COOMA: A Components Overlaid Mining Algorithm for Enumerating Connected Subgraphs with Common Itemsets

Kazuya Haraguchi Yusuke Momoi Aleksandar Shurbevski
Hiroshi Nagamochi

Abstract

In the present paper, we consider a graph mining problem of enumerating what we call connectors. Suppose that we are given a data set (G, I, σ) that consists of a graph $G = (V, E)$, an item set I, and a function $\sigma : V \to 2^I$. For $X \subseteq V$, we define $A_\sigma(X) \triangleq \bigcap_{v \in X} \sigma(v)$. Note that, for $X, Y \subseteq V$, $X \subseteq Y$ implies that $A_\sigma(X) \supseteq A_\sigma(Y)$. A vertex subset X is called a *connector* if (i) the subgraph $G[X]$ induced from G by X is connected; and (ii) for any $v \in V \setminus X$, $G[X \cup \{v\}]$ is disconnected or $A_\sigma(X \cup \{v\}) \subsetneq A_\sigma(X)$. To enumerate all connectors, we propose a novel algorithm named *COOMA* (*components overlaid mining algorithm*). Interestingly, COOMA is a total-polynomial time algorithm, i.e., the running time is polynomially bounded with respect to the input and output size. We show the efficiency of COOMA in comparison with COPINE [Sese et al., 2010], a depth-first-search based algorithm.

1 Introduction

Many existing data are stored in the form of a graph [5]. In graph data, a vertex is often associated with a set of items or attributes. For example, in a social network, each vertex corresponds to a user and two users are joined by an edge if they are friends. A user may be associated with products that he or she has purchased so far. In a genetic network, each vertex may correspond to an SNP (single nucleotide polymorphism), and two SNPs are joined by an edge if they have significant relationship in some context. An SNP may be associated with patients who possess it [18].

We consider a graph mining problem as follows. Suppose that we are given a tuple (G, I, σ) of a graph $G = (V, E)$, an item set $I = \{i_1, \ldots, i_q\}$, and a function $\sigma : V \to 2^I$. For each vertex $v \in V$, the subset $\sigma(v)$ represents the set of items with which v is associated. For $X \subseteq V$, we denote by $A_\sigma(X)$ the set of items common to $\sigma(v)$ for all vertices $v \in X$, i.e., $A_\sigma(X) \triangleq \bigcap_{v \in X} \sigma(v)$. For $X, Y \subseteq V$, $X \subseteq Y$ implies that $A_\sigma(X) \supseteq A_\sigma(Y)$. A vertex subset X is called a *connector* if the following conditions hold:

(i) the subgraph $G[X]$ induced from G by X is connected; and

[1]Technical report 2018-002, December 27, 2018

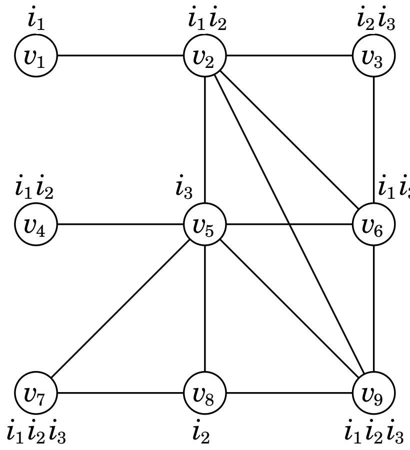

Figure 1: An instance (G, I, σ)

Table 1: All connectors X and their item sets $A_\sigma(X)$ of the instance given by Figure 1

X	$A_\sigma(X)$
$\{v_1, \ldots, v_9\}$	$\emptyset$
$\{v_1, v_2, v_6, v_9\}$	$\{i_1\}$
$\{v_2, v_3, v_7, v_8, v_9\}$	$\{i_2\}$
$\{v_3, v_5, v_6, v_7, v_9\}$	$\{i_3\}$
$\{v_4\}$	$\{i_1, i_2\}$
$\{v_2, v_9\}$	$\{i_1, i_2\}$
$\{v_6, v_9\}$	$\{i_1, i_3\}$
$\{v_3\}$	$\{i_2, i_3\}$
$\{v_7\}$	$\{i_1, i_2, i_3\}$
$\{v_9\}$	$\{i_1, i_2, i_3\}$

(ii) adding any vertex $v \in V \setminus X$ to X loses the connectivity of the subgraph or decreases the common item set, i.e., $G[X \cup \{v\}]$ is disconnected or $A_\sigma(X \cup \{v\}) \subsetneq A_\sigma(X)$.

We illustrate an instance (G, I, σ) in Figure 1. For this instance, we show in Table 1 all connectors, along with their item sets. A connector X is *nontrivial* if $A_\sigma(X) \neq \emptyset$, and it is *trivial* otherwise. When X is a trivial connector, X is a vertex set of a connected component of G.

In the context of social networks, a nontrivial connector X may represent a maximal subset of users such that any two of them are connected by a sequence of individuals in the set who are pairwise friends, and that all of them have purchased the products in $A_\sigma(X)$. It should be meaningful to obtain connectors in terms of marketing. For example, we may recommend a product i in $A_\sigma(X)$ to a user u who is not in X but has a friend in X, expecting that u may like i and thus buy it.

We consider the problem of enumerating all connectors for a given instance (G, I, σ). This problem was first introduced for biological networks and an algorithm named COPINE was proposed [16, 17]. Recently, Okuno [11] and Okuno et al. [13, 12] studied parallelization of COPINE.

We claim that there should be room for exploring better algorithms. COPINE is a straightforward algorithm in some sense. Based on gSpan [21], the algorithm traverses a search tree in a depth-first manner. For enumeration problems, however, several algorithmic frameworks have been invented so far; e.g., reverse search [1], BDD/ZDD [9], and dynamic programming [2]. These frameworks have been applied to various enumeration problems [4, 10, 20]. COPINE is not the only algorithmic solution to our problem. We may develop other enumeration algorithms, aiming at a better graph mining tool for practitioners.

With this in mind, we propose a novel enumeration algorithm named COOMA, which stands for a components overlaid mining algorithm. The highlight of COOMA is that the running time is total-polynomial, i.e., polynomially bounded with respect to the input and output size.

The paper is organized as follows. In Section 2, we introduce notation and terminologies and provide essential properties of connectors. In Section 3, we propose the algorithm COOMA and its extended version ExtCOOMA, along with time complexity analyses. In Section 4, we discuss a generalization of the connector enumeration algorithm and how COOMA and COPINE work for the generalized problem. In Section 5, we make empirical comparison of the three algorithms, COOMA, ExtCOOMA and COPINE, in terms of computation time and memory consumption. Finally we give concluding remark in Section 6.

2 Preliminaries

2.1 Graphs

In the present paper, a graph stands for a simple undirected graph. The vertex set (resp., edge set) of a graph H is denoted by $V(H)$ (resp., $E(H)$).

Let $G = (V, E)$ be a graph with a vertex set V and an edge set E. For a vertex $v \in V$, let $N_G(v)$ denote the set $\{u \in V : uv \in E\}$ of neighbors of v in G. The *degree* of v is defined to be $|N_G(v)|$, and we denote by Δ the maximum degree over V, i.e., $\Delta \triangleq \max_{v \in V} |N_G(v)|$. Let X be a subset of V, and F be a subset of E. Define $X[F]$ to be the set of vertices $x \in X$ such that x is an end-vertex of an edge in F, $F[X]$ to be the set of edges $e = uv \in F$ with $u, v \in X$, and $G[X]$ (resp., $G[X, F]$) to be the subgraph $(X, E[X])$ (resp., $(X[F], F[X])$). A vertex subset Z of a graph H is called a *component* of H if $H[Z]$ is connected and $H[Z \cup \{v\}]$ is not connected for any vertex $v \in V(H) \setminus Z$. Let $\mathcal{C}(X)$ (resp., $\mathcal{C}(X, F)$) denote the family of all components of the graph $G[X]$ (resp., $G[X, F]$).

For the example in Figure 1, let us take $X = \{v_1, v_2, v_3, v_6, v_9\}$. Then $E[X]$, the edge set of $G[X]$, is $\{v_1v_2, v_2v_3, v_2v_6, v_2v_9, v_3v_6, v_6v_9\}$. For an edge set $F = \{v_2v_6, v_2v_9, v_5v_9\}$, we have $X[F] = \{v_2, v_6, v_9\}$ and $F[X] = \{v_2v_6, v_2v_9\}$. The subgraph $G[X, F]$ has just one component.

2.2 Connectors

Assume that we are given an instance (G, I, σ) that consists of a graph $G = (V, E)$, an item set $I = \{i_1, \ldots, i_q\}$ and a function $\sigma : V \to 2^I$, where $q = |I|$ denotes the total number of items.

We consider the problem of enumerating all connectors for the given instance. It is easy to enumerate trivial connectors; we only have to compute connected components of G by a conventional graph search algorithm (e.g., depth-first search) and to output those whose common item sets are empty. Hereafter we concentrate only on nontrivial connectors. We denote by $\mathcal{M}$ the family of all nontrivial connectors for the given instance. The problem is summarized as the CE (connector enumeration) problem as follows.

Problem CE

Input: An instance (G, I, σ) that consists of a graph $G = (V, E)$, an item set I and a function $\sigma : V \to 2^I$.

Output: The family $\mathcal{M}$ of nontrivial connectors for (G, I, σ).

For an item $i \in I$, we define $V_{\langle i \rangle}$ as the set of vertices that have the item i, and $E_{\langle i \rangle}$ as the set of edges such that both of the endpoints have the item i;

$$V_{\langle i \rangle} \triangleq \{v \in V : i \in \sigma(v)\}, \quad E_{\langle i \rangle} \triangleq \{uv \in E : i \in \sigma(u) \cap \sigma(v)\}.$$

For $\mathcal{M}' \subseteq \mathcal{M}$, we represent by $\|\mathcal{M}'\|$ the sum of $|X|$ over $X \in \mathcal{M}'$.

We present three lemmas that describe essential properties of connectors.

Lemma 1 *Given an instance (G, I, σ), let $i \in I$ be an item. Then any connected component in the subgraph $G[V_{\langle i \rangle}]$ is a connector.*

Proof: Let X be a connected component of $G[V_{\langle i \rangle}]$. For each vertex $v \in V \setminus X$, if $i \in \sigma(v)$, then $G[X \cup \{v\}]$ is not connected from the definition of a connected component. If $i \notin \sigma(v)$, then we have $i \in A_\sigma(X) \setminus \sigma(v)$ and thus $A_\sigma(X \cup \{v\}) \subsetneq A_\sigma(X)$. $\square$

We call a connected component in $G[V_{\langle i \rangle}]$ a *base connector*. Let $\mathcal{B}$ denote the union of all base connectors, i.e., $\mathcal{B} = \bigcup_{i \in I} \mathcal{C}(V_{\langle i \rangle})$. In Figure 1,

$$\mathcal{C}(V_{\langle i_1 \rangle}) = \{\{v_1, v_2, v_6, v_9\}, \{v_4\}, \{v_7\}\},$$
$$\mathcal{C}(V_{\langle i_2 \rangle}) = \{\{v_2, v_3, v_7, v_8, v_9\}, \{v_4\}\}, \quad \mathcal{C}(V_{\langle i_3 \rangle}) = \{\{v_3, v_5, v_6, v_7, v_9\}\},$$

and $\mathcal{B}$ is the union of these three families.

Lemma 2 *Let $X_1, X_2 \in \mathcal{M}$ be two nontrivial connectors for a given instance (G, I, σ). Then it holds that $\mathcal{C}(X_1 \cap X_2) \subseteq \mathcal{M}$.*

Proof: Let Y be a set in $\mathcal{C}(X_1 \cap X_2, E)$. The subgraph $G[Y]$ is connected, but $G[Y \cup \{v\}]$ is not connected for any vertex $v \in (X_1 \cap X_2) \setminus Y$. Let v be a vertex such that $v \in V \setminus (X_1 \cap X_2)$. It suffices to show that $G[Y \cup \{v\}]$ is not connected or $A_\sigma(Y) \setminus \sigma(v) \neq \emptyset$. Since $X_i \in \mathcal{M}$, $i = 1, 2$, $G[X_i \cup \{v\}]$ is not connected or $A_\sigma(X_i) \setminus \sigma(v) \neq \emptyset$. Hence we see that $G[Y \cup \{v\}]$ is also not connected or $A_\sigma(Y) \setminus \sigma(v) \supseteq A_\sigma(X_i) \setminus \sigma(v) \neq \emptyset$ for $i = 1$ or 2, as required. $\square$

Lemma 3 *Given an instance (G, I, σ), let $Y \in \mathcal{M} \setminus \mathcal{B}$ be a non-base connector, $i \in A_\sigma(Y)$ be an item that belongs to the common item set $A_\sigma(Y)$, and $C \in \mathcal{C}(V_{\langle i \rangle})$ be a base connector. If $Y \subseteq C$, then there exists a connector $X \in \mathcal{M}$ such that $Y \subsetneq X$ and $Y \in \mathcal{C}(X \cap C)$.*

PROOF: Because $Y \subseteq C$, it holds that $A_\sigma(Y) \supseteq A_\sigma(C)$. If $A_\sigma(Y) = A_\sigma(C)$, then $Y = C \in \mathcal{B}$ would hold, which contradicts $Y \notin \mathcal{B}$. Then we have $A_\sigma(Y) \supsetneq A_\sigma(C)$ and there is an item $j \in A_\sigma(Y) \setminus A_\sigma(C)$. There is a base connector $C' \in \mathcal{C}(V_{\langle j \rangle})$ such that $Y \subsetneq C'$. Moreover, Y is contained in a component of the graph $G[C' \cap C]$. This means that $\mathcal{M}$ contains a connector X with $X \supsetneq Y$ such that Y is contained in a component of the graph $G[X \cap C]$. We choose X as a minimal subset among all such connectors. Let Z denote the component of the graph $G[X \cap C]$ that contains Y, where $Z \in \mathcal{M}$ by Lemma 2. If $Z \neq Y$, then $Y \in G[Z \cap C]$, contradicting the choice of X. Hence $Z = Y$ and the connector X satisfies the lemma. $\square$

Definition 1 *Let $\mathcal{M}' \subseteq \mathcal{M}$ and $\mathcal{B}' \subseteq \mathcal{B}$. We call $\mathcal{M}'$* self-contained with respect to $\mathcal{B}'$ *if* **(a)** *$\mathcal{B}' \subseteq \mathcal{M}'$, and* **(b)** *for every $(X, C) \in \mathcal{M}' \times \mathcal{B}'$, it holds that $\mathcal{C}(X \cap C) \subseteq \mathcal{M}'$.*

For the instance in Figure 1, $\mathcal{M}' = \{\{v_1, v_2, v_6, v_9\}, \{v_3, v_5, v_6, v_7, v_9\}, \{v_6, v_9\}\}$ is self-contained with respect to $\mathcal{C}(V_{\langle i_1 \rangle}) \cup \mathcal{C}(V_{\langle i_3 \rangle})$. On the other hand, $\mathcal{M}' = \{\{v_2, v_3, v_7, v_8, v_9\}, \{v_6, v_9\}\}$ is not self-contained with respect to any $I' \subseteq I$ because the intersection $\{v_9\}$ of the two sets in $\mathcal{M}'$ is not in $\mathcal{M}'$.

Lemma 4 *Given an instance (G, I, σ), let $\mathcal{M}' \subseteq \mathcal{M}$ be a subfamily of connectors. If $\mathcal{M}'$ is self-contained with respect to $\mathcal{B}$, then $\mathcal{M}' = \mathcal{M}$.*

PROOF: From the definition of self-containment, $\mathcal{M}'$ contains the whole set $\mathcal{B}$ of base connectors. Because $\mathcal{M}' \subseteq \mathcal{M}$, we show that the equality holds. To derive a contradiction, assume that there is a set $Y \in \mathcal{M} \setminus \mathcal{M}'$, where we choose Y as a maximal subset among all such connectors. Let $i \in A_\sigma(Y)$ be an item and denote by C the component in $\mathcal{C}(V_{\langle i \rangle})$ that contains Y. It holds that $C \supsetneq Y$ since $Y \notin \mathcal{M}' \supseteq \mathcal{B} \supseteq \mathcal{C}(V_{\langle i \rangle})$. By Lemma 3, there is a connector $X \in \mathcal{M}$ with $X \supsetneq Y$ such that $Y \in \mathcal{C}(X \cap C)$. Because Y is a maximal subset in $\mathcal{M} \setminus \mathcal{M}'$, we have $X \in \mathcal{M}'$. This, however, means $Y \in \mathcal{C}(X \cap C, E) \setminus \mathcal{M}'$, contradicting that $\mathcal{M}'$ is self-contained with respect to $\mathcal{B}$. $\square$

3 Two Algorithms for the Connector Enumeration Problem

In this section, we propose two algorithms for the CE problem. The first is COOMA, which is presented in Section 3.1. The other is EXTCOOMA, an extension of COOMA, which we present in Section 3.2. The time complexities of both algorithms are polynomially bounded with respect to the input and output size.

3.1 Algorithm COOMA

Overview. The following lemma suggests the direction of the algorithm COOMA.

Lemma 5 *Given an instance (G, I, σ), let $\mathcal{M}' \subseteq \mathcal{M}$, $I' \subsetneq I$, and $i \in I \setminus I'$. We denote $\mathcal{B}' = \bigcup_{i' \in I'} \mathcal{C}(V_{\langle i' \rangle})$. If $\mathcal{M}'$ is self-contained with respect to $\mathcal{B}'$, then $\mathcal{N} = \mathcal{M}' \cup \mathcal{M}'' \cup \mathcal{C}(V_{\langle i \rangle})$ is self-contained with respect to $\mathcal{B}' \cup \mathcal{C}(V_{\langle i \rangle})$, where $\mathcal{M}''$ is defined as*

$$\mathcal{M}'' = \bigcup_{X \in \mathcal{M}'} \mathcal{C}(X \cap V_{\langle i \rangle}). \tag{1}$$

Proof: Observe that $\mathcal{C}(V_{\langle i \rangle}) \subseteq \mathcal{N}$ holds, and that, for every $i' \in I'$, $\mathcal{C}(V_{\langle i' \rangle}) \subseteq \mathcal{M}' \subseteq \mathcal{N}$ holds. We show that $\mathcal{C}(X \cap C) \subseteq \mathcal{N}$ holds for every pair $(X, C) \in \mathcal{N} \times (\mathcal{B}' \cup \mathcal{C}(V_{\langle i \rangle}))$. We observe the following four cases:

(i) $X \in \mathcal{M}'$ and $C \in \mathcal{B}'$;

(ii) $X \in \mathcal{M}'$ and $C \in \mathcal{C}(V_{\langle i \rangle})$;

(iii) $X \in \mathcal{M}'' \cup \mathcal{C}(V_{\langle i \rangle})$ and $C \in \mathcal{C}(V_{\langle i \rangle})$; and

(iv) $X \in \mathcal{M}'' \cup \mathcal{C}(V_{\langle i \rangle})$ and $C \in \mathcal{B}'$.

(i) The inclusion $\mathcal{C}(X \cap C) \subseteq \mathcal{M}' \subseteq \mathcal{N}$ holds by the assumption that $\mathcal{M}'$ is self-contained with respect to $\mathcal{B}'$. **(ii)** From the definition of $\mathcal{M}''$, that is (1), we have $\mathcal{C}(X \cap C) \subseteq \mathcal{C}(X \cap V_{\langle i \rangle}) \subseteq \mathcal{M}'' \subseteq \mathcal{N}$. **(iii)** Recall that $\mathcal{C}(V_{\langle i \rangle})$ is a collection of connected components. Because $i \in A_\sigma(X)$, there is only one component $C_X \in \mathcal{C}(V_{\langle i \rangle})$ with $C_X \supseteq X$. If $C = C_X$, then we have $\mathcal{C}(X \cap C) = \{C_X\} \subseteq \mathcal{C}(V_{\langle i \rangle}) \subseteq \mathcal{N}$. Otherwise, we have $X \cap C = \emptyset$. **(iv)** If $X \in \mathcal{C}(V_{\langle i \rangle})$, then the discussion is reduced to the case (ii), by interchanging X and C. Otherwise, there are a base connector $C_X \in \mathcal{C}(V_{\langle i \rangle})$ with $C_X \supseteq X$ and a connector $Y \in \mathcal{M}'$ such that $X \in \mathcal{C}(Y \cap C_X)$. We have $\mathcal{C}(Y \cap C) \subseteq \mathcal{M}'$, and also have $\mathcal{C}(Y \cap C \cap C_X) \subseteq \mathcal{M}''$ by (ii). Then it holds that

$$\mathcal{N} \supseteq \mathcal{M}'' \supseteq \mathcal{C}(Y \cap C \cap C_X) = \mathcal{C}(Y \cap C_X \cap C) = \bigcup_{X' \in \mathcal{C}(Y \cap C_X)} \mathcal{C}(X' \cap C) \supseteq \mathcal{C}(X \cap C).$$

□

Using Lemma 5, we can enumerate all the nontrivial connectors in $\mathcal{M}$ as follows. First, we compute $\mathcal{C}(V_i)$ for all $i \in I$ by using a conventional graph search. We choose an arbitrary item $i_1 \in I$, and let $\mathcal{M}' \leftarrow \mathcal{C}(V_{i_1})$ and $I' \leftarrow \{i_1\}$. Obviously, this $\mathcal{M}'$ is self-contained with respect to $\bigcup_{i' \in I'} \mathcal{C}(V_{\langle i' \rangle}) = \mathcal{C}(V_{i_1})$. Then we enlarge the family $\mathcal{M}'$ so that $\mathcal{M}'$ is self-contained with respect to $\bigcup_{i' \in I' \cup \{i\}} \mathcal{C}(V_{\langle i' \rangle})$, where the item i is arbitrarily chosen from $I \setminus I'$. Specifically, we compute the family $\mathcal{M}''$ of (1) and append $\mathcal{M}''$ and $\mathcal{C}(V_{\langle i \rangle})$ to $\mathcal{M}'$. The obtained $\mathcal{M}'$ is self-contained with respect to $\bigcup_{i' \in I' \cup \{i\}} \mathcal{C}(V_{\langle i' \rangle})$ by Lemma 5. Updating $I' \leftarrow I' \cup \{i\}$, we repeat this process as long as $I' \subsetneq I$. Finally, when $I' = I$, $\mathcal{M}'$ is self-contained with respect to $\mathcal{B}$. By Lemma 4, this $\mathcal{M}'$ is equivalent to $\mathcal{M}$.

The algorithm is summarized as COOMA in Algorithm 1. In the description, any set union is taken without creating duplication.

Theorem 1 *Given an instance (G, I, σ), the algorithm COOMA (Algorithm 1) outputs the family $\mathcal{M}$ correctly.*

Implementation. We store the graph $G = (V, E)$ by the conventional adjacency list, and the item set $\sigma(v)$ of a vertex $v \in V$ by a q-dimensional binary vector ($q = |I|$).

The algorithm retains the family $\mathcal{M}'$ of generated connectors. We realize $\mathcal{M}'$ by making use of a *radix tree* (a.k.a., *patricia trie*) [14, 6, 15]. Originally, radix tree is a data structure that is used to retain a set of strings, where a string is a sequence of characters. We assign integral ids to the vertices and use a radix tree to retain connectors, regarding a vertex id as a character, and a connector as a string.

Algorithm 1 COOMA

Input: An instance (G, I, σ)

Output: The set $\mathcal{M}$ of nontrivial connectors of (G, I, σ)

1: Compute $\mathcal{C}(V_{\langle i \rangle})$ for $i \in I$;
2: Choose an item $i_1 \in I$;
3: $\mathcal{M}' \leftarrow \mathcal{C}(V_{\langle i_1 \rangle})$;
4: $I' \leftarrow \{i_1\}$;
5: **while** $I' \subsetneq I$ **do**
6:     $\mathcal{M}'' \leftarrow \emptyset$;
7:     Choose an item $i \in I \setminus I'$;
8:     **for each** $X \in \mathcal{M}'$ **do**
9:         Compute $\mathcal{C}(X \cap V_{\langle i \rangle})$;
10:         $\mathcal{M}'' \leftarrow \mathcal{M}'' \cup \mathcal{C}(X \cap V_{\langle i \rangle})$
11:     **end for**;
12:     $\mathcal{M}' \leftarrow \mathcal{M}' \cup \mathcal{M}'' \cup \mathcal{C}(V_{\langle i \rangle})$;
13:     $I' \leftarrow I' \cup \{i\}$
14: **end while**;
15: Output $\mathcal{M}'$ as $\mathcal{M}$

Let $b_{\max}$ be the maximum size of a base connector, i.e., $b_{\max} = \max\{|B| : i \in I,\ B \in \mathcal{C}(V_{\langle i \rangle})\}$. Note that any connector X satisfies $|X| \le b_{\max}$. We realize $\mathcal{M}'$ by a set of $b_{\max}$ radix trees, denoted by $R_1, \ldots, R_{b_{\max}}$. Each R_b is used to retain connectors of size b.

A radix tree is represented by a rooted tree, and each leaf corresponds to a connector. During the algorithm, every time we obtain a connector X with $b = |X|$, we need to decide whether X belongs to R_b or not, and insert X to R_b if not. The operation can be done in $O(b)$ time. We add the following mechanisms to the radix trees.

- We connect all leaves in R_b by means of a linked list.
- When we insert X to R_b as a new connector, we label the new leaf with the item as follows;

  (line 3) The leaf is labeled with the item i_1.

  (lines 10 and 12) The leaf is labeled with i, the item chosen in the current iteration.

  On the other hand, when X already belongs to R_b, there exists a leaf that corresponds to X. In this case, we overwrite the leaf label in the above way.

Time complexity. Next we analyze the time complexity of COOMA. The following operations have nontrivial time complexity analyses:

(a) Scan all connectors $X \in \mathcal{M}'$ (line 8).

(b) Search components of $\mathcal{C}(X \cap V_{\langle i \rangle})$ (line 9).

(c) Add each connector $Y \in \mathcal{C}(X \cap V_{\langle i \rangle})$ to $\mathcal{M}''$ without creating duplication (line 10).

(d) Append $\mathcal{M}''$ and $\mathcal{C}(V_{\langle i \rangle})$ to $\mathcal{M}'$ (line 12).

For (a), all leaves of the radix trees can be scanned in $O(|\mathcal{M}'|)$ time because we connect the leaves of each R_b by a linked list.

We maintain $\mathcal{M}''$ implicitly in the radix trees, making use of the leaf label. Recall that a leaf of a radix tree corresponds to a connector, and that the leaf label retains the item of the latest iteration in which the connector is generated by the graph search. Hence, while we scan $X \in \mathcal{M}'$ in line 8, if the leaf for X is labeled with the current item i, we ignore it because it is not a member of $\mathcal{M}' \setminus \mathcal{M}''$.

The search in (b) can be done in $O(\Delta|X|)$ time as follows; When the currrent item i is chosen at line 7, we construct the subgraph $G_i \triangleq (V_{\langle i \rangle}, E)$. For each $X \in \mathcal{M}'$, we extract connected components in $(X \cap V_{\langle i \rangle}, E)$ by executing a restricted graph search on G_i such that only $v \in X$ and the edges incident to v are searched.

For (c), we decide whether each connector $Y \in \mathcal{C}(X \cap V_{\langle i \rangle})$ belongs to the radix tree $R_{b'}$ $(b' = |Y|)$ or not. If yes, we overwrite the leaf label as i, and otherwise, we insert Y to $R_{b'}$ and label the leaf with i. This can be done in $O(|X|)$ time.

For (d), we already store all the connectors of $\mathcal{M}''$ in the radix trees. We only have to add each $Y \in \mathcal{C}(V_{\langle i \rangle})$ to the radix trees, which can be done in the same way as (c). This takes $O(\|\mathcal{C}(V_{\langle i \rangle})\|)$ time.

Theorem 2 *The running time of algorithm COOMA (Algorithm 1) is $O(\Delta|I|\,\|\mathcal{M}\|)$.*

Proof: An iteration of the while-loop from line 5 to 14 takes $O(\Delta\,\|\mathcal{M}'\|)$ time, where $\mathcal{M}'$ is the set upon completion of line 12. This is repeated $|I|$ times and clearly we have $\|\mathcal{M}'\| \leq \|\mathcal{M}\|$. □

3.2 Algorithm ExtCOOMA

In this subsection, we consider an extension of COOMA. For $\mathcal{B}_1, \ldots, \mathcal{B}_r \subseteq \mathcal{B}$, the collection $\mathbb{C} = \{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$ is called a *cover of* $\mathcal{B}$ if $\bigcup_{p=1}^{r} \mathcal{B}_p = \mathcal{B}$.

Definition 2 *Given an instance (G, I, σ), let $\mathbb{C}$ be a cover of $\mathcal{B}$. We call $\mathbb{C}$ a* base cover *of $\mathcal{B}$ if, for every $\mathcal{B}_p \in \mathbb{C}$, any two base connectors $X, Y \in \mathcal{B}_p$ $(X \neq Y)$ satisfy $X \cap Y = \emptyset$.*

Obviously, $\mathbb{C}_I = \{\mathcal{C}(V_{\langle i_1 \rangle}), \ldots, \mathcal{C}(V_{\langle i_q \rangle})\}$ is a base cover of $\mathcal{B}$. The following lemma is a generalization of Lemma 5, suggesting us to use a "good" base cover instead of $\mathbb{C}_I$.

Lemma 6 *Given an instance (G, I, σ), let $\mathcal{M}' \subseteq \mathcal{M}$, $\mathbb{C} = \{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$ be a base cover of $\mathcal{B}$, $\mathbb{C}' \subsetneq \mathbb{C}$, and $\mathcal{B}_p \in \mathbb{C} \setminus \mathbb{C}'$. We denote $\mathcal{B}' = \bigcup_{\mathcal{B}_{p'} \in \mathbb{C}'} \mathcal{B}_{p'}$, $V_p = \bigcup_{C \in \mathcal{B}_p} C$ and $E_p = \bigcup_{C \in \mathcal{B}_p} E[C]$. If $\mathcal{M}'$ is self-contained with respect to $\mathcal{B}'$, then $\mathcal{N} = \mathcal{M}' \cup \mathcal{M}'' \cup \mathcal{B}_p$ is self-contained with respect to $\mathcal{B}' \cup \mathcal{B}_p$, where $\mathcal{M}''$ is defined as;*

$$\mathcal{M}'' = \bigcup_{X \in \mathcal{M}'} \{\mathcal{C}(X \cap C) : C \in \mathcal{C}(V_p, E_p)\}. \tag{2}$$

Note that each component in the subgraph $G[V_p, E_p]$ is a base connector in $\mathcal{B}_p$ that consists of more than one vertex. In (2), the set in the right-hand side represents the set of connectors that are obtained by "overlaying" X on $G[V_p, E_p]$.

Proof: Observe that $\mathcal{B}' \cup \mathcal{B}_p \subseteq \mathcal{N}$ holds. We show that $\mathcal{C}(X \cap C) \subseteq \mathcal{N}$ holds for every pair $(X, C) \in \mathcal{N} \times (\mathcal{B}' \cup \mathcal{B}_p)$. We observe the following four cases:

(i) $X \in \mathcal{M}'$ and $C \in \mathcal{B}'$;

(ii) $X \in \mathcal{M}'$ and $C \in \mathcal{B}_p$;

(iii) $X \in \mathcal{M}'' \cup \mathcal{B}_p$ and $C \in \mathcal{B}_p$; and

(iv) $X \in \mathcal{M}'' \cup \mathcal{B}_p$ and $C \in \mathcal{B}'$.

(i) The inclusion $\mathcal{C}(X \cap C) \subseteq \mathcal{M}' \subseteq \mathcal{N}$ holds by the assumption that $\mathcal{M}'$ is self-contained with respect to $\mathcal{B}'$. **(ii)** From (2), we have $\mathcal{C}(X \cap C) \subseteq \mathcal{M}'' \subseteq \mathcal{N}$. **(iii)** Because $\mathcal{C}(V_p, E_p)$ is a collection of connected components in $G[V_p, E_p]$, there is only one base connector $C_X \in \mathcal{B}_p$ such that $C_X \supseteq X$. If $C = C_X$, then we have $\mathcal{C}(X \cap C) = \{C_X\} \subseteq \mathcal{B}_p \subseteq \mathcal{N}$. Otherwise, we have $X \cap C = \emptyset$. **(iv)** If $X \in \mathcal{B}_p$, then the discussion is reduced to the case (ii), by interchanging X and C. Otherwise (i.e., if $X \in \mathcal{M}'' \setminus \mathcal{B}_p$), there are a base connector $C_X \in \mathcal{B}_p$ with $C_X \supseteq X$ and a connector $Y \in \mathcal{M}'$ such that $X \in \mathcal{C}(Y \cap C_X)$. We have $\mathcal{C}(Y \cap C) \subseteq \mathcal{M}'$, and also have $\mathcal{C}(Y \cap C \cap C_X) \subseteq \mathcal{M}''$ by (ii). Then it holds that

$$\mathcal{N} \supseteq \mathcal{M}'' \supseteq \mathcal{C}(Y \cap C \cap C_X) = \mathcal{C}(Y \cap C_X \cap C) = \bigcup_{X' \in \mathcal{C}(Y \cap C_X)} \mathcal{C}(X' \cap C) \supseteq \mathcal{C}(X \cap C).$$

□

Lemma 5 is a special case of Lemma 6 such that $\mathbb{C}_I$ is given as the input base cover $\mathbb{C}$.

As we did for Lemma 5, we can enumerate all the nontrivial connectors by using Lemma 6. First, we determine a base cover $\mathbb{C} = \{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$ of $\mathcal{B}$ somehow. We will discuss how to determine $\mathbb{C}$ later. We let $\mathcal{M}' \leftarrow \mathcal{B}_1$. Obviously, this $\mathcal{M}'$ is self-contained with respect to $\mathcal{B}_1$. Then for $p = 2, \ldots, r$, we enlarge $\mathcal{M}'$ so that it is self-contained with respect to $\bigcup_{p'=1}^{p} \mathcal{B}_{p'}$. That is, we compute the family $\mathcal{M}''$ of (2), and append $\mathcal{M}''$ and $\mathcal{B}_p$ to $\mathcal{M}'$. The obtained $\mathcal{M}'$ is self-contained with respect to $\bigcup_{p'=1}^{p} \mathcal{B}_{p'}$ by Lemma 6. Upon completion of the iteration, we have $\mathcal{M}'$ that is self-contained with respect to $\bigcup_{p=1}^{r} \mathcal{B}_p = \mathcal{B}$; it is equivalent to $\mathcal{M}$, by Lemma 4.

We summarize this algorithm as ExtCOOMA (an extended version of COOMA) in Algorithm 2. Recall that the running time of COOMA is $O(\Delta q \,\|\mathcal{M}\|)$, where $q = |I|$ is the number of iterations. Similarly, we can bound the running time of ExtCOOMA by using $r = |\mathbb{C}|$.

Theorem 3 *The algorithm* ExtCOOMA (*Algorithm* 2) *outputs the family* $\mathcal{M}$ *correctly in* $O(\Delta r \,\|\mathcal{M}\|)$ *time.*

How to determine $\mathbb{C}$. Because the time complexity of ExtCOOMA is $O(\Delta r \,\|\mathcal{M}\|)$, where $r = |\mathbb{C}|$, it is natural to consider constructing as small a base cover $\mathbb{C}$ as possible. Unfortunately, it is NP-hard to obtain a smallest $\mathbb{C}$.

Theorem 4 *Given a set* $\mathcal{B}$ *of base connectors, it is NP-hard to construct a smallest base cover of* $\mathcal{B}$.

Proof: The proof is given by reduction from the *vertex coloring problem*, a well-known NP-hard problem. For a graph $G = (V, E)$, a vertex subset $S \subseteq V$ is an *independent set* if no

Algorithm 2 ExtCOOMA

Input: An instance (G, I, σ) with a set $\mathcal{B}$ of base connectors and a base cover $\mathbb{C} = \{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$ of $\mathcal{B}$
Output: The set $\mathcal{M}$ of nontrivial connectors

1: $\mathcal{M}' \leftarrow \mathcal{B}_1$;
2: **for each** $p \in \{2, \ldots, r\}$ **do**
3: $\quad \mathcal{M}'' \leftarrow \emptyset$;
4: $\quad V_p \leftarrow \bigcup_{C \in \mathcal{B}_p} C$;
5: $\quad E_p \leftarrow \bigcup_{C \in \mathcal{B}_p} E[C]$;
6: $\quad$ **for each** $X \in \mathcal{M}'$ **do**
7: $\quad\quad$ Compute $\{\mathcal{C}(X \cap C) : C \in \mathcal{C}(V_p, E_p)\}$;
8: $\quad\quad \mathcal{M}'' \leftarrow \mathcal{M}'' \cup \{\mathcal{C}(X \cap C) : C \in \mathcal{C}(V_p, E_p)\}$
9: $\quad$ **end for**;
10: $\quad \mathcal{M}' \leftarrow \mathcal{M}' \cup \mathcal{M}'' \cup \mathcal{B}_p$
11: **end for**;
12: Output $\mathcal{M}'$ as $\mathcal{M}$

two vertices in S are adjacent. For an integer k, G is *k-colorable* if the vertex set V can be partitioned into k independent sets. Given a graph G and an integer k, it is NP-complete to decide whether G is k-colorable [3]. The vertex coloring problem asks for the smallest k such that G is k-colorable.

The reduction is given as follows; for each $v \in V$, construct a set $B_v = \{e \in E : e \text{ is incident to } v\}$. Let us define $\mathcal{B} = \bigcup_{v \in V} \{B_v\}$. Observe that $B_v \cap B_u = \emptyset$ holds iff v and u are not adjacent. Then one sees that there is a base cover $\mathbb{C}$ with $|\mathbb{C}| = k$ iff G is k-colorable. □

To obtain a small $\mathbb{C}$, we could apply heuristic algorithms that are invented for the vertex coloring problem [8].

Here, we propose constructing $\mathbb{C}$ based on another idea, motivated by our preliminary experiments. Let $\mathbb{C} = \{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$. See Algorithm 2. For integers p, p' such that $1 \leq p < p' \leq r$, base connectors in $\mathcal{B}_p$ are taken as X in line 6 more frequently than those in $\mathcal{B}_{p'}$. Because the graph search in line 7 takes $O(\Delta|X|)$ time, we desire that base connectors in $\mathcal{B}_p$ are small.

Based on this observation, to construct $\mathcal{B}_1$, we include as many base connectors as possible so that the base connectors are mutually disjoint. Because this is the set packing problem, a well-known NP-hard problem [3], we employ a minimum-cardinality greedy method. The subsequent $\mathcal{B}_p$, $p = 2, 3, \ldots$, are constructed by applying the greedy method to the remaining base connectors, and we are done when all base connectors are included in the collection. In the experiments in Section 5.2, we will show the effectiveness of this method that constructs $\mathbb{C}$.

4 Discussion

In this section, we discuss three topics concerning the CE problem and our algorithms: a generalization of the CE problem (Section 4.1), problem reduction (Section 4.2), and comparison of COOMA with COPINE, an existing algorithm, in terms of how they behave in a search tree (Section 4.3).

4.1 Generalization of the CE Problem

The number of connectors is exponentially large in general, but most of them could be ignored or useless in some applications. In the context of social networks, the cardinality $|X|$ of a connector X represents how many users belong to the connector, and $|A_\sigma(X)|$ represents how many items users in X have in common. A practitioner may like to focus on connectors that have enough values for these two measures. Let θ_V and θ_I be positive integers. Using these as thresholds on the connector size and the size of the common item set, respectively, we define the subset $\mathcal{M}(\theta_V, \theta_I)$ of connectors to be;

$$\mathcal{M}(\theta_V, \theta_I) = \{X \in \mathcal{M} : |X| \geq \theta_V,\ |A_\sigma(X)| \geq \theta_I\}.$$

We summarize the GenCE (generalized CE) problem as follows.

Problem GenCE

Input: An instance (G, I, σ) that consists of a graph $G = (V, E)$, an item set I and a function $\sigma : V \to 2^I$, and thresholds $\theta_V, \theta_I \in \mathbb{Z}_+$.

Output: The family $\mathcal{M}(\theta_V, \theta_I)$ of nontrivial connectors for (G, I, σ).

Obviously, the CE problem is a special case of the GenCE problem such that $\theta_V = \theta_I = 1$.

4.2 Problem Reduction

The GenCE problem is solved by enumerating all the connectors in $\mathcal{M}$, and then by dropping X from $\mathcal{M}$ such that $X \notin \mathcal{M}(\theta_V, \theta_I)$. To perform the enumeration efficiently, we may reduce the given instance by preprocessing. Here we introduce some of such techniques.

Reduction 1 *If a base connector $X \in \mathcal{C}(V_{\langle i \rangle})$ of an item i satisfies $|X| < \theta_V$, then we can drop the item i from any vertex $v \in X$ (i.e., $\sigma(v) \leftarrow \sigma(v) \setminus \{i\}$).*

This is possible because, for any subset $X' \subseteq X$, $|X'| < |X| < \theta_V$ holds.

Reduction 2 *Any vertex $v \in V$ with $|\sigma(v)| < \theta_I$ can be removed from G.*

This is possible because $\mathcal{M}(\theta_V, \theta_I)$ remains unchanged after v is removed from G. Analogously, we can remove an edge uv with $|A_\sigma(\{u, v\})| < \theta_I$.

Reduction 3 *Any edge $uv \in E$ with $|A_\sigma(\{u, v\})| < \theta_I$ can be removed from G.*

For any edge uv with $\sigma(u) = \sigma(v)$, it holds that $|X \cap \{u, v\}| = 0$ or 2 for any connector X. This leads to the following reduction.

Reduction 4 *We can contract any edge $uv \in E$ with $\sigma(u) = \sigma(v)$ to obtain a smaller graph.*

Note that Reduction 4 can be applied to a leaf edge $uv \in E$ with $\sigma(u) = \sigma(v)$.

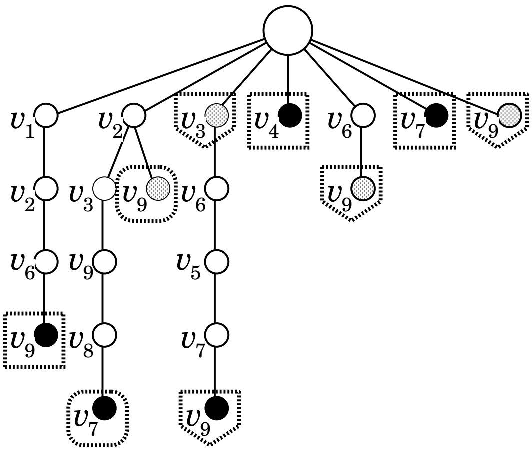

Figure 2: The search tree of COPINE for the example of Figure 1

4.3 Behavior in Search Tree

An existing algorithm COPINE traverses a search tree in a depth-first manner. In Figure 2, we show the search tree for the instance of Figure 1. In the search tree, each node except the root is associated with a vertex in G, and accordingly, it is also associated with a subset of vertices such that the subset consists of vertices on the path from the root to the node. The black nodes represent base connectors in $\mathcal{B}$, whereas the gray nodes represent connectors in $\mathcal{M} \setminus \mathcal{B}$. COPINE identifies whether the vertex subset X of the visited node is a connector or not, and outputs X if it is so. It has a mechanism for pruning the tree, by which redundant search is avoided. For example, if $G[X]$ is disconnected, then COPINE skips the search of the descendants of the current node.

COOMA enumerates connectors in a completely different way. The nodes indicated by a rectangle, that is $\{v_1, v_2, v_6, v_9\}, \{v_4\}, \{v_7\} \in \mathcal{C}[V_{\langle i_1 \rangle}] \subseteq \mathcal{B}$, represent the connectors in $\mathcal{M}'$ as of line 3 in Algorithm 1. In the while-loop from line 5 to 14, for $i = i_2$, the connectors indicated by a rounded rectangle, that is $\{v_2, v_9\} \in \mathcal{M} \setminus \mathcal{B}$ and $\{v_2, v_3, v_7, v_8, v_9\} \in \mathcal{C}[V_{\langle i_2 \rangle}] \subseteq \mathcal{B}$, are added to $\mathcal{M}'$. For $i = i_3$ in the next iteration, the connectors indicated by a pentagon, that is $\{v_3\}, \{v_6, v_9\}, \{v_9\} \in \mathcal{M} \setminus \mathcal{B}$ and$\{v_3, v_5, v_6, v_7, v_9\} \in \mathcal{C}[V_{\langle i_3 \rangle}] \subseteq \mathcal{B}$, are added to $\mathcal{M}'$.

For the GenCE problem, pruning strategies are possible for both algorithms. When $\theta_V > 1$, COOMA does not need to maintain connectors X such that $|X| < \theta_V$. Specifically, we do not need to retain radix trees $R_1, \ldots, R_{\theta_V - 1}$. This is because $Y \in \mathcal{C}(X \cap C)$ satisfies $|Y| \leq |X| < \theta_V$ for $C \in \mathcal{B}$. We may regard that COOMA can prune nodes in low depths of the search tree. On the other hand, when $\theta_I > 1$, if COPINE visits a node such that the corresponding subset X satisfies $A_\sigma(X) < \theta_I$, then it can skip the search of descendants. This is because, for connectors $X, Y \in \mathcal{M}$, $X \subseteq Y$ implies $A_\sigma(X) \supseteq A_\sigma(Y)$ and thus $\theta_I > |A_\sigma(X)| \geq |A_\sigma(Y)|$. COPINE can prune nodes in high depths of the search tree.

5 Computational Experiments

We report some experimental results concerning COOMA in this section. First in Section 5.1, because $|\mathcal{M}|$, the total number of nontrivial connectors, has a great influence on the computation time of an enumeration algorithm, we make an empirical investigation on $|\mathcal{M}|$ of a random instance. In Section 5.2, we compare the three algorithms, COOMA, ExtCOOMA and COPINE, in terms of computation time, to demonstrate the efficiency of the former two algorithms. We also study when ExtCOOMA is more effective than COOMA. Then in Section 5.3, we discuss how much memory ExtCOOMA uses.

The experiments are done on a cygwin environment that is installed on a computer with an Intel Xeon CPU E5-1660 v3 (3.00 GHz) and 64GB RAM. We implemented the algorithms COOMA and ExtCOOMA in C++. For COPINE, we employ the source code (written in C) that Dr. Okuno kindly provided to us [11, 12, 13]. We compile the source codes of the algorithms by the gcc compiler (ver. 7.3.0) with `-O2` option.

We treat random instances in the experiments. We generate a random instance as follows, using four parameters, n, q, ρ_E and ρ_I, where n and q are positive integers and $\rho_E, \rho_I \in [0, 1]$. For the graph, we generate a random graph of the Erdös-Rényi model such that $|V| = n$ and an edge is drawn between any two vertices with probability ρ_E. We take the item set I with $|I| = q$ and associate a vertex with an item $i \in I$ with probability ρ_I. Given an instance (G, I, σ) that is generated in this way, we call $\frac{|E|}{\binom{n}{2}}$ the *edge density*, and $\frac{\sum_{v \in V} |\sigma(v)|}{|V||I|}$ the *item density*. The parameters ρ_E and ρ_I determine the expected values of the edge density and the item density, and we call them the *edge density parameter* and the *item density parameter*, respectively.

We deal with the CE problem (i.e., $\theta_V = \theta_I = 1$) and apply Reductions 2 and 3 to reduce a given instance.

5.1 Total Number of Nontrivial Connectors

We count $|\mathcal{M}|$ of a random instance. Fixing $n = |V| = 100$ and $q = |I| = 20$, we evaluate how $|\mathcal{M}|$ changes with respect to ρ_I, where ρ_E is taken from $\{0.05, 0.10, 0.25\}$. We show the result in Figure 3. In the figure, the horizontal axis indicates ρ_I, and the vertical axis indicates $|\mathcal{M}|$ in a logarithmic scale. For each (n, q, ρ_E, ρ_I), we generate five random instances with different random seeds.

As shown in the figure, $|\mathcal{M}|$ is generally increasing with respect to ρ_I, up to $\rho_I = 0.95$. Because the vertical axis employs a logarithmic scale, we do not plot points for $\rho_I = 0$; it holds that $|\mathcal{M}| = 0$ when $\rho_I = 0$, i.e., no item is given to a vertex. The number $|\mathcal{M}|$ dramatically decreases when $\rho_I > 0.95$. In particular, when $\rho_I = 1$, $|\mathcal{M}|$ equals to the number of connected components of a graph because every vertex is given all items and thus $A_\sigma(X) = I$ holds for all vertex subsets $X \subseteq V$. Hence, if the graph is connected, then it holds that $|\mathcal{M}| = 1$. We also see that, given an item density parameter ρ_I, the larger the edge density parameter ρ_E is, the larger $|\mathcal{M}|$ is likely to be.

It is expected that $|\mathcal{M}|$ becomes so large when the instance is "dense," that is, the edge density and/or the item density are large to some extent. It must be intractable to enumerate connectors from an instance that is dense as well as large (i.e., having many vertices and/or items).

However, many existing data are known to be "sparse" [7]. For example, the genetic database provided by Dr. Jiexun Wang, a biostatistician from Khoo Teck Puat Hospital in Singapore, is sparse in the sense of the item density. The database consists of 22 data sets, one of which corresponds to a pair of autosome chromosomes of a human cell. Each data set can be transformed into an instance of the CE problem such that the item density is just around 0.05. The instances are so small that the numbers of vertices are from 50 to 300. In our preliminary experiments, we confirmed that the three algorithms (i.e., COOMA,

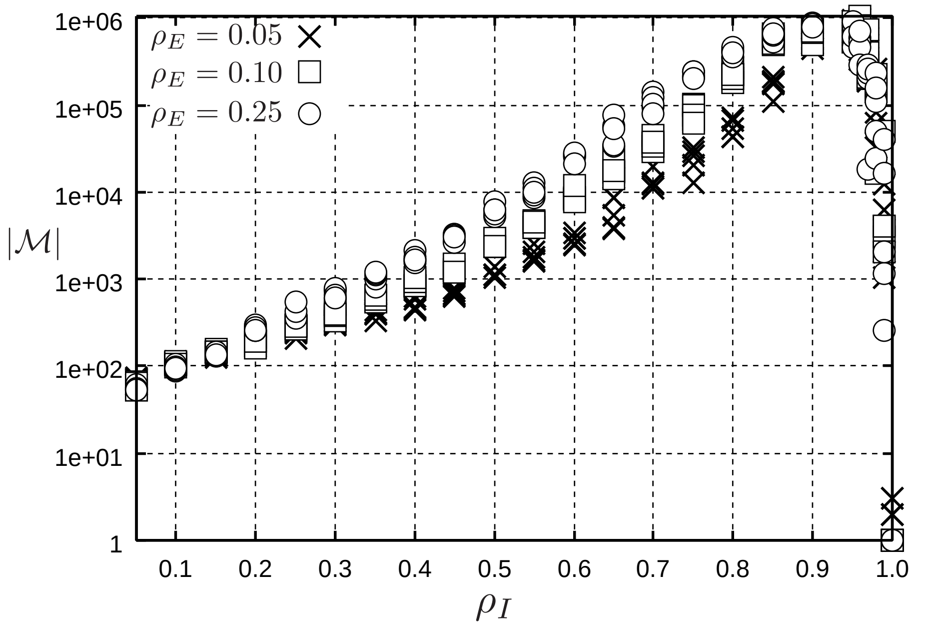

Figure 3: Change of the number $|\mathcal{M}|$ of nontrivial connectors with respect to the item density parameter ρ_I; $n = |V| = 100$, $q = |I| = 20$, and $\rho_E \in \{0.05, 0.10, 0.25\}$

ExtCOOMA and COPINE) enumerate all nontrivial connectors within a couple of seconds.

Another example is the DBLP data set from [19], which consists of 108,030 vertices, 276,653 edges, and 23,285 items. This instance is huge, but is tractable as it is sparse; the edge density is 4.7×10^{-5} and the item density is 5.9×10^{-4}, which are much smaller than the values we have used in the experiment. In fact, our ExtCOOMA enumerates 43,334,401 connectors in about 40 minutes.

Based on the fact that many existing data are sparse, we use small values for ρ_I in the subsequent experiments.

5.2 Computation Time

We evaluate the computation times of the three algorithms (i.e., COOMA, ExtCOOMA and COPINE) for random instances. For (n, q, ρ_E, ρ_I), we take $n \in \{100, \ldots, 1200\}$, $q \in \{100, 200, 300\}$, $\rho_E \in \{0.10, 0.25, 0.50\}$, and $\rho_I \in \{0.05, 0.10, 0.15\}$. We generate five instances with different random seeds for each (n, q, ρ_E, ρ_I).

We show the result in Figure 4. In the figure, the vertical axis indicates the computation time, and the horizontal axis indicates $\Delta|I|\,\|\mathcal{M}\|$; recall that the running time of COOMA is $O(\Delta|I|\,\|\mathcal{M}\|)$ (Theorem 2). The three symbols $\circ$ (ExtCOOMA), $\times$ (COOMA) and $\Box$ (COPINE) on the same vertical line show the computation time for the same instance.

COOMA and ExtCOOMA are much faster than COPINE when $\Delta|I|\,\|\mathcal{M}\|$ is large to some extent (e.g., $\Delta|I|\,\|\mathcal{M}\| \geq 0.5 \times 10^{12}$). The computation time of COOMA and ExtCOOMA increases almost linearly with respect to $\Delta|I|\,\|\mathcal{M}\|$, whereas the computation time of COPINE is more sensitive to the parameters; we see two major curves for COPINE. The left one is for $\rho_E = .25$, and the right one is for $\rho_E = .50$. We also see that ExtCOOMA is faster than COOMA although the difference is much smaller than the difference between ExtCOOMA (and COOMA) and COPINE.

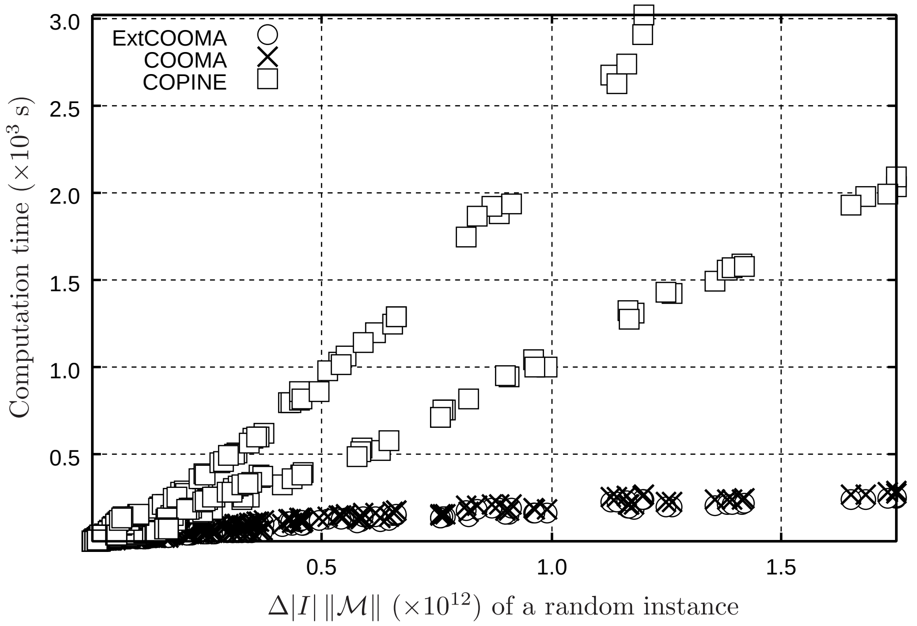

Figure 4: Computation time of the three algorithms for random instances

Next, we analyze when ExtCOOMA is more effective than COOMA. In Figure 5, we show how the size $r = |\mathbb{C}|$ of a base cover $\mathbb{C} = \{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$ constructed by the heuristic method of Section 3.2 changes with respect to the item density parameter ρ_I. In this experiment, we fix $n = |V| = 200$ and $q = |I| = 100$, and the edge density parameter ρ_E is taken from 0.05, 0.10 and 0.25. As shown, we obtain a base cover that is significantly smaller than I when $\rho_I \le 0.20$. When $\rho_I > 0.20$, the size $|\mathbb{C}|$ of an obtained base cover $\mathbb{C}$ is around 100 ($= q$). This phenomenon is explained as follows; when $\rho_I > 0.20$, because the item density is rather high, it is likely that $\mathcal{C}[V_{\langle i \rangle}]$ consists of exactly one base connector and thus there are q base connectors, and that $C \cap C' \neq \emptyset$ holds for any two base connectors $C \in \mathcal{C}[V_{\langle i \rangle}]$ and $C' \in \mathcal{C}[V_{\langle i' \rangle}]$ ($i \neq i'$). Hence the method of Section 3.2 constructs $\mathbb{C} = \{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$ just by sorting base connectors $C_1, \ldots, C_q$ in a nondecreasing order of the cardinality so that $|C_1| \le \cdots \le |C_q|$, and by letting $\mathcal{B}_p = \{C_p\}$, $p = 1, \ldots, r$.

We show in Figure 6 the ratio of the computation time of ExtCOOMA over the computation time of COOMA. When $\rho_I \le 0.20$, i.e., when r is significantly smaller than q, the ratio is below 1.0 in general which means that ExtCOOMA runs faster than COOMA. Interestingly, when $\rho_I > 0.20$, although it holds that r is approximately equal to q, the ratio is from 0.7 to 0.8. This is supported by the observation in Section 3.2; in a family $\mathcal{B}_p$ with a small p, we should include as many "small" base connectors as possible.

5.3 Memory Usage

Let us observe how much memory the algorithm ExtCOOMA consumes. For (n, q, ρ_E, ρ_I), we take $n \in \{100, \ldots, 1200\}$, $q \in \{100, 200, 300\}$, $\rho_E \in \{0.10, 0.25, 0.50\}$, and $\rho_I \in \{0.05, 0.10, 0.15\}$. We generate five instances with different random seeds for each (n, q, ρ_E, ρ_I).

Figure 7 shows the amount of memory used by ExtCOOMA (All), along with the amount of memory that is used to store the instance (Instance). The horizontal axis indicates $\|\mathcal{M}\|$, the sum of $|X|$ over $X \in \mathcal{M}$, and the vertical axis indicates the amount of memory. The amount of memory is evaluated by the `VmSize` value of the file `/proc/self/status` (i.e., the

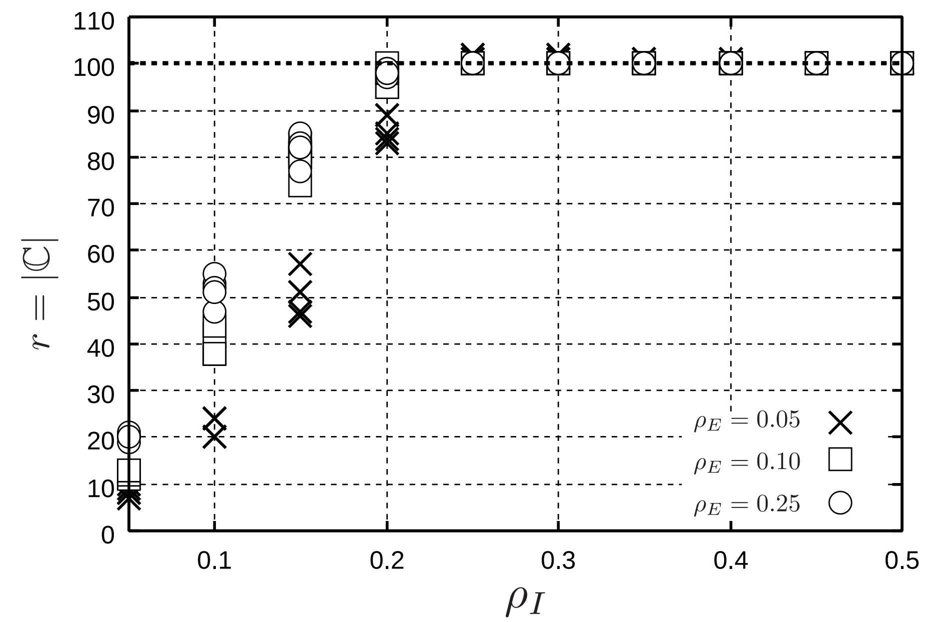

Figure 5: Size $r = |\mathbb{C}|$ of a base cover $\mathbb{C}$ that is constructed by the method mentioned in Section 3.2 ($n = |V| = 200$, $q = |I| = 100$ and $\rho_E \in \{0.05, 0.10, 0.25\}$)

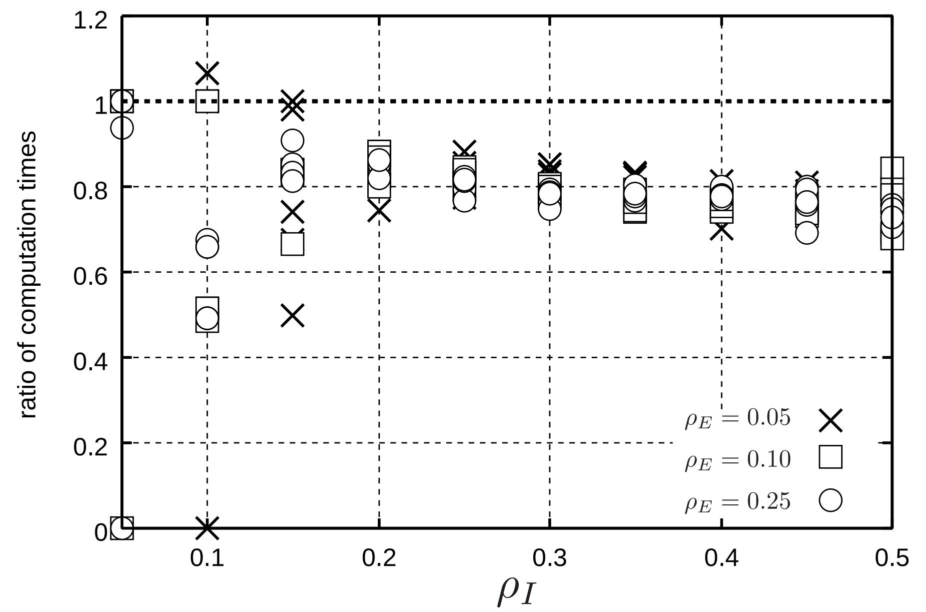

Figure 6: Ratio of the computation time of ExtCOOMA over the computation time of COOMA ($n = |V| = 200$, $q = |I| = 100$ and $\rho_E \in \{0.05, 0.10, 0.25\}$); When the computation time of COOMA is smaller than 10^{-3} seconds, the ratio is set to zero.

amount of virtual memory used by the current process) in the cygwin environment.

As shown in the figure, the amount of memory needed to store the instance (Instance) is much smaller than the whole amount of memory used by ExtCOOMA (All). The "All" amount increases almost linearly with respect to $\|\mathcal{M}\|$. We see two "All" lines; the upper one is for instances generated by $\rho_E = 0.50$ and the lower one is for instances generated by $\rho_E = 0.25$. The reason why there are such lines is described as follows; the more the edge

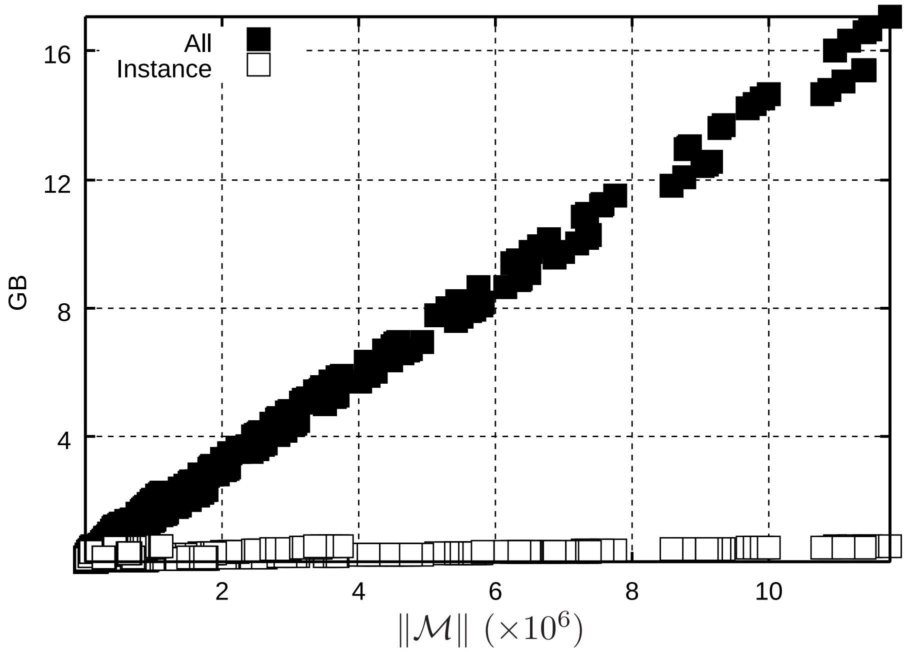

Figure 7: (All) the whole amount of memory used by ExtCOOMA; (Instance) the amount of memory that is used to store the instance

density is, the more the largest connector size $b_{\max}$ should be. As mentioned in Section 3.1, we store $b_{\max}$ radix trees in our implementation. Hence, we need more radix trees for instances generated by $\rho_E = 0.50$ than instances generated by $\rho_E = 0.25$. This should cause the two lines in the figure.

6 Concluding Remark

We have proposed a novel algorithm COOMA for the connector enumeration problem. The running time is polynomially bounded with respect to the input and output size. We have shown the empirical efficiency in comparison with COPINE.

We will extend the problem to other graph models (e.g., hypergraphs, digraphs and vertex-and/or edge-weighted cases) and consider various requirements (e.g., k-edge- and/or k-vertex-connectivity, min/max degree and flow values or distance in weighted versions).

Acknowledgment

We gratefully acknowledge Dr. Shingo Okuno for sharing his COPINE source code with us. We also thank Dr. Jiexun Wang for providing the genetic data to us.

References

[1] Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics **65**(1), 21–46 (1996)

[2] Bellman, R.E.: Dynamic Programming. Princeton University Press (1957)

[3] Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Company (1979)

[4] Imada, T., Ota, S., Nagamochi, H., Akutsu, T.: Efficient enumeration of stereoisomers of tree structured molecules using dynamic programming. Journal of Mathematical Chemistry **49**(4) (2011), `http://hdl.handle.net/2433/139571`

[5] Inokuchi, A., Washio, T., Motoda, H.: Complete mining of frequent patterns from graphs: Mining graph data. Machine Learning **50**(3), 321–354 (2003), `https://doi.org/10.1023/A:102172622`

[6] Knuth, D.E.: The Art of Computer Programming (Sorting and Searching), vol. 3. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2nd edn. (1998)

[7] Kuramochi, M., Karypis, G.: Finding frequent patterns in a large sparse graph. Data Mining and Knowledge Discovery **11**(3), 243–271 (2005), `https://doi.org/10.1007/s10618-005-0003-9`

[8] Lewis, R.M.R.: A Guide to Graph Coloring. Springer (2016)

[9] Minato, S.: Zero-suppressed bdds for set manipulation in combinatorial problems. In: Proceedings of the 30th International Design Automation Conference (DAC '93). pp. 272–277 (1993)

[10] Minato, S.: Power of enumeration - recent topics on BDD/ZDD-based techniques for discrete structure manipulation. IEICE Transactions on Information and Systems **E100.D**(8), 1556–1562 (2017)

[11] Okuno, S.: Parallelization of Graph Mining using Backtrack Search Algorithm. Ph.D. thesis, Kyoto University (2017), `https://doi.org/10.14989/doctor.k20518`

[12] Okuno, S., Hiraishi, T., Nakashima, H., Yasugi, M., Sese, J.: Reducing redundant search in parallel graph mining using exceptions. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 328–337 (2016), `https://doi.org/10.1109/IPDPSW.2016.136`

[13] Okuno, S., Hiraishi, T., Nakashima, H., Yasugi, M., Sese, J.: Parallelization of extracting connected subgraphs with common itemsets. Information and Media Technologies **9**(3), 233–250 (2014), `https://doi.org/10.11185/imt.9.233`

[14] Pietracaprina, A., Zandolin, D.: Mining frequent itemsets using patricia tries. In: Proceedings of Workshop on Frequent Itemset Mining Implementations (FIMI 03) (2003)

[15] Sedgewick, R.: Algorithms in C: Parts 1-4, Fundamentals, Data Structures, Sorting, and Searching. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edn. (1997)

[16] Seki, M., Sese, J.: Identification of active biological networks and common expression conditions. In: 2008 8th IEEE International Conference on BioInformatics and BioEngineering. pp. 1–6 (2008)

[17] Sese, J., Seki, M., Fukuzaki, M.: Mining networks with shared items. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10). pp. 1681–1684 (2010)

[18] Sese, J., Terada, A., Saito, Y., Tsuda, K.: Statistically significant subgraphs for genome-wide association study. In: Proceedings of the Workshop on Statistically Sound Data Mining at ECML/PKDD. Proceedings of Machine Learning Research, vol. 47, pp. 29–36 (2015)

[19] Silva, A., Meira, Jr., W., Zaki, M.J.: Mining attribute-structure correlated patterns in large attributed graphs. Proceedings of VLDB Endow. **5**(5), 466–477 (2012)

[20] Wasa, K.: Enumeration of enumeration algorithms. CoRR **abs/1605.05102** (2016), `http://arxiv.org/abs/1605.05102`

[21] Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: Proceedings of 2002 IEEE International Conference on Data Mining (ICDM '02). pp. 721–724 (2002)