnum` and the baseline. We compare `LDPEnum` and `BaselineBK` using $(3, 3, 7)$ and $(3, 3, 9)$ in Figures 12 and 13. We observe that `LDPEnum` successfully completes 14 of datasets, whereas `BaselineBK` fails to solve 13 out of 15 datasets within 24 hours. This result is consistent with the observations for the parameter setting $(k, \ell, s) = (3, 3, 5)$. Generally, as the graph size or the size constraint s increases, the performance gap between the two algorithms becomes more obvious. For instance, on dataset G5 with $(k, \ell, s) = (3, 3, 9)$, `LDPEnum` completes in only 0.03 seconds, while `BaselineBK` times out after 86,400 seconds. These results demonstrate that `LDPEnum` is up to seven orders of magnitude faster than `BaselineBK`.

TABLE VII: Abolition studies of `DPEnum` with $(k, \ell) = (2, 4)$

ID	`DPEnum`	`DPEnum\Pre`	`DPEnum\Piv`
G1	**0.14**	0.14	134.07
G2	1.95	**1.75**	4.17
G3	6.86	6.86	577.25
G4	**2.85**	2.89	2.92
G5	**3.43**	5.19	3.44
G6	299.98	**289.48**	INF
G7	**363.27**	419.59	398.90
G8	**71.90**	165.32	141.17
G9	**2,401.85**	2,613.01	INF
G10	**1,934.64**	3,606.25	INF
G11	27,695.87	**26,488.20**	29,674.95
G12	INF	INF	INF
G13	**69,610.11**	INF	71,420.70
G14	INF	INF	INF
G15	**3,259.34**	INF	3,261.20

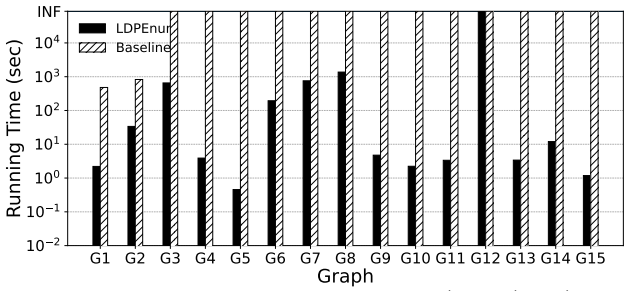


Fig. 12: Running time on all datasets with $(k, \ell, s) = (3, 3, 7)$

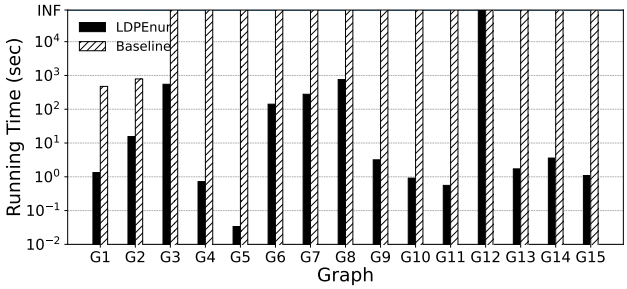


Fig. 13: Running time on all datasets with $(k, \ell, s) = (3, 3, 9)$

Comparison between `LDPEnum` and `DPEnum`. We compare the running time, number of DPlexes, and their density for `LDPEnum` with varying values of s and for `DPEnum` without the size constraint, using $(k, \ell) = (3, 3)$. As shown in Figure VIII, the running time decreases as s increases. This is because, with larger values of s, the number of maximal DPlexes decreases, and the remaining DPlexes tend to be denser and easier to identify. Compared to `DPEnum`, `LDPEnum` generates fewer but denser DPlexes in less time.

These results highlight the importance of imposing a size constraint on s.

TABLE VIII: Comparison between `LDPEnum` and `DPEnum`

Graph	(k, ℓ, s) / (k, ℓ)	# maximal DPlexes	Time	Density
OTC	3,3	33,912,597,657	3,710.36	0.000822
	3,3,5	7,693,998	36.97	0.615708
	3,3,7	637,312	3.90	0.778598
	3,3,9	40,75	0.72	0.845961
Wiki-Vote	3,3	60,007,000,734	226.25	0.000032
	3,3,5	488,900	7.95	0.685156
	3,3,7	22,629	0.46	0.815889
	3,3,9	516	0.03	0.86293