Manuscript, 2013.05.13, Wasa and Arimura, IKN Lab., Hokkaido University (under submission)

Polynomial Delay and Space Discovery of Connected and Acyclic Sub-Hypergraphs in a Hypergraph

Kunihiro Wasa[1], Takeaki Uno[2], Kouichi Hirata[3], and Hiroki Arimura[1]

[1] Graduate School of IST, Hokkaido University, Sapporo, Japan
{wasa,arim}@ist.hokudai.ac.jp
[2] National Institute of Informatics, Tokyo, Japan
uno@nii.jp
[3] Dept. of Artificial Intelligence, Kyushu Institute of Technology, Iizuka, Japan
hirata@ai.kyutech.ac.jp

Abstract. In this paper, we study the problem of finding all connected and Berge-acyclic sub-hypergraphs contained in an input hypergraph with potential applications to substructure mining from relational data, where Berge acyclicity is a generalization of a tree and one of the most retricted notion of acyclicity for hypergraphs (Fagin, J. ACM, Vol.30(3), pp.514–550, 1983). As main results, we present efficient depth-first algorithm that finds all connected and Berge acyclic sub-hypergraphs S contained in an input hypergraph $\mathcal{H}$ with m hyperedges and n vertices in $O(nm^2)$ delay (time per solution) using $O(N)$ space, where $N = ||\mathcal{H}||$ is the total input size. In the algorithms, we use maximum elimination sequences. This result gives the first polynomial delay and time algorithm for discovery of Berge-acyclic sub-hypergraphs. We also discuss adaptive enumeration whose delay depends only on a parameter on each discovered subgraph S, rather than the whole input of size N. Our modified algorithm runs in $O(k + k')$ delay using $O(N)$ space and preprocessing, where its delay depends only on the size $k = ||S||$ of a sub-hypergraph S to find and the size $k' = ||N(S)||$ of its neighbor hyperedges.

1 Introduction

It is a well-studied problem to enumerate all interesting substructures of a given discrete structure under various notions of subclasses of substructures to extracts. In particular, such enumeration algorithms have played fundamental roles in data mining such as frequent itemset mining [24, 27, 28], sequence mining [1, 2, 18], trees and graph mining [3, 12, 13, 26], and kernel-like similarity computation [15].

In this paper, we study the problem of enumerating all connected and acyclic sub-hypergraphs contained in an input hypergraph for the notions of acyclicity, called Berge-acyclicity [6], which is at the bottom of hierarchy of acyclicities given by Fagin [9]. A *hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ of a collection $\mathcal{V}$ of vertices and a collection $\mathcal{E}$ of hyperedges (See Fig. 1 for example), where a hyperedge is any finite set $e \subseteq \mathcal{V}$ of vertices. Essentially, a hypergraph is a representation of

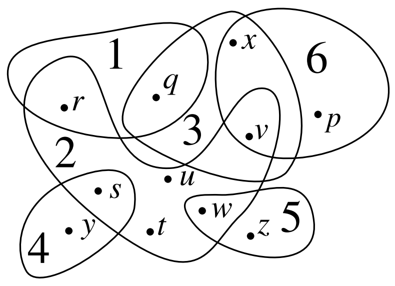

Fig. 1. An example of a hypergraph $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ with the vertex set $\mathcal{V}_1 = \{p, q, r, \dots, x, y, z\}$ and the hyperedge set $\mathcal{E}_1 = \{1, 2, 3, 4, 5, 6\}$, where each point with a lowercase letter indicates a vertex and each region with a digit surrounded by a solid line indicates a hyperedge. The subset of hyperedges $S = \{1, 2, 4, 5\}$ forms a connected and Berge-acyclic sub-hypergraph of $\mathcal{H}_1$ to discover, while its super set $\{1, 2, 3, 4, 5\}$ is not acyclic.

any group relations, or any *set collection* consisting of groups of objects taken from the universe. For example, the followings are examples of such set collections: transaction databases, author groups in bibliographic data, co-citation networks in social networks, and interaction graphs for genes and proteins in bioinformatics [16]. In such networks, discovery of some classes of subsets, such as connected components, connected subtrees, cliques, quasi-cliques, and dense subgraphs have been extensively studied in the context of network mining [16, 20, 23].

An *acyclic* sub-hypergraph is a generalization of the notion of subtrees for hypergraphs, which means the hypergraph contains no cycle of connecting hyperedges. In the example of Fig. 1, the subset consisting of hyperedges 1, 2, 4, and 5 forms such a connected and acyclic sub-hypergraph. Particularly, we consider Berge-acyclicity (Berge, 1973), which locates the bottom of the degrees among other notions of acyclicities such as α-, β-, and γ-acyclicities. A Berge-acyclic graph has tree-like shape consisting of hyperedges. Thus, mining of such connected and Berge-acyclic sub-hypergraphs in a hypergraph can be regarded as finding less redundant subsets of groups reachable by chains of neighbor relations.

Main results: We present efficient depth-first algorithms BergeMine (Theorem 1) that finds all connected Berge-acyclic sub-hypergraphs contained in an input hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ without duplicates in $O(Nm) = O(nm^2)$ delay (time per solution) and $O(N)$ words of space, where $n = |\mathcal{V}|$, $m = |\mathcal{E}|$, and $N = ||\mathcal{E}||$ are the numbers of vertices and hyperedges, and the total size of the hyperedges in $\mathcal{H}$. To achieve polynomial delay and space complexity, our algorithm searches for all solutions in the depth-first manner on a tree-shaped search space without using any extra memory for table-lookup. This search space is designed based on a characterization of Berge-acyclic sub-hypergraphs given by us, with which we proposed an efficient and complete pruning strategy.

Next, we present the faster version of the algorithm, called FastBergeMine, using adaptive computation. The algorithm uses incremental computation of the maximum border set, and finds all connected Berge-acyclic sub-hypergraphs contained in $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ without duplicates in $O(||S|| + ||N(S)||)$ delay using $O(N)$ space and $O(N)$ preprocessing, where $||S|| = O(nm)$ is the total length of hyperedges in S and $||N(S)||$ is the sum of the lengths of hyperedges adjacent to S. This algorithm has the delay that depends only on the size $||S|| + ||N(S)||$ of a discovered subset S and its neighbors, and thus, it will be more efficient for the large inputs in the real world.

Related work: For the class of α-acyclic sub-hypergraphs [9], Hirata *et al.* [11] presented an efficient algorithm that finds one of the maximal connected and acyclic sub-hypergraphs in an input hypergraph in linear time in the total input size. Extending this work, Daigo and Hirata [8] presented a polynomial delay and space algorithm that finds all connected and acyclic sub-hypergraphs in an input hypergraph. For the class of Berge-acyclic sub-hypergraphs, Lovász [17] showed a polynomial time algorithm that finds one of the maximal connected and Berge-acyclic sub-hypergraphs in an input hypergraph.

As closely related work, Ferreira *et al.* [10] presented an efficient algorithm for finding all distinct subtrees of size k in an input graph in $O(k)$ time (time per solution) and space, and Wasa *et al.* [25] the improved version in constant delay when an input is a tree. However, their approaches cannot be directly applicable to our problem.

In the case that the maximum size of hyperedges, the *rank*, is restricted to two, the problem coincides to the well-known spanning tree problem for undirected graphs. For the problem, Tarjan and Read [22] first presented an $O(ns)$ time and $O(n)$ space algorithm in 1960's, where n is the number of edges in G. Recently, Shioura, Tamura, and Uno [19] presented $O(n+s)$ time and $O(n)$ space algorithm. Unfortunately, it is not easy to extend the algorithms for spanning tree enumeration to subtree enumeration.

Organization of this paper: In Sec. 2, we give basic definitions and notations on hypergraphs and our data mining problem. In Sec. 3, we present the basic depth-first algorithm BergeMine for the problem. in Sec. 4, we present the modified version of the algorithm, FastBergeMine, using incremental computation. Finally, in Sec. 5, we conclude.

2 Preliminaries

In this section, we give the definitions and notations for hypergraphs. For the definitions not found here, please consult appropriate textbooks (e.g., [7] or [6]).

2.1 Hypergraphs

A *hypergraph* over a set of vertices $\mathcal{V}$ is any pair $\mathcal{H} = (\mathcal{V}, \mathcal{E}) = (\mathcal{V}(\mathcal{H}), \mathcal{E}(\mathcal{H}))$ consists of the following components:

- A set of *vertices* $\mathcal{V} = \{1, \ldots, n\}$, $n \geq 0$, and

– A set of *hyperedges* $\mathcal{E} = \{e_1, \dots, e_m\}$, $m \geq 0$,

where for every $1 \leq i \leq m$, the hyperedge e_i is any subset $e_i \subseteq \mathcal{V}$ of $\mathcal{V}$, and its index i is called the *edge ID* of e_i.

In this paper, we fix an input hypergraph $\mathcal{H} = (\mathcal{V}(\mathcal{H}), \mathcal{E}(\mathcal{H}))$ consists of n vertices and m hyperedges. Then, a *subset* of hyperedges in $\mathcal{H}$ is just a subset $S \subseteq \mathcal{E}(\mathcal{H})$, where the underlying vertex set is denoted by $V(S) = \{\, x \in \mathcal{V} \mid x \in e, e \in S \,\} = \bigcup S$. The *total size* of $\mathcal{S}$ is defined by the sum $||S|| = \sum_{e \in S} |e|$ of the sizes of its members. For the analysis of the adoptive complexity of the algorithm in Sec. 4, which depends only on the solution S, we define the *neighbor size* of S by $||N(S)|| = \sum_{e \in S} |N(e)|$. In this way, we refer to any subset $S \subseteq \mathcal{E}(\mathcal{H})$ as a *sub-hypergraph* of $\mathcal{H}$ [4], meaning $(V(S), S)$ if it is clear from context.

In what follows, we refer to vertices as $x, y \in \mathcal{V}$, hyperedges as $e, f \subseteq \mathcal{V}$, and hyperedge subsets as $S, T \subseteq 2^{\mathcal{V}}$, possibly subscripted.

Example 1. Let $\mathcal{V}_1 = \{p, q, r, \dots, x, y, z\}$ be a set of eleven vertices. In Fig. 1, we show an example of a hypergraph $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ on $\mathcal{V}_1$. In $\mathcal{H}_1$, the hyperedge set $\mathcal{E}_1$ consists of six hyperedges $e_1 = \{q, r\}, e_2 = \{r, s, t, u, v, w\}, e_3 = \{q, v, x\}, e_4 = \{s, y\}, e_5 = \{w, z\}$, and $e_6 = \{p, v, x\}$, where the hyperedge e_i, $1 \leq i \leq 6$, is represented by the index i.

2.2 Connected and Berge-acyclic sub-hypergraphs

A *path* between hyperedges e and $f \in \mathcal{E}$ in $S \subseteq \mathcal{E}$ is a sequence $\pi = (e_1 = e, e_2, \dots, e_k = f)$ $(k \geq 1)$ of hyperedges that satisfies the condition $e_i \cap e_{i+1} \neq \emptyset$ for every $1 \leq i \leq k-1$.

Definition 1. A subset S is *connected* if any pair of hyperedges e and f has some path between them in S.

By definition, the empty set and singletonset of hyperedges are connected. We can easily test the connectivity of a given subset S in $O(||S||)$ time.

Example 2. In the hypergraph $\mathcal{H}_1$ in Example 1, both of the subsets $S_1 = \{1, 2, 3, 4\}$ and $S_2 = \{1, 2, 4, 5\}$ are connected. On the other hand, the subset $S_3 = \{1, 3, 5\}$ is not connected since there is no path between the edges 1 and 5, and also the edges 3 and 5.

Definition 2 ([6]). *A* Berge-cycle *(or simply a* cycle*) in a hypergraph $\mathcal{H}$ is a sequence $\pi = (e_1, x_1, \dots, e_k, x_k)$ $(k \geq 2)$ that satisfies the following conditions (i)–(iii):*

(i) $e_1, \dots, e_k$ *are mutually distinct hyperedges.*
(ii) $x_1, \dots, x_k$ *are mutually distinct vertices.*
(iii) For each $1 \leq i \leq k-1$, $x_i \in e_i \cap e_{i+1}$ *holds, and* $x_k \in e_k \cap e_1$ *holds,*

[4] The definition of a sub-hypergraph in this paper is also refered to as a *partial hypergraph* in literatures.

where k is called the length of the cycle π. Then, we say that the set $\{e_1, \ldots, e_k\}$ of hyperedges forms a Berge-cycle.

Intuitively, a Berge-cycle is a path of length more than or equal to two that starts from some hyperedge and returns to the start.

Example 3. In the hypergraph $\mathcal{H}_1$ in Example 1, the hyperedge subset $S_4 = \{1, 2, 3\}$ forms a Berge-cycle $\pi_4 = (1, r, 2, v, 3, q)$ of length three. From Lemma 1, we also see that the pair of hyperedges $S_5 = \{3, 6\}$ forms a Berge-cycle $\pi_5 = \langle 3, x, 6, v\rangle$ of length two since hyperedges 3 and 6 share common vertices x and v.

From the construction of minimum length cycle S_5 in the above example, we have the following lemma, which is well-known providing a fundamental property of Berge-acyclicity.

Lemma 1 (Berge [6]). *If two hyperedges e and f contain mutually distinct vertices x and y in common, i.e., $x, y \in e \cap f$, then they form a Berge-cycle.*

Proof. Take a path $\pi = (e, x, f, y)$ as a Berge-cycle. ■

Definition 3 (Berge-acyclic subgraph [6]). *A sub-hypergraph S is* Berge-acyclic *if it contains no Berge-cycles.*

By definition, the empty set and any singleton sets of hyperedges are Berge-acyclic. From the next lemma, Berge-acyclicity is closed under subsets.

Lemma 2. *If a non-empty subset S is Berge-acyclic, then any subset S' ($S' \subseteq S$) is also Berge-acyclic.*

From Lemma 1 above, we see that Berge-acyclicity has strong restriction compared to other notions of hypergraph acyclicities. Actually, there is a hierarchy of acyclicities for hypergraphs, called the *degrees of acyclicities* of Fagin [9], that consists of α-acyclicity, β-acyclicity, γ-acyclicity, and Berge-acyclicities. In this hierarchy, α-acyclicity is most general, while Berge-acyclicity is most restricted.

In what follows, we denote by $\mathcal{AC} = \mathcal{AC}(\mathcal{H})$ the class of *all connected, and Berge-acyclic sub-hypergraphs* in an input hypergraph $\mathcal{H}$. Now, we state our data mining problem.

Definition 4 (Connected and Berge-acyclic sub-hypergraph mining problem in a hypergraph). *Given an input hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, the task is to find all connected, and Berge-acyclic sub-hypergraphs $S \subseteq \mathcal{E}$ in $\mathcal{H}$ belonging to the class $\mathcal{AC}(\mathcal{H})$ without duplicates.*

Example 4. Cosider the hypergraph $\mathcal{H}_1$ in Example 1 again. Then, the subset $S_1 = \{1, 2, 3, 4\}$ is not a connected and Berge-acyclic subset in $\mathcal{AC}(\mathcal{H}_1)$ because it is connected but cyclic. On the other hand, the subset $S_2 = \{1, 2, 4, 5\}$ is a connected and Berge-acyclic subset in $\mathcal{AC}(\mathcal{H}_1)$

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} p & q & r & s & t & u & v & w & x & y & z \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Fig. 2. The incident matrix A_1 of the hypergraph $\mathcal{H}_1$ in Fig. 1, where each row indicates a vertex and each column indicates a hyperedge.

Our goal here is to devise an efficient algorithm for the connected and Berge-acyclic sub-hypergraph mining problem in the complexities of *polynomial delay* and *polynomial space*, Intuitively, this means that an algorithm has high-throughput and small memory footprint, both are necessary properties for large scale applications of data mining algorithms.

Formally, we evaluate the performance of a data mining algorithm in terms of enumeration algorithms (Avis and Fukuda [4]). Let N and M be the input size and the number of patterns as solutions. An enumeration algorithm $\mathcal{A}$ outputs all solutions, namely *connected and Berge-acyclic subsets* here, after receiving an input. $\mathcal{A}$ is said to have *polynomial delay* (poly-delay) if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a polynomial $p(N)$ in N. $\mathcal{A}$ is of *polynomial space* (poly-space) if the maximum size of its working space, in addition to that of output stream O, is bounded by a polynomial $p(N)$.

2.3 Other definitions and properties

Leaves and connection counts: Let $S \subseteq \mathcal{E}$ be a subset of hyperedges, or a sub-hypergraph of $\mathcal{H}$. A hyperedge e *connects* S if the intersection $e \cap V(S)$ is not empty. Any vertex x in the intersection is called a *connection point*. Then, the *connection count* of e relative to S is defined by $cnt(f, S) = |f \cap V[S]| \geq 0$. In the next section, we give a characterization of connected and Berge-acyclic sub-hypergraphs using the connection count.

Definition 5 (leaf). *A* leaf *of a subset S is a hyperedge $e \in S$ such that $cnt(e, S - e) = 1$, that is, that has a single connection point in S except itself.*

Clearly, the empty subset $\emptyset$ has no leaf at all, and any singleton $S = \{e\}$ has the hyperedge e as its only leaf. We denote by $L(S)$ the *set of all leaves* of S. Actually, $L(S) = \{\, e \in S \mid cnt(e, S \setminus \{e\}) = 1 \,\}$. Roughly speaking,

Representation of hypergraphs: Let $x \in \mathcal{V}$ be a vertex, and $f \subseteq \mathcal{V}$ be hyperedges. If $x \in e$ holds, then we say that either the vertex x is *contained in* a hyperedge e, or the hyperedge e is *incident to* a vertex x. We denote the set of all hyperedges in $\mathcal{E}$ that is incident to the vertex x by $N(x) = \{\, f \in \mathcal{E} \mid x \in f \,\}$. Hyperedges f is *adjacent to* e or e is a *neighbor* of f if f overlaps e, that is, $f \cap e \neq \emptyset$ holds.

Using the incident relation, a hypergraph $\mathcal{H}$ with n vertices and m hyperedges is represented in our algorithms, to be described later, as an $n \times m$ binary matrix $A = (a_{i,j}) \in \{0,1\}^{m \times n}$, called the *incident matrix* of $\mathcal{H}$, in a standard way, where for every $1 \leq i \leq n$ and $1 \leq j \leq m$, $a_{i,j} = 1$ if and only if the i-th vertex x_i is contained in the j-th hyperedge e_j, that is, $x_i \in e_j$ $(1 \leq i \leq n, 1 \leq j \leq m)$ holds. In Fig. 2, we show an example of an incident matrix. Actually, each row $1 \leq i \leq n$ represents the incident set $N(x_i)$ of x_i, and each column $1 \leq j \leq m$ represents the corresponding hyperedge e_j itself as a set of vertices.

Data structure: In our algorithms in Sec. 3 and Sec. 4, we use a dynamic data structure, denoted by $\mathcal{D}$, similar to the *DLX* (also known as "*Dancing Links*") data structure by Knuth [14], for efficiently and dynamically maintaining a given subset of hyperedges of $\mathcal{H}$ in the form of incident matrix. The difference of our structure from Knuth's DLX is the use of the dynamic predecessor dictionaries (such as the hash table or the `map` collection of C++/STL or Java) [7].

Our data structure $\mathcal{D}$ stores the incident matrix of a set collection $D \subseteq \mathcal{E}$ in linear words of space in $||D||$ supporting the following operations: (i) retrieval of a hyperedge $e = e_i$ by an edge ID i, (ii) retrieval of the neighbor $N(x, D)$ by a vertex x, and (ii) insert/delete of elements to/from an edge or a neighbor set. Such a data structure can be implmented by linked lists and dynamic predecessor dictionary that allows to execute the above operations in sublinear time $t = f(k)$, where we have $f(k) = \log k$ if we use ordinary binary tree, and $f(k) = O(((\log\log k)^2/\log\log\log k))$ for $k = \max\{n, m\}$ if we use the dynamic data structure of [5]. The details are omitted here.

3 The Basic algorithm

In this section, we show the basic version of our DFS mining algorithm BergeMine that finds all connected, and Berge-acyclic sub-hypergraphs in $\mathcal{H}$ in polynomial delay and space. In what follows, we write $S - e$ for $S' \setminus \{e\}$.

To devise efficient depth-first search algorithm, we need a systematic way to reduce the search for larger subsets to smaller subsets. The next lemma is essential to our algorithm.

Lemma 3. *Let $S' \subseteq \mathcal{E}$ is a subset such that $|S'| \geq 2$. If S' is connected and Berge-acyclic, then $cnt(e, S' \setminus \{e\}) = 1$ holds for some e. Furthermore, $S = S' \setminus \{e\}$ is connected and Berge-acyclic, too, and has size $|S| < |S'|$.*

Proof. We can show the lemma by induction on $|S'|$. If $|S'| = 2$, the claim is obvious since S' consists of two edges. Otherwise, assume that $|S'| > 2$. Since S' is connected, $cnt(e, S - e) \geq 1$ always holds for any $e \in S'$. Fruthermore, if $cnt(e, S - e) \leq 1$ holds for any $e \in S'$, then we are done. Therefore, we assume that $cnt(e, S - e) \geq 2$ holds for some $e \in S'$. Consider this case. Then, we split S' by removing e, and consider the connected components $S_1, \ldots, S_k$ of $S' - e$,

Algorithm 1 A basic algorithm BergeMine for mining all connected, Berge-acyclic sub-hyupergraphs based on the reverse search

```
1: procedure BergeMine($\mathcal{H} = (\mathcal{V}, \mathcal{E})$: input hypergraph )
2:     BasicRec($\emptyset, \mathcal{H}$);

3: procedure BasicRec($S$: sub-hypergraph, $\mathcal{H}$: input hypergraph)
4:     Output $S$;
5:     $Border(S) \leftarrow \{ f \in (\mathcal{E}(\mathcal{H}) \setminus S) \mid cnt(f, S) = 1 \}$;
6:     for each $f \in Border(S)$ do                    ▷ Generation of children
7:         $S' \leftarrow S \cup \{f\}$;
8:         if $f = \max L(S')$ then
9:             BasicRec($S', \mathcal{H}$);
```

where $k \geq 1$. There are two cases. (i) If e connects to some S_i at at least two points, then $S_i \cup \{e\}$, and thus S', immediately has a cycle, and we are done (ii) Otherwise, using induction hypothesis, we can show that there exists an edge f in some component, say S_1, such that $\{e, f\} \cup R$ forms a cycle for some $R \in \{\{e\}, S_1 - f, S_2, \ldots, S_k\}$ (details are omitted), and we are done. Hence, by contradiction, the lemma follows. ■

From Lemma 3 above, Starting from any connected and Berge-acyclic subhypergraph S with more than one edges, we can obtain a series of sub-hypergraphs $\mathcal{R} = S_0 = S \supset S_1 \supset \cdots \supset S_\ell = \{e\}$ of length $\ell = |S| - 1 \geq 0$. Our DFS algorithm reverses this process by starting from any singleton set $\{e\}$, $e \in \mathcal{E}$, and by iteratively expanding the current subset $S \subseteq \mathcal{E}$ by adding new hyperedge $e \in \mathcal{E} \setminus S$ in a systematic manner using backtracking.

However, there is one problem with this approach. The above DFS search process may generate the same subset by exponentially many different paths. One easy way to avoid this duplication is to use table-lookup. When we discovered a new subset S, we lookup a hash table H to decide if $S \in H$. If so, we skip S, and otherwise, we output S and register it to H. This modification yields a polynomial delay, but exponential space mining algorithm.

We solve this problem by pruning of redundant path by careful design of the tree-shaped search space described as follows. Recall the previous key lemma, Lemma 3. In the lemma, the possible source of redundancy is more than one choice of a leaf $e \in S$ of S to delete. An idea to solve this is to restrict the deletion in reduction (and the addition in generation) to the *maximum* leaf of S. This ensures the reduction sequence $\mathcal{R} = S_0, \ldots, S_\ell$ for S to be unique to each S. We call such a unique sequence the *maximum elimination sequence* for a sub-hypergraph S, and denote by $\mathcal{MES}(S)$.

Lemma 4. *$\mathcal{MES}(S)$ is the unique signature of each connected and Berge-acyclic sub-hypergraph $S \subseteq \mathcal{E}$.*

From this lemma, we can generate S in a unique way by generating $\mathcal{MES}(S)$ instead. Now, we describe our algorithm.

Definition 6. Let S' be any connected and Berge-acyclic subhypergraph S' such that $|S'| \geq 2$. Then, the *parent* of S' is the set $\mathcal{P}(S') = S' - f$, where f is the leaf of S such that $cnt(f, S' - f) = 1$ having the maximum edge ID among all leaves, that is, $f = \max(L(S))$. This condition is called the *maximum leaf condition*. In this case, we call S' a *child* of $S = \mathcal{P}(S')$.

Then, we define our tree-shaped search space. Let $\mathcal{H}$ be an input hypergraph. The *family tree* for the class $\mathcal{CA}$ of connected, and Berge-acyclic sub-hypergraphs of $\mathcal{H}$ is a multi-rooted DAG $\mathcal{T} = (\mathcal{CA}, \mathcal{P}, \mathcal{I})$, where

- $\mathcal{CA}$ is the vertex set of $\mathcal{T}$ that consists of all connected, and Berge-acyclic sub-hypergraphs in an input hypergraph $\mathcal{H}$.
- $\mathcal{P}$ defines the set of reverse edges of $\mathcal{T}$ that assign the parent $\mathcal{P}(S')$ to a child S'.
- $\mathcal{I}$ is the set of all single subsets as the root nodes of $\mathcal{T}$.

The next lemma says that the family tree is actually a tree-shaped search root.

Lemma 5. *For any input hypergraph $\mathcal{H}$, the family tree for $\mathcal{CA}$ on $\mathcal{H}$ is a spanning forest that contains all connected and Berge-acyclic subsets in $\mathcal{CA}$ as its nodes.*

Proof. From Lemma 3, it immediately follows that $\mathcal{P}(S')$ is always connected, and Berge-acyclic, and has size strictly smaller than S'. Since each path of $\mathcal{T}$ is a $\mathcal{MES}$ for some element of $\mathcal{CA}$, $\mathcal{T}$ is connected at some root in $\mathcal{I}$. On the other hand, since each reverse edge strictly reduces the size of S, $\mathcal{T}$ contains no cycle. Hence, the lemma is proved. ■

In Algorithm 1, we show our basic DFS algorithm BergeMine and its recursive subprocedure BasicRec that finds all connected, and Berge-acyclic subhypergraphs in $\mathcal{H}$ in depth-first manner. This algorihtm is a simple backtracking algorithm, working as follows. Starting from each singleton subset $\{e\}$ in $\mathcal{I}$, the algorithm searches the family tree $\mathcal{T}$ for connected, and Berge-acyclic subsets by expanding the parent subset S by adding a new leaf f to obtain a child $S' = S \cup \{f\}$. In the expansion, it apply pruning for redundant subsets using the definition of a correct child based on the maximum leaf condition of the child. If expansion is no longer possible, it backtrack to the parent.

To compute the border set, we use the procedure ComputeBorder in Algorithm 2.

Lemma 6. *The algorithm* ComputeBorder *in Fig. 2 computes the border set of an hyperedge subset S $O(||S||) = O(nm)$ time using $O(n)$ additional space.*

We give the time and space complexity of the basic algorithm below.

Theorem 1 (main result). *The algorithm* BergeMine *of Fig. 1 finds all connected Berge-acyclic sub-hypergraphs contained in an input hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ without duplicates in $O(Nm) = O(nm^2)$ delay and $O(N)$ words of space, where $n = |\mathcal{V}|$, $m = |\mathcal{E}|$, and $N = ||\mathcal{E}||$ are the numbers of vertices and hyperedges, and the total size of the hyperedges in $\mathcal{H}$.*

Algorithm 2 The algorithm for computing the border set of a sub-hyupergraph

```
1: procedure ComputeBorder($S$: sub-hypergraph, $\mathcal{V}, \mathcal{E}$)
2:     Output: $Border(S) = \{ f \in (\mathcal{E} \setminus S) \mid cnt(f, S) = 1 \}$.
3:     Mark all vertices of $\mathcal{V}(S)$;
4:     $Border \leftarrow \emptyset$;
5:     for each $e \in \mathcal{E}(S)$ do
6:         Count the number $cnt(e, S)$ of all marked vertices in $e$;
7:         if $cnt(e, S) = 1$ then
8:             $Border \leftarrow Border \cup \{e\}$;
9:     return $Border$;
```

From the theorem, we have the following corollary.

Corollary 1. *The class of all connected Berge-acyclic sub-hypergraphs contained in an input hypergraph $\mathcal{H}$ can be enumerated in polynomial delay and polynomial space in the size of input $\mathcal{H}$.*

4 The Modified Algorithm

In this subsection, we show a modified version of our depth-first mining algorithm that finds all connected, Berge-acyclic sub-hypergraphs in an input hypergraph $\mathcal{H}$ in $O(||S||)$ time using $O(N)$ space and preprocessing, where $N = ||\mathcal{E}(\mathcal{H})||$. This algorithm is *adaptive* since its time complexity only depends on the size of the discovered sub-hypergraph S, rather than the whole input. This adaptivity is quite important in mining a large hypergraph.

The basic idea of our modified algorithm is incremental maintenance of the subset $MaxBorder(S)$ of hyperedge canndidates to insert, called the maximal border hyperedges.

Definition 7. The maximal border of a sub-hypergraph S is the set of hyperedges defined by:

$$MaxBorder(S) = \{ e \in \mathcal{E} \setminus S \mid cnt(e, S) = 1,\ e = \max L(S \cup \{e\}) \}, \qquad (1)$$

that is, $MaxBorder(S)$ consists of all hyperedges e of $\mathcal{H}$ satisfying the next conditions: (i) e is a border of S (i.e., $cnt(e, S) = 1$), and (ii) e is the maximum leaf of $S' = S \cup \{e\}$ among all leaves when it is added to S.

In Algorithm 4, we show our modified depth-first algorithm FastBergeMine as well as its recursive subprocedure FastRec for mining all connected, Berge-acyclic sub-hyupergraphs in incrementally. By using $MaxBorder(S)$, in our depth-first mining algorithm FastBergeMine, we can generate any children $S' = S \cup \{f\}$ from a parent S by just selecting any hyperedge $e \in MaxBorder(S)$ without testing the pruning condition for duplication because the condition is already included by the definition of the maximal border set. In other words, we are eager to make selection of border candidates and test for duplication at the same time in advance.

Therefore, it remains how to efficiently compute the maximal border set. Surprisingly, we can show that this is done in almost optimal time complexity in amortized analysis using a procedure similar to the α-acyclicity test by (Tarjan and Yannakakis [21]). The key to the algorithm is the following recurrence relation for the maximum and the second maximum leaves when we update a parent S by adding a new maximum border $f \in \mathit{MaxBorder}(S)$ to generate a children $S' = S \cup \{f\}$.

Lemma 7. *Let us denote by $max(S)$ and $2max(S)$ the maximum and the second maximum leaves of a parent set $S \subseteq \mathcal{E}$. Then, the maximum and the second maximum leaves $max(S')$ and $2max(S')$ of a child $S' = S \cup \{f\}$ satisfy the following recurrence:*

- *If f connects $\ell_{\max} = \mathit{maxLeaf}(S)$:*
  - *If $f > 2max(S)$, then $max(S') \leftarrow f$ and $2max(S') \leftarrow max(S)$ hold.*
  - *Otherwise, $max(S') \leftarrow max(S)$ and $2max(S') \leftarrow 2max(S)$ hold.*
- *Otherwise:*
  - *If $f > max(S)$, then $max(S') \leftarrow f$ and $2max(S') \leftarrow max(S)$ hold.*
  - *Otherwise, $max(S') \leftarrow max(S)$ and $2max(S') \leftarrow 2max(S)$.*

Proof. In each ease, the proof immediately follows from the case analysis using the definitions of *max*, *2max*, and the maximal border set. ■

From Lemma 7 above, we can update $max(S)$ and $2max(S)$ incrementally in constant time. Now, we show the algorithm UpdateMaxBorder in Algorithm 3 that incrementally updates the new border est $\mathit{MaxBorder}(S \cup \{f\})$ from the older one given the border edge f to add, S, and $\mathit{MB} = \mathit{MaxBorder}(S)$.

For efficient update, the algorithm uses a dynamic data structure $\mathcal{D}$ for storing a set $\mathcal{D}$ of candidate hyperedges, which is similar to the *DLX* (also known as "*Dancing Links*") data structure by Knuth [14] as described in Sec. 2. For complexity analysis, recall that $||N(S)||$ denotes the sum $\sum_{e \in S} |N(e)|$ of neghbor hyperedges to S, where $||S|| + ||N(S)|| = O(||\mathcal{E}||) = O(nm)$. Then, we have the next lemma.

Lemma 8. *Let $S \subseteq \mathcal{E}$ be a sub-hypergraph and $f \in R = (\mathcal{E} \setminus S)$ be a maximum border hyperedge of S. Given f, B, and R, the algorithm* UpdateMaxBorder *in Algorithm 3 computes the set $\mathit{MaxBorder}(S \cup \{f\})$ of all maximum border hyperedges of $S' = S \cup \{f\}$ in $O(||S|| + ||N(S)||)$ amortized time using $O(N)$ space and $O(N)$ preprocessing (at once in the initialization), where $B = \mathit{MaxBorder}(S)$ is the set of all maximum borders of S, and $N = ||\mathcal{E}(\mathcal{H})||$.*

Proof. Consider Algorithm 3. During the computation of the recursive mining procedure, we maintain the pointers $max(S)$, $2max(S)$, and the dynamic data structure $\mathcal{D}$. From Lemma 7, Step 1 correctly updates the maximum and 2nd maximum leaves in S in constant time. When a new border f is added to S, the only borders to be changed are (i) f is removed, and (ii) all neiboring hyperedges

Algorithm 3 The algorithm for computing the border set of a sub-hyupergraph

```
1: procedure UpdateMaxBorder($S$, $f$: hyperedge, $B, R \subseteq \mathcal{E}(\mathcal{H})$, $\mathcal{H}$. ), where $\mathcal{D}$ is a
   dynamic data structure for storing a hyperedge subset $\mathcal{D}$ in linear space supporting
   membership, insert, and delete in sublinear time $t = f(m)$.
   Pre-conditions: $S' = S \cup \{f\}$, $f \in R$, $B = MB(S)$, and $R = \mathcal{E}(\mathcal{H}) \setminus S$.
   Output: $MB(S') = \{ f \in (\mathcal{E} \setminus S') \mid cnt(f, S') = 1, f = \max L(S') \}$.
2:     //Step 1: Update the maximum leaves.
3:     $S' = S \cup \{f\}$;
4:     if $f$ connects $\ell_{\max} = maxLeaf(S)$ then update by:                ▷ in $O(1)$ time
5:         – Set $max(S') \leftarrow f$ and $2max(S') \leftarrow max(S)$ if $f > 2max(S)$.
6:         – Set $max(S') \leftarrow max(S)$ and $2max(S') \leftarrow 2max(S)$ otherwise.
7:     else update by:                                                   ▷ in $O(1)$ time
8:         – $max(S') \leftarrow f$ and $2max(S') \leftarrow max(S)$ if $f > max(S)$.
9:         – $max(S') \leftarrow max(S)$ and $2max(S') \leftarrow 2max(S)$ otherwise.
10:    end if
11:    //Step 2: Update the maximum border set.
12:    $MB(S') \leftarrow \emptyset$;
13:    //Step 2.1: Existing borders other than $f$
14:    for each $e$ in $MB(S) \setminus \{f\}$ do
15:        Add $e$ to $MB(S')$ if $e > max(S')$.          ▷ Charge $O(|MB(S)|)$ time to $S$
16:    //Step 2.2: New borders connecting to $f$
17:    for each vertex $x \in f$ do                           ▷ Charge $O(f)$ time to $S'$
18:        for each hyperedge id $e \in N(x, \mathcal{D})$ do    ▷ Charge $O(1)$ time to $e$ in $\mathcal{D}$
19:            $cnt[e] \leftarrow cnt[e] + 1$;
20:            if $cnt[e] = 1$ then                         ▷ $cnt$ increased from 0 to 1
21:                Add $e$ to the candidate set $\mathcal{D}$;      ▷ Charge $O(f(n))$ time to $e$
22:                Add $e$ to $MB(S')$ if $e > max(S')$.
23:            else if $cnt[e] = 2$ then                    ▷ $cnt$ increased from 1 to 2
24:                Remove $e$ from candidate set $\mathcal{D}$;     ▷ Charge $O(f(n))$ time to $e$
25:        //Note: each hyperedge is processed at most twice overall
26:    end for
27:    return $MB$;
```

of f, and (ii) all existing border edges that has a non-empty intersection to f other than its connection point to S. Step 2 handles these cases correctly. For time analysis of Step 2, we observe that during computation from the root hypothesis $\emptyset$ to the current set S, any hyperedge e in $\mathcal{H}$ will be processed at most twice after initialization, that is, it is incremented to $cnt(e) = 1$ at the first time, and it is incremented to $cnt(e) = 2$ the second time. Then, it is removed from $\mathcal{D}$ forever (othewise a back tracking occurs). By using appropriate charging scheme of the cost to each edges in S, we can show that the amortized cost for Step 2 to obtain each S is at most $||S|| + ||N(S)|| = O(nm) = O(N)$, where $||S|| = \sum_{e \in S} |e|$. ■

From Lemma 8, we show the main theorem of this paper.

Algorithm 4 The modified algorithm FastBergeMine for mining all connected, Berge-acyclic sub-hyupergraphs based on the reverse search

```
1: procedure FastBergeMine($\mathcal{H} = (\mathcal{V}, \mathcal{E})$: input hypergraph )
2:     for each hyperedge $e \in \mathcal{E}(\mathcal{H})$ do
3:         $MB_f \leftarrow \{ f \in \mathcal{E}(\mathcal{H}) \mid (|f \cap e| = 1) \}$;
4:         $R_f \leftarrow \mathcal{E}(\mathcal{H}) \setminus \{f\}$;
5:         FastRec($\{e\}, MB_f, R_f, \mathcal{H}$);
6: procedure FastRec($S, MB, R \subseteq \mathcal{E}(\mathcal{H})$, $\mathcal{H}$: hypergraph)
7:     Invariant: $MB = MaxBorder(S)$ and $R = \mathcal{E}(\mathcal{H}) \setminus S$ hold.
8:     Output $S$;
9:     for each border hyperedge $f \in MB$ do            ▷ Generation of children
10:        $S' \leftarrow S \cup \{f\}$;
11:        $R' \leftarrow R \setminus \{f\}$;
12:        Incrementally compute $MB' = MaxBorder(S', \mathcal{H})$ from $f$, $MB$, and $R$;
13:        FastRec($S', MB', R', \mathcal{H}$);
```

Theorem 2 (The adaptive delay by the modified mining algorithm). *The algorithm* FastBergeMine *of Fig. 4 finds all connected Berge-acyclic sub-hypergraphs contained in an input hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ without duplicates in $O(||S|| + ||N(S)||)$ delay (time per solution) using $O(N)$ space and $O(N)$ preprocessing, where $||S|| = O(nm)$ is the total length of hyperedges in S, and $N = ||\mathcal{E}||$ are the numbers of vertices and hyperedges, and the total size of the hyperedges in $\mathcal{H}$.*

Proof. The correctness of the algorithm FastBergeMine is obvious from that of the basic algorithm and the definition of *MaxBorder*. Lemma 8, the time complexity follows. ■

From the theorem, we have the following corollary.

Corollary 2. *The class of all connected Berge-acyclic sub-hypergraphs contained in an input hypergraph $\mathcal{H}$ can be enumerated in delay that depends only on the size $||S||$ of a discovered subset S using polynomial space and preprocessing in the input size $||\mathcal{E}(\mathcal{H})||$.*

5 Conclusion

In this paper, we considered the problem of finding all all connected Berge-acyclic sub-hypergraphs contained in an input hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ without duplicate with applications to generalization of itemset mining from transaction databases and also discovering connected substructures from datasets in the form of sets of sets. As main results, we presented an efficient DFS algorithm for the problem that achieves polynomial delay and space complexity. We also presented an improved algorithm that has adaptive delay depending only on the size of discovered sub-hypergraph.

In this paper, we focused on only the theoretical aspect of the problem. One of the most important future researches is implementation and empirical evaluation of the proposed algorithms on artificial and real datasets. It is also an important problem to find suitable application of this problem in knowledge discovery problems in the real world including knowlege discovery from mobility data or social networks. We want to study these problems in future.

References

1. H. Arimura and T. Uno. An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. *Journal of Combinatorial Optimization*, 13:243–262, 2006.
2. H. Arimura and T. Uno. Mining maximal flexible patterns in a sequence. In *New Frontiers in Artificial Intelligence*, LNCS 4914, pages 307–317, 2007.
3. H. Arimura and T. Uno. Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In *Proceedings of the SIAM Int'l Conf. on Data Mining 2009 (SDM'09)*, pages 1087–1098, 2009.
4. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Math.*, 65:21–46, 1993.
5. P. Beame and F. E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38 – 72, 2002.
6. C. Berge and E. Minieka. *Graphs and hypergraphs*, volume 7. North-Holland, 1973.
7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2 edition, 2001.
8. T. Daigo and K. Hirata. On Generating All Maximal Acyclic Subhypergraphs with Polynomial Delay. *SOFSEM 2009: Theory and Practice of Computer ...*, pages 181–192, 2009.
9. R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, July 1983.
10. R. Ferreira, R. Grossi, and R. Rizzi. Output-sensitive listing of bounded-size trees in undirected graphs. *Algorithms ESA 2011*, pages 275–286, 2011.
11. K. Hirata, M. Kuwabara, and M. Harao. On finding acyclic subhypergraphs. *Fundamentals of Computation Theory*, pages 491–503, 2005.
12. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. PKDD'00, LNCS 1910*, pages 13–23. Springer, 2000.
13. S. KAWASOE, H. SAKAMOTO, H. ARIMURA, and S. ARIKAWA. Efficient substructure discovery from large semi-structured data. *IEICE TRANSACTIONS on Information and Systems*, 87(12):2754–2763, 2004.
14. D. E. Knuth. Dancing links. *eprint arXiv:cs/0011047*, Nov. 2000.
15. T. Kuboyama, K. Hirata, and K. F. Aoki-Kinoshita. An efficient unordered tree kernel and its application to glycan classification. In *Proc. PAKDD'08, LNCS 5012*, pages 184–195. Springer, 2008.
16. X.-L. Li, S.-H. Tan, C.-S. Foo, S.-K. Ng, et al. Interaction graph mining for protein complexes using local clique merging. *Genome Informatics*, 16(2):260, 2005.
17. L. Lovász. Matroid matching and some applications. *Journal of Combinatorial Theory, Series B*, 236:208–236, 1980.

18. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE TKDE*, 16(11):1424–1440, 2004.
19. A. Shioura, A. Tamura, and T. Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.*, 26(3):678–692, 1997.
20. A. Silva, W. Meira Jr, and M. J. Zaki. Structural correlation pattern mining for large graphs. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pages 119–126. ACM, 2010.
21. R. Tarjan and M. Yannakakis. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM Journal on computing*, 13(3), 1984.
22. R. E. Tarjan and R. C. Read. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.
23. T. Uno and H. Arimura. Ambiguous frequent itemset mining and polynomial delay enumeration. In *Proc. PAKDD2008, LNAI 5012*, pages 357–368. Springer, 2008.
24. T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Proc. the 7th Discovery Science (DS'04)*, volume 3245 of *LNCS*, pages 16–31. Springer, 2004.
25. K. Wasa, Y. Kaneta, T. Uno, and H. Arimura. Constant time enumeration of bounded-size subtrees in trees and its application. In *Computing and Combinatorics*, pages 347–359. Springer, 2012.
26. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.
27. M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, May/Jun 2000.
28. M. J. Zaki and C.-J. Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):462–478, 2005.