Vol.2019-AL-172 No.1
2019/3/5

IPSJ SIG Technical Report

A Polynomial-delay Algorithm for Enumerating Connectors under Various Connectivity Conditions

Kazuya Haraguchi[1,a)] Hiroshi Nagamochi[2,b)]

Abstract: We are given an instance (G, I, σ) with a graph $G = (V, E)$, a set I of items, and a function $\sigma : V \to 2^I$. For a subset X of V, let $G[X]$ denote the subgraph induced from G by X, and $I_\sigma(X)$ denote the common item set over X. A subset X of V such that $G[X]$ is connected is called a connector if, for any vertex $v \in V \setminus X$, $G[X \cup \{v\}]$ is not connected or $I_\sigma(X \cup \{v\})$ is a proper subset of $I_\sigma(X)$.

In this paper, we present the first polynomial-delay algorithm for enumerating all connectors. For this, we first extend the problem of enumerating connectors to a general setting so that the connectivity condition on X in G can be specified in a more flexible way. We next design a new algorithm for enumerating all solutions in the general setting, which leads to a polynomial-delay algorithm for enumerating all solutions for several connectivity conditions on X in G, such as the biconnectivity of $G[X]$ or the k-edge-connectivity among vertices in X in G.

1. Introduction

In this paper, we consider enumeration of subgraphs in a given *attributed graph*, that is, vertices are given items. The subgraphs should be connected, and at the same time, be maximal with respect to the common item set.

Let us review related studies. For a usual graph (i.e., a non-attributed graph), there are some studies on enumeration of connected subgraphs. Avis and Fukuda [3] showed that all connected induced subgraphs are enumerable in output-polynomial time and in polynomial space, by means of reverse search. Nutov [9] showed that minimal undirected Steiner networks, and minimal k-connected and k-outconnected spanning subgraphs are enumerable in incremental polynomial time. Wasa [15] develops a catalog of enumeration problems in the literature.

For an attributed graph, community detection [7] and frequent subgraph mining [6] are among significant graph mining problems. The latter asks to enumerate all subgraphs that appear in a given set of attributed graphs “frequently,” where the graph isomorphism is defined by taking into account the items. For the problem, gSpan [16] should be one of the most successful algorithms. The algorithm enumerates all frequent subgraphs by growing up a search tree. In the search tree, a node in a depth d corresponds to a subgraph that consists of d vertices, and a node u is the parent of a node v if the subgraph for v is obtained by adding one vertex to the subgraph for u.

Now we introduce our research problem. We are given an instance (G, I, σ) with a graph $G = (V, E)$, a set I of items, and a function $\sigma : V \to 2^I$. For a subset $X \subseteq V$, let $G[X]$ denote the subgraph induced from G by X, and $I_\sigma(X)$ denote the common item set $\bigcap_{u \in X} \sigma(u)$. A subset $X \subseteq V$ such that $G[X]$ is connected called a *connector*, if for any vertex $v \in V \setminus X$, $G[X \cup \{v\}]$ is not connected or $I_\sigma(X \cup \{v\}) \subsetneq I_\sigma(X)$; i.e., there is no proper superset Y of X such that $G[Y]$ is connected and $I_\sigma(Y) = I_\sigma(X)$.

For the connector enumeration problem, Sese et al. [13] proposed the first algorithm, named COPINE, which explores the search space by utilizing the similar search tree as gSpan. Okuno et al. [11], [12] and Okuno [10] studied parallelization of COPINE. No algorithm with a theoretical time bound had been known until Haraguchi et al. [4], [5] proposed an output-polynomial algorithm, named COOMA. COOMA enumerates connectors in a sequential way with respect to items. First, the algorithm considers a subproblem such that $\{i\}$ is the given item set, where $i \in I$ is chosen arbitrarily. For the subproblem, the algorithm searches for connectors by means of a conventional graph search (e.g., depth-first search). It then goes to the subproblem such that $\{i, i'\}$ is the item set, $i' \in I$, where connector candidates are searched by utilizing the connectors discovered by then. In this way, the subproblems are solved $|I|$ times so that each subproblem is generated by adding an item to the item set of the previous subproblem. Finally we obtain all connectors.

In this paper, we present the first polynomial-delay algorithm for enumerating all connectors. For this, we first extend the problem of enumerating connectors to a general setting so that the connectivity condition on a vertex subset X in G can be specified in a more flexible way. Concretely, we define a family of sets, called a “transitive system,” which is a generalization of the family of all vertex subsets that induce connected subgraphs. The notion of connector is also extended to the transitive system and it will be called a solution. We then design a new algorithm for enumerating all solutions in the transitive system, which leads to

1 Faculty of Commerce, Otaru University of Commerce
2 Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University
a) haraguchi@res.otaru-uc.ac.jp
b) nag@amp.i.kyoto-u.ac.jp

a polynomial-delay algorithm for enumerating all solutions for several connectivity conditions on X in G, such as the biconnectivity of $G[X]$ or the k-edge-connectivity among vertices in X in G.

The paper is organized as follows. In Section 2, we introduce the transitive system, a solution, and two oracles that we require for the transitive system, along with preparation of the notation and the terminology. We explain the structure of the family tree of solutions in Section 3. The proposed algorithm enumerates the solutions by traversing the family tree. The family tree is determined once the parent-child relationship among solutions is defined. We present how we define the parent of a given solution and how to generate its children. Then in Section 4, we provide an algorithm that enumerates all the solutions by traversing the family tree. We also show that, applying the algorithm, all connectors for (G, I, σ) are enumerable in polynomial-delay and in polynomial space. In Section 5, we explain how we deal with various notions of edge- and vertex-connectivity in the enumeration algorithm, followed by concluding remarks in Section 6.

2. Preliminaries

For two integers a and b, let $[a, b]$ denote the set of integers i with $a \le i \le b$. For two subsets $J = \{j_1, j_2, \dots, j_{|J|}\}$ and $K = \{k_1, k_2, \dots, k_{|K|}\}$ of a set A with a total order, where $j_1 < j_2 < \cdots < j_{|J|}$ and $k_1 < k_2 < \cdots < k_{|K|}$, we denote by $J \prec K$ if $J \subsetneq K$ or the sequence $(j_1, j_2, \dots, j_{|J|})$ is lexicographically smaller than the sequence $(k_1, k_2, \dots, k_{|K|})$. We denote $J \preceq K$ if $J \prec K$ or $J = K$.

A system $(V, \mathcal{C})$ consists of a finite set V and a family $\mathcal{C} \subseteq 2^V$, where an element in V is called a *vertex*, and a set in $\mathcal{C}$ is called a *component*. A system $(V, \mathcal{C})$ (or $\mathcal{C}$) is called *transitive* if

any tuple of $Z, X, Y \in \mathcal{C}$ with $Z \subseteq X \cap Y$ implies $X \cup Y \in \mathcal{C}$.

For a subset $X \subseteq V$, a component $Z \in \mathcal{C}$ with $Z \subseteq X$ is called *X-maximal* if no other component $W \in \mathcal{C}$ satisfies $Z \subsetneq W \subseteq X$. Let $\mathcal{C}_{\max}(X)$ denote the family of all X-maximal components.

For example, any Sperner family, a family of subsets every two of which intersect, is a transitive system. Also the family $\mathcal{C}_G$ of vertex subsets $X \in 2^V$ in a graph $G = (V, E)$ such that $G[X]$ is connected is transitive, where $G[X]$ with $|X| = 1$ (resp., $X = \emptyset$) is connected (resp., disconnected).

We define an instance to be a tuple $(V, \mathcal{C}, I, \sigma)$ of a set V of $n \ge 1$ vertices, a family $\mathcal{C} \subseteq 2^V$, a set I of $q \ge 1$ items and a function $\sigma : V \to 2^I$. For each subset $X \subseteq V$, let $I_\sigma(X) \subseteq I$ denote the common item set over $\sigma(v)$, $v \in X$; i.e., $I_\sigma(X) = \bigcap_{v \in X} \sigma(v)$. A *solution* is defined to be a component $X \in \mathcal{C}$ such that

any component $Y \in \mathcal{C}$ with $Y \supsetneq X$ satisfies $I_\sigma(Y) \subsetneq I_\sigma(X)$.

Let $\mathcal{S}$ denote the family of all solutions to the instance. Our aim is to design an algorithm for enumerating all solutions in $\mathcal{S}$ when $\mathcal{C}$ is transitive. When an instance $(V, \mathcal{C}, I, \sigma)$ is given, we assume that $\mathcal{C}$ is implicitly given as two oracles L_1 and L_2 such that

- given non-empty subsets $X \subseteq Y \subseteq V$, $\mathrm{L}_1(X, Y)$ returns a component $Z \in \mathcal{C}_{\max}(Y)$ with $X \subseteq Z$ (or $\emptyset$ if no such Z exists) in $\theta_{1,\mathrm{t}}$ time and $\theta_{1,\mathrm{s}}$ space; and
- given a non-empty subset $Y \subseteq V$, $\mathrm{L}_2(Y)$ returns $\mathcal{C}_{\max}(Y)$ in $\theta_{2,\mathrm{t}}$ time and $\theta_{2,\mathrm{s}}$ space.

We also denote by $\delta(Y)$ an upper bound on $|\mathcal{C}_{\max}(Y)|$, where we assume that δ is a non-decreasing function in the sense that $\delta(X) \le \delta(Y)$ if $X \subseteq Y$. For the example of family $\mathcal{C}_G$ of vertex subsets X such that $G[X]$ is connected in a graph G with n vertices and m edges, we see that $\theta_{i,\mathrm{t}} = O(n + m)$, $i = 1, 2$, $\theta_{i,\mathrm{s}} = O(n + m)$, $i = 1, 2$, and $\delta(Y) = O(|Y|)$.

We show that the time delay of our algorithm is polynomial of $\theta_{1,\mathrm{t}}$, $\theta_{2,\mathrm{t}}$ and $\delta(V)$.

To facilitate our aim, we introduce a total order over the items in I by representing I as a set $[1, q] = \{1, 2, \dots, q\}$ of integers. For each subset $X \subseteq V$, let $\min I_\sigma(X) \in [0, q]$ denote the minimum item in $I_\sigma(X)$, where $\min I_\sigma(X) \triangleq 0$ for $I_\sigma(X) = \emptyset$. For each $i \in [0, q]$, define $\mathcal{S}_i \triangleq \{X \in \mathcal{S} \mid \min I_\sigma(X) = i\}$, where we see that $\mathcal{S}$ is a disjoint union of $\mathcal{S}_i$, $i \in [0, q]$. We design an algorithm that enumerates all solutions in $\mathcal{S}_k$ for any specified $k \in [0, q]$.

We observe an important property on a transitive family of components.

Lemma 1 Let $(V, \mathcal{C})$ be a transitive system. For a component $X \in \mathcal{C}$ and a superset $Y \supseteq X$, there is exactly one component $C \in \mathcal{C}_{\max}(Y)$ that contains X.

Proof: Since $X \subseteq Y$, $\mathcal{C}_{\max}(Y)$ contains a Y-maximal component C that contains X. For any component $W \in \mathcal{C}$ with $X \subseteq W \subseteq Y$, the transitivity of $\mathcal{C}$ and $X \subseteq C \cap W$ imply $C \cup W \in \mathcal{C}$, where $C \cup W = C$ must hold by the Y-maximality of C. Hence C is unique. □

For a component $X \in \mathcal{C}$ and a superset $Y \supseteq X$, we denote by $C(X; Y)$ the component $C \in \mathcal{C}_{\max}(Y)$ that contains X.

3. Defining Family Tree

To generate all solutions in $\mathcal{S}$ efficiently, we use the idea of family tree, where we first introduce a parent-child relationship among solutions, which defines a rooted tree (or a set of rooted trees), and we traverse each tree starting from the root and generating the children of a solution recursively. Our tasks to establish such an enumeration algorithm are as follows:

- Define the roots, called "bases," over all solutions in $\mathcal{S}$;
- Define the "parent" $\pi(S) \in \mathcal{S}$ of each non-base solution $S \in \mathcal{S}$, where S is called a "child" of $T = \pi(S)$;
- Design an algorithm A that, given $S \in \mathcal{S}$, returns $\pi(S)$; and
- Design an algorithm B that, given a solution $T \in \mathcal{S}$, generates a set $\mathcal{X}$ of components $X \in \mathcal{C}$ such that $\mathcal{X}$ contains all children of T. For each component $X \in \mathcal{X}$, we construct $\pi(X)$ by algorithm A to see if X is a child of T (i.e., $\pi(X)$ is equal to T).

Starting from each base, we recursively generate the children of a solution. The complexity of delay-time of the entire algorithm is the time complexity of algorithms A and B, where $|\mathcal{X}|$ is bounded from above by the time complexity of algorithm B.

3.1 Defining Base

Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system. We define $V_{\langle 0 \rangle} \triangleq V$ and $V_{\langle i \rangle} \triangleq \{v \in V \mid i \in \sigma(v)\}$, $i \in I$. For each non-empty subset $J \subseteq I$, define $V_{\langle J \rangle} \triangleq \bigcap_{i \in J} V_{\langle i \rangle}$. For $J = \emptyset$, define

$V_{\langle J\rangle} \triangleq V$. Define

$$\mathcal{B}_i \triangleq \{X \in \mathcal{C}_{\max}(V_{\langle i\rangle}) \mid \min I_\sigma(X) = i\}, \text{ for each } i \in [0, q],$$

and $\mathcal{B} \triangleq \bigcup_{i\in[0,q]} \mathcal{B}_i$. We call a component in $\mathcal{B}$ a *base*.

Lemma 2 Let $(V, C, I = [1, q], \sigma)$ be an instance on a transitive system.

(i) For each non-empty set $J \subseteq [1, q]$ or $J = \{0\}$, it holds that $\mathcal{C}_{\max}(V_{\langle J\rangle}) \subseteq \mathcal{S}$;

(ii) For each $i \in [0, q]$, a solution $S \in \mathcal{S}_i$ is contained in a base in $\mathcal{B}_i$; and

(iii) $\mathcal{S}_0 = \mathcal{B}_0$ and $\mathcal{S}_q = \mathcal{B}_q$.

Proof: (i) Let X be a component in $\mathcal{C}_{\max}(V_{\langle J\rangle})$, where $J \subseteq I_\sigma(X)$. When $J = \{0\}$ (i.e., $V_{\langle J\rangle} = V$), no proper superset of X is a component, and X is a solution. Consider the case of $\emptyset \neq J \subseteq [1, q]$. To derive a contradiction, assume that X is not a solution; i.e., there is a proper superset Y of X such that $I_\sigma(Y) = I_\sigma(X)$. Since $\emptyset \neq J \subseteq I_\sigma(X) = I_\sigma(Y)$, we see that $V_{\langle J\rangle} \supseteq Y$. This, however, contradicts the $V_{\langle J\rangle}$-maximality of X. This proves that X is a solution.

(ii) We prove that each solution $S \in \mathcal{S}_i$ is contained in a base in $\mathcal{B}_i$, where $i = \min I_\sigma(S)$. By Lemma 1, S is a subset of the component $C(S; V_{\langle i\rangle}) \in \mathcal{C}_{\max}(V_{\langle i\rangle})$, where $I_\sigma(S) \supseteq I_\sigma(C(S; V_{\langle i\rangle}))$. Since $i \in I_\sigma(C(S; V_{\langle i\rangle}))$ for $i \geq 1$ (resp., $I_\sigma(C(S; V_{\langle i\rangle})) = \emptyset$ for $i = 0$), we see that $\min I_\sigma(S) = i = \min I_\sigma(C(S; V_{\langle i\rangle}))$. This proves that $C(S; V_{\langle i\rangle})$ is a base in $\mathcal{B}_i$.

(iii) Let $k \in \{0, q\}$. We see from (i) that $\mathcal{C}_{\max}(V_{\langle k\rangle}) \subseteq \mathcal{S}$, which implies that $\mathcal{B}_k = \{X \in \mathcal{C}_{\max}(V_{\langle k\rangle}) \mid \min I_\sigma(X) = k\} \subseteq \{X \in \mathcal{S} \mid \min I_\sigma(X) = k\} = \mathcal{S}_k$. We prove that any solution $S \in \mathcal{S}_k$ is a base in $\mathcal{B}_k$. By (ii), there is a base $X \in \mathcal{B}_k$ such that $S \subseteq X$, which implies that $I_\sigma(S) \supseteq I_\sigma(X)$, $\min I_\sigma(S) \leq \min I_\sigma(X)$. We see that $I_\sigma(S) = I_\sigma(X)$, since $\emptyset = I_\sigma(S) \supseteq I_\sigma(X)$ for $k = 0$, and $q = \min I_\sigma(S) \leq \min I_\sigma(X) \leq q$ for $k = q$. Hence $S \subsetneq X$ would contradict that S is a solution. Therefore $S = X \in \mathcal{B}_k$, as required. □

By Lemma 2(iii), we can find all solutions in $\mathcal{S}_0 \cup \mathcal{S}_q$ by calling oracle $\mathrm{L}_2(Y)$ for $Y = V_{\langle 0\rangle} = V$ and $Y = V_{\langle q\rangle}$. In the following, we consider how to generate all solutions in $\mathcal{S}_k$ with $1 \leq k \leq q-1$.

For a notational convenience, we denote by $C(X; i)$ the component $C(X; V_{\langle i\rangle})$ with $i \in I_\sigma(X)$ and by $C(X; J)$ the component $C(X; V_{\langle J\rangle})$ with $J \subseteq I_\sigma(X)$.

Lemma 3 Let $(V, C, I = [1, q], \sigma)$ be an instance on a transitive system. Let $S, T \in \mathcal{S}$ be solutions such that $S \subseteq T$. It holds that $T = C(S; I_\sigma(T))$.

Proof: Let $T' = C(S; I_\sigma(T)) \in \mathcal{C}_{\max}(V_{\langle I_\sigma(T)\rangle})$, where $S \subseteq T \subseteq V_{\langle I_\sigma(T)\rangle}$. The uniqueness of maximal component $T' = C(S; I_\sigma(T))$ by Lemma 1 indicates $T \subseteq T'$. To derive a contradiction, assume that $T \subsetneq T'$. By Lemma 2(i), $T' \in \mathcal{C}_{\max}(V_{\langle I_\sigma(T)\rangle})$ is a solution. Since T and T' are solutions with $T \subsetneq T'$, it must hold that $I_\sigma(T) \supsetneq I_\sigma(T')$, implying that $V_{\langle I_\sigma(T)\rangle} \not\supseteq T'$, a contradiction. Therefore we have $T = T'$. □

3.2 Defining Parent

This subsection defines the "parent" of a non-base solution. For two solutions $S, T \in \mathcal{S}$, we say that T is a *superset solution* of S if $T \supsetneq S$ and $S, T \in \mathcal{S}_i$ for some $i \in [1, q-1]$. A superset solution T of S is called *minimal* if no proper subset $Z \subsetneq T$ is a superset solution of S. Let S be a non-base solution in $\mathcal{S}_k \setminus \mathcal{B}_k$, $k \in [1, q-1]$. We call a minimal superset solution T of S *the lex-min solution* of S if $I_\sigma(T) \leq I_\sigma(T')$ for all minimal superset solutions T' of S.

Algorithm 1 Parent(S): Finding the lex-min solution of a solution S

Input: An instance $(V, C, I = [1, q], \sigma)$ on a transitive system, an item $k \in [1, q-1]$, and a non-base solution $S \in \mathcal{S}_k \setminus \mathcal{B}_k$, where $k = \min I_\sigma(S)$.
Output: The lex-min solution $T \in \mathcal{S}_k$ of S.

Let $\{k, i_1, i_2, \ldots, i_p\} := I_\sigma(S)$, where $k < i_1 < i_2 < \cdots < i_p$;
$J := \{k\}$; /* $C(S; k) \supsetneq S$ by $S \notin \mathcal{B}_k$ */
for $j = 1, 2, 3, \ldots, p$ **do**
  if $C(S; J \cup \{i_j\}) \neq S$ **then**
    $J := J \cup \{i_j\}$
  end if
end for; /* $J = I_\sigma(T)$ holds */
Return $T := C(S; J)$

Lemma 4 Let $(V, C, I = [1, q], \sigma)$ be an instance on a transitive system. For a non-base solution $S \in \mathcal{S}_k \setminus \mathcal{B}_k$ with $k \in [1, q-1]$, let $I_\sigma(S) = \{k, i_1, i_2, \ldots, i_p\}$, where $k < i_1 < i_2 < \cdots < i_p$, and let T denote the lex-min solution of S.

(i) For an integer $j \in [1, p]$, let $J = I_\sigma(T) \cap \{k, i_1, i_2, \ldots, i_{j-1}\}$. Then $i_j \in I_\sigma(T)$ if and only if $C(S; J \cup \{i_j\}) \supsetneq S$; and

(ii) Given S, algorithm Parent(S) in Algorithm 1 correctly delivers the lex-min solution of S in $O(q(n + \theta_{1,\mathrm{t}}))$ time and $O(q + n + \theta_{1,\mathrm{s}})$ space.

Proof: (i) By Lemma 2(i) and $\min I_\sigma(S) = k$, we see that $C(S; J \cup \{i_j\}) \in \mathcal{S}_k$.

Case 1. $C(S; J \cup \{i_j\}) = S$: For any set $J' \subseteq \{i_{j+1}, i_{j+2}, \ldots, i_p\}$, the component $C(S; J \cup \{i_j\} \cup J')$ is equal to S and cannot be a minimal superset solution of S. This implies that $i_j \notin I_\sigma(T)$.

Case 2. $C(S; J \cup \{i_j\}) \supsetneq S$: Then $C = C(S; J \cup \{i_j\})$ is a solution by Lemma 2(i). Observe that $k \in J \cup \{i_j\} \subseteq I_\sigma(C) \subseteq I_\sigma(S)$ and $\min I_\sigma(C) = k$, implying that $C \in \mathcal{S}_k$ is a superset solution of S. Then C contains a minimal superset solution $T^* \in \mathcal{S}_k$ of S, where $I_\sigma(T^*) \cap [1, i_{j-1}] = I_\sigma(T^*) \cap \{k, i_1, i_2, \ldots, i_{j-1}\} \supseteq J = I_\sigma(T) \cap \{k, i_1, i_2, \ldots, i_{j-1}\} = I_\sigma(T) \cap [1, i_{j-1}]$ and $i_j \in I_\sigma(T^*)$. If $I_\sigma(T^*) \cap [1, i_{j-1}] \supsetneq J$ or $i_j \notin I_\sigma(T)$, then $I_\sigma(T^*) \prec I_\sigma(T)$ would hold, contradicting that T is the lex-min solution of S. Hence $I_\sigma(T) \cap [1, i_{j-1}] = J = I_\sigma(T^*) \cap [1, i_{j-1}]$ and $i_j \in I_\sigma(T)$.

(ii) Based on (i), we can obtain the solution T as follows. First we find the item set $I_\sigma(T)$ by applying (i) to each $j \in [1, p]$, where we construct subsets $J_0 \subseteq J_1 \subseteq \cdots \subseteq J_p \subseteq I_\sigma(S)$ such that $J_0 = \{k\}$ and

$$J_j = \begin{cases} J_{j-1} \cup \{i_j\} & \text{if } C(S; J_{j-1} \cup \{i_j\}) \supsetneq S, \\ J_{j-1} & \text{otherwise.} \end{cases}$$

Each J_j can be obtained from J_{j-1} by testing whether $C(S; J_{j-1} \cup \{i_j\}) \supsetneq S$ holds or not, where $C(S; J_{j-1} \cup \{i_j\})$ is computable by calling the oracle L_1. By (i), we have $J_j = I_\sigma(T) \cap \{k, i_1, \ldots, i_j\}$, and in particular, $J_p = I_\sigma(T)$ holds. Next we compute $C(S; J_p)$ by calling the oracle $\mathrm{L}_1(S, V_{\langle J_p\rangle})$, where $C(S; J_p)$ is equal to the solution T by Lemma 3. The above algorithm is described as algorithm Parent(S) in Algorithm 1.

Let us mention critical parts in terms of time complexity analysis. In line 1, it takes $O(qn)$ time to compute $I_\sigma(S)$. The for-loop from line 3 to 7 is repeated $O(q)$ times. In line 4, the oracle $\mathrm{L}_1(S, V_{\langle J\cup\{i_j\}\rangle})$ is called to obtain a component $Z = C(S; J \cup \{i_j\})$ and whether $S = Z$ or not is tested. This takes $O(\theta_{1,\mathrm{t}} + n)$ time. The overall running time is $O(q(n + \theta_{1,\mathrm{t}}))$. It takes $O(q)$ space to store $I_\sigma(S)$ and J, and $O(n)$ space to store S and Z. An additional $O(\theta_{1,\mathrm{s}})$ space is needed for the oracle L_1. □

For each non-base solution in $\mathcal{S}_k \setminus \mathcal{B}_k$, $k \in [1, q-1]$, the *parent* $\pi(S)$ of S is defined to be the lex-min solution of S. For a solution $T \in \mathcal{S}_k$, each non-base solution $S \in \mathcal{S}_k \setminus \mathcal{B}_k$ such that $\pi(S) = T$ is called a *child* of T.

3.3 Generating Children

This subsection shows how to construct a family $\mathcal{X}$ of components so that all children of a solution T are included in $\mathcal{X}$.

Lemma 5 Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system. For an item $k \in [1, q-1]$, let $T \in \mathcal{S}_k$ be a solution.

(i) For each child $S \in \mathcal{S}_k \setminus \mathcal{B}_k$ of T, it holds that $[k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T)) \neq \emptyset$ and $S \in \mathcal{C}_{\max}(T \cap V_{\langle j\rangle})$ for any $j \in [k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$.

(ii) The set of all children of T can be constructed in $O(q\theta_{2,\mathrm{t}} + q^2(n + \theta_{1,\mathrm{t}})\delta(T))$ time and $O(q + n + \theta_{1,\mathrm{s}} + \theta_{2,\mathrm{s}})$ space.

Proof: (i) Note that $[0, k] \cap I_\sigma(S) = [0, k] \cap I_\sigma(T) = \{k\}$ since $S, T \in \mathcal{S}_k$. Since $S \subsetneq T$ are both solutions, $I_\sigma(S) \supsetneq I_\sigma(T)$. Hence $[k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T)) \neq \emptyset$. Let $j \in [k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$. Since $S \subseteq T \cap V_{\langle j\rangle}$, there is a $(T \cap V_{\langle j\rangle})$-maximal component $C \in \mathcal{C}_{\max}(T \cap V_{\langle j\rangle})$ with $S \subseteq C$, where $S \subseteq C \subseteq T$ and $I_\sigma(S) \supseteq I_\sigma(C) \supseteq I_\sigma(T)$. Then $k = \min I_\sigma(S) = \min I_\sigma(T)$ implies $\min I_\sigma(C) = k$.

We show that $C \in \mathcal{S}$, which implies $C \in \mathcal{S}_k$. Note that $j \in I_\sigma(C) \setminus I_\sigma(T)$, and $C \subsetneq T$. Assume that C is not a solution; i.e., there is a solution $C^* \in \mathcal{S}$ such that $C \subsetneq C^*$ and $I_\sigma(C) = I_\sigma(C^*)$, where $j \in I_\sigma(C) = I_\sigma(C^*)$ means that $C^* \subseteq V_{\langle j\rangle}$. Hence $C^* \setminus T \neq \emptyset$ by the $(T \cap V_{\langle j\rangle})$-maximality of C. Since $C, C^*, T \in \mathcal{C}$ and $C \subseteq C^* \cap T$, we have $C^* \cup T \in \mathcal{C}$ by the transitivity. We also see that $I_\sigma(C^* \cup T) = I_\sigma(C^*) \cap I_\sigma(T) = I_\sigma(C) \cap I_\sigma(T) = I_\sigma(T)$. This, however, contradicts that T is a solution, proving that $C \in \mathcal{S}_k$. If $S \subsetneq C$, then $S \subsetneq C \subsetneq T$ would hold for $S, C, T \in \mathcal{S}_k$, contradicting that T is a minimal superset solution of S. Therefore $S = C$.

(ii) By (i), the union of families $\mathcal{C}_{\max}(T \cap V_{\langle j\rangle})$ with $j \in [k+1, q] \setminus I_\sigma(T)$ contains all children of T. Whether a set S is a child of T or not can be tested by checking if Parent(S) is equal to T or not. However, for two items $j, j' \in [k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$, the same child S can be generated from the different families $\mathcal{C}_{\max}(T \cap V_{\langle j\rangle})$ and $\mathcal{C}_{\max}(T \cap V_{\langle j'\rangle})$. To avoid this, we output a child S of T when $S \in \mathcal{C}_{\max}(T \cap V_{\langle j\rangle})$ for the minimum item j in the item set $[k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$. In other words, we discard any set $S \in \mathcal{C}_{\max}(T \cap V_{\langle j\rangle})$ if j is not the minimum item in $[k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$. An entire algorithm is described in Algorithm 2.

Now we analyze the time and space complexities of the algorithm. Note that T may have no children. The outer for-loop from line 1 to 10 is repeated $O(q)$ times. Computing $\mathcal{C}(T \cap V_{\langle j\rangle})$ in line 2 takes $\theta_{2,\mathrm{t}}$ time by calling the oracle L_2. The inner for-loop from line 3 to 9 is repeated at most $\delta(T \cap V_{\langle j\rangle})$ times for each j, and the most time-consuming part of the inner for-loop is algorithm Parent(S) in line 5, which takes $O(q(n + \theta_{1,\mathrm{t}}))$ time by Lemma 4(ii). Recall that δ is a non-decreasing function. Then the running time of algorithm Children(T, k) is evaluated by

$$O\Big(q\theta_{2,\mathrm{t}} + q(n + \theta_{1,\mathrm{t}}) \sum_{j \in [k+1,q] \setminus I_\sigma(T)} \delta(T \cap V_{\langle j\rangle})\Big)$$
$$= O\Big(q\theta_{2,\mathrm{t}} + q^2(n + \theta_{1,\mathrm{t}})\delta(T)\Big).$$

For the space complexity, we do not need to share the space between iterations of the outer for-loop from line 1 to 10. In each iteration, we use the oracle L_2 and algorithm Parent(S), whose space complexity is $O(q + n + \theta_{1,\mathrm{s}})$ by Lemma 4(ii). Then algorithm Children(T, k) uses $O(q + n + \theta_{1,\mathrm{s}} + \theta_{2,\mathrm{s}})$ space. □

Algorithm 2 Children(T, k): Generating all children

Input: An instance $(V, \mathcal{C}, I, \sigma)$, $k \in [1, q-1]$ and a solution $T \in \mathcal{S}_k$.
Output: All children of T, each of which is output whenever it is generated.

```
1:  for each j ∈ [k+1, q] \ I_σ(T) do
2:      Compute C_max(T ∩ V_⟨j⟩);
3:      for each S ∈ C_max(T ∩ V_⟨j⟩) do
4:          if k = min I_σ(S) and j = min{i | i ∈ [k+1, q] ∩ (I_σ(S) \ I_σ(T))} then
5:              if T = Parent(S) (i.e., S is a child of T) then
6:                  Output S as one of the children of T
7:              end if
8:          end if
9:      end for
10: end for
```

4. Traversing Family Tree

We are ready to describe an entire algorithm for enumerating solutions in $\mathcal{S}_k$ for a given $k \in [0, q]$. We first compute $\mathcal{C}_{\max}(V_{\langle k\rangle})$. We next compute the set $\mathcal{B}_k$ ($\subseteq \mathcal{C}_{\max}(V_{\langle k\rangle})$) of bases by testing whether $k = \min I_\sigma(T)$ or not, where $\mathcal{B}_k \subseteq \mathcal{S}_k$. When $k = 0$ or q, we are done with $\mathcal{B}_k = \mathcal{S}_k$ by Lemma 2(iii). Let $k \in [1, q-1]$. Suppose that we are given a solution $T \in \mathcal{S}_k$, we find all the children of T by Children(T, k) in Algorithm 2. By applying Algorithm 2 to a newly found child recursively, we can find all solutions in $\mathcal{S}_k$.

When no child is found to a given solution $T \in \mathcal{S}_k$, we may need to go up to an ancestor by traversing recursive calls $O(n)$ times before we generate the next solution. This would result in $O(n\alpha)$ time delay, where α denotes the time complexity required for a single run of Children(T, k). To improve the delay to $O(\alpha)$, we employ the *alternative output method* [14], where we output the children of T after (resp., before) generating all descendants when the depth of the recursive call to T is an even (resp., odd) integer.

The entire enumeration algorithm is described in Algorithm 3 and Algorithm 4.

Theorem 1 Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system. For each $k \in [0, q]$, the set $\mathcal{S}_k$ of solutions can be enumerated in $O(q\theta_{2,\mathrm{t}} + q^2(n + \theta_{1,\mathrm{t}})\delta(V_{\langle k\rangle}))$ time delay and in

Algorithm 3 An algorithm to enumerate solutions in $\mathcal{S}_k$ for a given $k \in [0, q]$

Input: An instance $(V, C, I = [1, q], \sigma)$ on a transitive system, and an item $k \in [0, q]$
Output: The set $\mathcal{S}_k$ of solutions to (V, C, I, σ)

Compute $C_{\max}(V_{\langle k \rangle})$; $d := 1$;
for each $T \in C_{\max}(V_{\langle k \rangle})$ **do**
  if $k = \min I_\sigma(T)$ (i.e., $T \in \mathcal{B}_k$) **then**
    Output T;
    if $k \in [1, q-1]$ **then**
      Descendants$(T, k, d + 1)$
    end if
  end if
end for

Algorithm 4 Descendants(T, k, d): Generating all descendants

Input: An instance (V, C, I, σ), $k \in [1, q - 1]$, a solution $T \in \mathcal{S}_k$, and the current depth d of recursive call of Descendants
Output: All descendants of T in $\mathcal{S}_k$

for each $j \in [k + 1, q] \setminus I_\sigma(T)$ **do**
  Compute $C_{\max}(T \cap V_{\langle j \rangle})$;
  for each $S \in C_{\max}(T \cap V_{\langle j \rangle})$ **do**
    if $k = \min I_\sigma(S)$ and $j = \min\{i \mid i \in [k + 1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))\}$ **then**
      if $T =$Parent(S) (i.e., S is a child of T) **then**
        if d is odd **then**
          Output S
        end if;
        Descendants$(S, k, d + 1)$;
        if d is even **then**
          Output S
        end if
      end if
    end if
  end for
end for

$O((q + n + \theta_{1,\mathrm{s}} + \theta_{2,\mathrm{s}})n)$ space.

Proof: First we analyze the time delay. Let α denote the time complexity required for a single run of Children(T, k). By Lemma 5(ii) and $\delta(T) \le \delta(V_{\langle k \rangle})$, we have $\alpha = O(q\theta_{2,\mathrm{t}} + q^2(n + \theta_{1,\mathrm{t}})\delta(V_{\langle k \rangle}))$. Hence we see that the time complexity of Algorithm 3 and Descendants without including recursive calls is $O(\alpha)$.

From Algorithm 3 and Descendants, we observe:
(i) When d is odd, the solution S for any call Descendants$(S, k, d + 1)$ is output
immediately before Descendants$(S, k, d + 1)$ is executed; and
(ii) When d is even, the solution S for any call Descendants$(S, k, d + 1)$ is output
immediately after Descendants$(S, k, d + 1)$ is executed.

Let m denote the number of all calls of Descendants during a whole execution of Algorithm 3. Let $d_1 = 1, d_2, \ldots, d_m$ denote the sequence of depths d in each Descendants$(S, k, d + 1)$ of the m calls. Note that $d = d_i$ satisfies (i) when d_{i+1} is odd and $d_{i+1} = d_i + 1$, whereas $d = d_i$ satisfies (ii) when d_{i+1} is even and $d_{i+1} = d_i - 1$. Therefore we easily see that during three consecutive calls with depth d_i, d_{i+1} and d_{i+2}, at least one solution will be output. This implies that the time delay for outputting a solution is $O(\alpha)$.

We analyze the space complexity. Observe that the number of calls Descendants whose executions are not finished during an execution of Algorithm 3 is the depth d of the current call Descendants$(S, k, d + 1)$. In Algorithm 4, $|T| + d \le n + 1$ holds initially, and Descendants$(S, k, d + 1)$ is called for a nonempty subset $S \subsetneq T$, where $|S| < |T|$. Hence $|S| + d \le n + 1$ holds when Descendants$(S, k, d + 1)$ is called. Then Algorithm 3 can be implemented to run in $O(n\beta)$ space, where β denotes the space required for a single run of Children(T, k). We have $\beta = O(q + n + \theta_{1,\mathrm{s}} + \theta_{2,\mathrm{s}})$ by Lemma 5(ii). Then the overall space complexity is $O((q + n + \theta_{1,\mathrm{s}} + \theta_{2,\mathrm{s}})n)$. □

Theorem 1 yields a polynomial-delay algorithm for the connector enumeration problem as follows.

Theorem 2 Given an instance $(G = (V, E), I, \sigma)$, we can enumerate all connectors in $O(q^2(n + m)n)$ time delay and in $O((q + n + m)n)$ space, where $n = |V|$, $m = |E|$ and $q = |I|$.

Proof: Recall that C_G denotes the family of vertex subsets $X \in 2^V$ such that $G[X]$ is connected. A connector induces a connected subgraph, and thus is an element in C_G. By the definition of solution, an element in C_G is a connector iff it is a solution. Hence, the connector enumeration problem for (G, I, σ) is solved by enumerating all solutions for the instance (V, C_G, I, σ).

For the transitive system (V, C_G), we see that $\theta_{i,\mathrm{t}} = O(n + m)$, $i = 1, 2$, $\theta_{i,\mathrm{s}} = O(n + m)$, $i = 1, 2$, and $\delta(Y) = O(|Y|) = O(n)$. By Theorem 1, we can enumerate all solutions in $\mathcal{S}$ in $O(q^2(n + m)n)$ time delay and in $O((q + n + m)n)$ space. □

5. Transitive System Based on Mixed Graphs

In addition to (V, C_G), we may obtain an alternative transitive system by selecting a different notion of connectivity such as the edge- or vertex-connectivity on a digraph or undirected graph. To treat those systems universally, this section presents a general method of constructing a transitive system based on a mixed graph and a weight function on elements in the graph.

In Section 5.1, we introduce the notions of mixed graph, meta-weight function and k-connectivity that is defined on them. We show that they altogether determine a transitive system. Then in Section 5.2, we present how to construct a meta-weight function from given mixed graph M and weight function w on elements in M. We also explain how to construct two oracles L_1 and L_2 for given M and w, by which we can run the enumeration algorithm in Section 4 for the corresponding transitive system. In Section 5.3, as case studies, we observe how to apply the enumeration algorithm to transitive systems that are determined by k-edge- and k-vertex-connectivity.

We omit the proofs of theorems and lemmas in this section, due to space limitation.

5.1 Mixed Graph and Meta-weight Function

Let $\mathbb{R}_+$ denote the set of non-negative reals. For a function $f : A \to \mathbb{R}_+$ and a subset $B \subseteq A$, we let $f(B)$ denote $\sum_{a \in B} f(a)$.

Let M be a *mixed graph*, which is defined to be a graph that may contain undirected edges and directed edges. In this paper,

M may have multiple edges but no self-loops. Let $V(M)$, $\vec{E}(M)$ and $\overline{E}(M)$ denote the sets of vertices, directed edges and undirected edges, respectively. Let $n = |V(M)|$ and $m = |E(M)|$. Let $E(M) \triangleq \vec{E}(M) \cup \overline{E}(M)$. For two vertices $u, v \in V(M)$, let
$\vec{E}(u, v)$ denote the set of directed edges from u to v,
$\overline{E}(u, v)$ denote the set of undirected edges between u and v in M, and
$E(u, v) \triangleq \vec{E}(u, v) \cup \overline{E}(u, v)$.
For two non-empty subsets $X, Y \subseteq V(M)$, let
$\vec{E}(X; Y) \triangleq \bigcup_{u \in X, v \in Y} \vec{E}(u, v)$,
$\overline{E}(X; Y) \triangleq \bigcup_{u \in X, v \in Y} \overline{E}(u, v)$ and
$E(X; Y) \triangleq \bigcup_{u \in X, v \in Y} E(u, v)$.
For two vertices $s, t \in V(M)$, an *s, t-cut* C is defined to be an ordered pair (S, T) of disjoint subsets $S, T \subseteq V(M)$ such that $s \in S$ and $t \in T$, and the element set $\varepsilon(C)$ of C ($\varepsilon(S, T)$ of (S, T)) is defined to be a union $F \cup R$ of the edge subset $F = E(S, T)$ and the vertex subset $R = V(M) \setminus (S \cup T)$, where $R = \emptyset$ is allowed.

We define a *meta-weight function* on M to be $\omega : 2^V \times (V(M) \cup E(M)) \to \mathbb{R}_+$. For each subset $X \in 2^V$, we denote $w(X, a), a \in V(M) \cup E(M)$ as a function $\omega_X : V(M) \cup E(M) \to \mathbb{R}_+$ such that $\omega_X(a) = \omega(X, a)$ for each $a \in V(M) \cup E(M)$. We call ω *monotone* if for any subsets $X \subseteq Y \subseteq V$, the next holds:

$$\omega_Y(a) \geq \omega_X(a) \text{ for any } a \in V(M) \cup E(M).$$

For two vertices $s, t \in V(M)$ and a subset $X \subseteq V(M)$, define $\mu(s, t; X) \triangleq \min\{\omega_X(\varepsilon(C)) \mid s, t\text{-cuts } C = (S, T) \text{ in } M\}$. We call a vertex subset $X \subseteq V(M)$ *k-connected* if $|X| = 1$ or $\mu(u, v; X) \geq k$ for each pair of vertices $u, v \in X$.

Lemma 6 Let (M, ω) be a mixed graph with a monotone meta-weight function, and $k \geq 0$. For two k-connected subsets $X, Y \subseteq V(M)$ such that $\omega_{X \cap Y}(X \cap Y) \geq k$, the subset $X \cup Y$ is k-connected.

For a mixed graph (M, ω) with a meta-weight function and a real $k \geq 0$, let $\mathcal{C}(M, \omega, k) \subseteq 2^{V(M)}$ denote the family of k-connected subsets $X \subseteq V$ with $\omega_X(X) \geq k$.

Lemma 7 For a mixed graph (M, ω) with a monotone meta-weight function a real $k \geq 0$, let $\mathcal{C} = \mathcal{C}(M, \omega, k)$. Then $\mathcal{C}$ is transitive.

5.2 Construction of Monotone Meta-weight Functions

This subsection shows a concrete method of constructing a monotone meta-weight function from a mixed graph with a standard weight function on the vertex and edge sets. We also present how to construct oracles L_1 and L_2 that are required when we apply the enumeration algorithm in Section 4 to the corresponding transitive system.

Let M be a mixed graph and $w : V(M) \cup E(M) \to \mathbb{R}_+$ be a weight function. We define a *coefficient function* to be $\gamma = (\alpha, \alpha^-, \alpha^+, \beta)$ that consists of functions

$$\alpha : \overline{E}(M) \to \mathbb{R}_+,$$
$$\alpha^+, \alpha^- : \vec{E}(M) \to \mathbb{R}_+, \text{ and}$$
$$\beta : V(M) \cup E(M) \to \mathbb{R}_+.$$

We call γ *monotone* if $1 \geq \alpha(e) \geq \beta(e)$ for each undirected edge $e \in \overline{E}(M)$, $1 \geq \alpha^+(e) \geq \beta(e)$, $1 \geq \alpha^-(e) \geq \beta(e)$ for each directed edge $e \in \vec{E}(M)$; and $1 \geq \beta(v)$ for each vertex $v \in V(M)$. We call a tuple (M, w, γ) a *system*, and define a meta-weight function $\omega : 2^V \times (V(M) \cup E(M)) \to \mathbb{R}_+$ to the system so that, for each subset $X \subseteq V(M)$, $\omega_X : V(M) \cup E(M) \to \mathbb{R}_+$ is given by

$$\omega_X(v) = \begin{cases} w(v) & \text{if } v \in X, \\ \beta(v)w(v) & \text{if } v \in V(M) \setminus X, \end{cases}$$

$$\omega_X(e) = \begin{cases} w(e) & \text{if } e \in E(X, X), \\ \alpha(e)w(e) & \text{if } e \in \overline{E}(X, V(M) \setminus X), \\ \alpha^+(e)w(e) & \text{if } e \in \vec{E}(X, V(M) \setminus X), \\ \alpha^-(e)w(e) & \text{if } e \in \vec{E}(V(M) \setminus X, X), \\ \beta(e)w(e) & \text{if } e \in E(V \setminus X, V \setminus X). \end{cases}$$

We call a system (M, w, γ) *monotone* if γ is monotone.

Lemma 8 For a monotone system (M, w, γ), the corresponding meta-weight function $\omega : 2^V \times (V(M) \cup E(M)) \to \mathbb{R}_+$ is monotone.

For a system (M, w, γ) on a mixed graph M with n vertices and m edges and a real $k \geq 0$, let $\tau(n, m, k)$ and $\sigma(n, m, k)$ denote the time and space complexities for testing if $\mu(u, v; X) < k$ holds or not for two vertices $u, v \in V(M)$ and a subset $X \subseteq V(M)$.

Lemma 9 For a monotone tuple (M, w, γ), let ω be the corresponding monotone meta-weight function.
(i) $\tau(n, m, k) = O(mn \log n)$ and $\sigma(n, m, k) = O(n + m)$; and
(ii) Let $X \subseteq Y \subseteq V(M)$ be non-empty subsets such that $\omega_X(X) \geq k$ and $\mu(u, u'; Y) \geq k$ for all vertices $u, u' \in X$. Given a vertex $t \in Y \setminus X$, whether there is a vertex $u \in X$ such that $\mu(u, t; Y) < k$ or not can be tested in $\tau(n, m, k)$ time and $\sigma(n, m, k)$ space.

We denote by $\mathcal{C}(M, w, \gamma, k)$ the family of k-connected sets in a system (M, w, γ). We consider how to construct oracles L_1 and L_2 to the system. For two non-empty subsets $X \subseteq Y \subseteq V(M)$, let $\mathcal{C}_{\max}(Y)$ denote the family of maximal subsets $X \in \mathcal{C}(M, w, \gamma, k)$ such that $X \subseteq Y$, and let $C_k(X; Y)$ denote a maximal set $X^* \in \mathcal{C}_{\max}(Y)$ such that $X \subseteq X^*$; and $C_k(X; Y) \triangleq \emptyset$ if no such set X^* exists.

Lemma 10 For a monotone system (M, w, γ), let ω denote the corresponding monotone meta-weight function. Let $X \subseteq Y \subseteq V(M)$ be non-empty subsets such that $\omega_X(X) \geq k$. Then
(i) $C_k(X; Y)$ is uniquely determined;
(ii) If there are vertices $u \in X$ and $v \in Y$ such that $\mu(u, v; Y) < k$, then $v \notin X^*$;
(iii) Assume that $\mu(u, v; Y) \geq k$ for all vertices $u \in X$ and $v \in Y \setminus X$. Then $C_k(X; Y) = Y$ if $\mu(u, u'; Y) \geq k$ for all vertices $u, u' \in X$; and $C_k(X; Y) = \emptyset$ otherwise; and
(iv) Finding $C_k(X; Y)$ can be done in $O(|Y|^2 \tau(n, m, k))$ time and $O(\sigma(n, m, k) + |Y|)$ space.

By the lemma, oracle $\mathrm{L}_1(X; Y)$ to a monotone system (M, w, γ) runs in $\theta_{1,\mathrm{t}} = O(|Y|^2 \tau(n, m, k))$ time and $\theta_{1,\mathrm{s}} = O(\sigma(n, m, k) + |Y|)$ space.

For a system (M, w, γ), we define a *k-core* of a subset $Y \subseteq V(M)$ to be a subset Z of Y such that $\omega_Z(Z) \geq k$ and any proper subset Z' of Z satisfies $\omega_{Z'}(Z') < k$.

Lemma 11 Let (M, w, γ) be a monotone system, and Y be a subset of $V(M)$. For the family $\mathcal{K}$ of all k-cores of Y, it holds that $\mathcal{C}_{\max}(Y) = \bigcup_{Z \in \mathcal{K}} \{C_k(Z; Y)\}$ and $|\mathcal{C}_{\max}(Y)| \leq |\mathcal{K}|$. Given $\mathcal{K}$,

$C_{\max}(Y)$ can be obtained in $O(|\mathcal{K}|(|Y|^2\tau(n,m,k)+|Y|\log|\mathcal{K}|))$ time and $O(\sigma(n,m,k)+|\mathcal{K}|\cdot|Y|)$ space.

By the lemma, oracle $\mathrm{L}_2(Y)$ to a monotone system (M,w,γ) runs in $\theta_{2,\mathrm{t}} = O(|\mathcal{K}|(|Y|^2\tau(n,m,k)+|Y|\log|\mathcal{K}|))$ time and $\theta_{2,\mathrm{s}} = O(\sigma(n,m,k)+|\mathcal{K}|\cdot|Y|)$ space, where we assume that the family $\mathcal{K}$ of k-cores of Y is given as input.

5.3 Edge- and Vertex-Connectivity in Digraph and Graph

Let G be an unweighted digraph or undirected graph with n vertices and m edges. Let $s,t\in V(G)$ be two vertices in G. Let $\lambda(s,t;G)$ denote the minimum size $|F|$ of a subset $F\subseteq E(G)$ so that the graph $G-F$ obtained from G by removing edges in F has no directed (resp., undirected) path from s to t. Let $\kappa(s,t;G)$ denote the minimum size $|S|$ of a subset $S\subseteq E(G)\cup(V(G)\setminus\{s,t\})$ to be removed from G so that the graph $G-S$ obtained from G by removing vertices and edges in S has no directed (resp., undirected) path from s to t, where such a minimum subset S can be chosen so that $S\setminus E(\{s\},\{t\})\subseteq V(G)$. By Menger's theorem [8], $\lambda(s,t;G)$ (resp., $\kappa(s,t;G)$) is equal to the maximum number of edge-disjoint (resp., internally disjoint) paths from s to t. We can test whether $\lambda(s,t;G)\geq k$ (resp., $\kappa(s,t;G)\geq k$) or not in $O(\min\{k,n\}m)$ (resp., $O(\min\{k,n^{1/2}\}m)$) time [1], [2]. A graph G is called *k-edge-connected* if $|V(G)|\geq 1$ and $\lambda(u,v;G)\geq k$ for any two vertices $u,v\in X$. A graph G is called *k-vertex-connected* if $|V(G)|\geq k+1$ and $\kappa(u,v;G)\geq k$ for any two vertices $u,v\in X$. In the following, we show two examples of transitive systems based on graph connectivity.

5.3.1 Connected Set in the Entire Graph

Given a digraph or graph G, we define "k-connected set" based on the connectivity of the entire graph G. Let us call a subset $X\subseteq V(G)$ *k-edge-connected* if $|X|=1$ or for any two vertices $u,v\in X$, $\lambda(u,v;G)\geq k$. Let $\mathcal{C}_{k,\mathrm{edge}}$ denote the family of k-edge-connected sets in G. Let us call a subset $X\subseteq V(G)$ *k-vertex-connected* if $|X|\geq k$ or for any two vertices $u,v\in X$, $\kappa(u,v;G)\geq k$. Let $\mathcal{C}_{k,\mathrm{vertex}}$ denote the family of k-vertex-connected sets in G.

Lemma 12 Let G be a digraph or undirected graph and $k\geq 0$ be an integer. Then:

(i) The family $\mathcal{C}=\mathcal{C}_{k,\mathrm{edge}}$ is transitive. For each non-empty subset $Y\subseteq V(G)$, it holds $|\mathcal{C}_{\max}(Y)|\leq|Y|$, oracles $\mathrm{L}_1(X;Y)$ and $\mathrm{L}_2(Y)$ run in $O(n^2)$ time and space after an $O(n^2\min\{k,n\}m)$-time and $O(n^2)$-space preprocessing; and

(ii) The family $\mathcal{C}=\mathcal{C}_{k,\mathrm{vertex}}$ is transitive. For each non-empty subset $Y\subseteq V(G)$, it holds $|\mathcal{C}_{\max}(Y)|\leq\binom{|Y|}{k}$, oracle $\mathrm{L}_1(X;Y)$ runs in $O(n^2)$ time and $O(n^2)$ space, and oracle $\mathrm{L}_2(Y)$ runs in $O(|Y|^kn^2)$ time and $O(|Y|^kn)$ space, after an $O(n^2\min\{k,n^{1/2}\}m)$-time and $O(n^2)$-space preprocessing.

Using Theorem 1 and Lemma 12, we have the following theorem on the time delay and the space complexity of enumeration of connectors that are k-edge-connected or k-vertex-connected.

Theorem 3 Let (G,I,σ) be an instance and $k\geq 0$ be an integer, where $G=(V,E)$ is either a digraph or an undirected graph, $n=|V|$, $m=|E|$, and $q=|I|$.

(i) We can enumerate all connectors that are k-edge-connected in $O(q^2n^3)$ time delay and in $O(qn+n^3)$ space, after an $O(n^2\min\{k,n\}m)$-time and $O(n^2)$-space preprocessing.

(ii) We can enumerate all connectors that are k-vertex-connected in $O(q^2n^{k+2})$ time delay and in $O(qn+n^{k+2})$ space, after an $O(n^2\min\{k,n^{1/2}\}m)$-time and $O(n^2)$-space preprocessing.

5.3.2 Connected Set in Induced Graph

Given a digraph or graph G, we define a "k-connected set" X based on the connectivity of the induced graph $G[X]$. Now consider the family $\mathcal{C}^{\mathrm{in}}_{k,\mathrm{edge}}$ (resp., $\mathcal{C}^{\mathrm{in}}_{k,\mathrm{vertex}}$) of subsets $X\in V(G)$ such that the induced graph $G[X]$ is k-edge-connected (resp., k-vertex-connected).

Lemma 13 Let G be a digraph or undirected graph and $k\geq 0$ be an integer. Then:

(i) The family $\mathcal{C}=\mathcal{C}^{\mathrm{in}}_{k,\mathrm{edge}}$ is transitive. For each non-empty subset $Y\subseteq V(G)$, it holds $|\mathcal{C}_{\max}(Y)|\leq|Y|$, oracle $\mathrm{L}_1(X;Y)$ runs in $O(|Y|^2(n^2+\min\{k,n\}m))$ time and $O(n^2)$ space, and $\mathrm{L}_2(Y)$ runs in $O(|Y|^3(n^2+\min\{k,n\}m))$ time and $O(n^2)$ space; and

(ii) The family $\mathcal{C}=\mathcal{C}^{\mathrm{in}}_{k,\mathrm{vertex}}$ is transitive. For each non-empty subset $Y\subseteq V(G)$, it holds $|\mathcal{C}_{\max}(Y)|\leq\binom{|Y|}{k}$, oracle oracle $\mathrm{L}_1(X;Y)$ runs in $O(|Y|^2(n^2+\min\{k,n^{1/2}\}m))$ time and $O(n^2)$ space, and oracle $\mathrm{L}_2(Y)$ runs in $O(|Y|^{k+2}(n^2+\min\{k,n^{1/2}\}m))$ time and $O(|Y|^kn)$ space.

Again, using Theorem 1 and Lemma 12, we have the following theorem on the time delay and the space complexity of enumeration of connectors such that the induced subgraphs are k-edge-connected or k-vertex-connected.

Theorem 4 Let (G,I,σ) be an instance and $k\geq 0$ be an integer, where $G=(V,E)$ is either a digraph or an undirected graph, $n=|V|$, $m=|E|$, and $q=|I|$.

(i) We can enumerate all connectors such that the induced subgraphs are k-edge-connected in $O(q^2n^3(n^2+\min\{k,n\}m))$ time delay and in $O(qn+n^3)$ space.

(ii) We can enumerate all connectors such that the induced subgraphs are k-vertex-connected in $O(q^2n^{k+2}(n^2+\min\{k,n^{1/2}\}m))$ time delay and in $O(qn+n^{k+2})$ space.

6. Concluding Remarks

In this paper, we have considered the connector enumeration problem in a general setting. We treated the problem on what we call a transitive system and proposed an algorithm for enumerating all solutions in the system (Algorithms 3 and 4 in Section 4). The algorithm requires two oracles L_1 and L_2, and the time delay is $O(q\theta_{2,\mathrm{t}}+q^2(n+\theta_{1,\mathrm{t}})\delta(V_{\langle k\rangle}))$, whereas the space complexity is $O((q+n+\theta_{1,\mathrm{s}}+\theta_{2,\mathrm{s}})n)$, as we stated in Theorem 1. As a consequence of the theorem, we have complexity results on enumerating connectors that satisfy several connectivity conditions. We summarize the results in Table 1. For future work, we investigate the possibility of improvement of the complexities for respective cases.

References

[1] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B.: *Optimization*, Handbooks in Management Science and Operations Research, Vol. 1, chapter Network Flows (IV), pp. 211–369, North-Holland (1989).

[2] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B.: *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ (1993).

Table 1 Complexity of enumerating connectors X that satisfy several connectivity conditions

Theorem	Condition	Delay	Space
2	$G[X]$ is connected	$O(q^2(n+m)n)$	$O((q+n+m)n)$
3(i)	X is k-edge-connected	$O(q^2n^3)$ (preprocessing is required)	$O(qn+n^3)$
3(ii)	X is k-vertex-connected	$O(q^2n^{k+2})$ (preprocessing is required)	$O(qn+n^{k+2})$
4(i)	$G[X]$ is k-edge-connected	$O(q^2n^3(n^2+\min\{k,n\}m))$	$O(qn+n^3)$
4(ii)	$G[X]$ is k-vertex-connected	$O(q^2n^{k+2}(n^2+\min\{k,n^{1/2}\}m))$	$O(qn+n^{k+2})$

[3] Avis, D. and Fukuda, K.: Reverse search for enumeration, *Discrete Applied Mathematics*, Vol. 65, No. 1, pp. 21–46 (1996).
[4] Haraguchi, K., Momoi, Y., Sherbevski, A. and Nagamochi, H.: COOMA: A Components Overlaid Mining Algorithm for Enumerating Connected Subgraphs with Common Itemsets, Technical Report 002, Department of Applied Mathematics and Physics, Kyoto University (2018).
[5] Haraguchi, K., Momoi, Y., Sherbevski, A. and Nagamochi, H.: COOMA: A Components Overlaid Mining Algorithm for Enumerating Connected Subgraphs with Common Itemsets, *Proceedings of 2nd International Workshop on Enumeration Problems and Applications (WEPA 2018)* (2018).
[6] Inokuchi, A., Washio, T. and Motoda, H.: An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, *Principles of Data Mining and Knowledge Discovery* (Zighed, D. A., Komorowski, J. and Żytkow, J., eds.), pp. 13–23 (online), DOI: 10.1007/3-540-45372-5_2.
[7] Li, Y., Sha, C., Huang, X. and Zhang, Y.: Community Detection in Attributed Graphs: An Embedding Approach, *Proceedings of AAAI-18*, (online), available from ⟨https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17142⟩ (2018).
[8] Menger, K.: Zur allegemeinen Kurventheorie, *Fundamenta Mathematicae*, Vol. 10, pp. 96–115 (1927).
[9] Nutov, Z.: Listing minimal edge-covers of intersecting families with applications to connectivity problems, *Discrete Applied Mathamatics*, Vol. 157, No. 1, pp. 112–117 (online), DOI: 10.1016/j.dam.2008.04.026 (2009).
[10] Okuno, S.: Parallelization of Graph Mining using Backtrack Search Algorithm, PhD Thesis, Kyoto University (2017).
[11] Okuno, S., Hiraishi, T., Nakashima, H., Yasugi, M. and Sese, J.: Reducing Redundant Search in Parallel Graph Mining Using Exceptions, *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 328–337 (online), available from ⟨https://doi.org/10.1109/IPDPSW.2016.136⟩ (2016).
[12] Okuno, S., Hiraishi, T., Nakashima, H., Yasugi, M. and Sese, J.: Parallelization of Extracting Connected Subgraphs with Common Itemsets, *Information and Media Technologies*, Vol. 9, No. 3, pp. 233–250 (online), available from ⟨https://doi.org/10.11185/imt.9.233⟩ (2014).
[13] Sese, J., Seki, M. and Fukuzaki, M.: Mining Networks with Shared Items, *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10)*, pp. 1681–1684 (2010).
[14] Uno, T.: Two general methods to reduce delay and change of enumeration algorithms, Technical Report NII-2003-004E, NII (2003).
[15] Wasa, K.: Enumeration of Enumeration Algorithms, *CoRR*, Vol. abs/1605.05102 (online), available from ⟨http://arxiv.org/abs/1605.05102⟩ (2016).
[16] Yan, X. and Han, J.: gSpan: Graph-Based Substructure Pattern Mining, *Proceedings of 2002 IEEE International Conference on Data Mining (ICDM '02)*, pp. 721–724 (2002).