Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Scalable Algorithm for Higher-Order Co-Clustering via Random Sampling

Daisuke Hatano,† Takuro Fukunaga,† Takanori Maehara,‡ Ken-ichi Kawarabayashi†
†National Institute of Informatics, Japan, JST, ERATO, Kawarabayashi Large Graph Project, Japan
{hatano, takuro, k_keniti}@nii.ac.jp
‡Shizuoka University, Japan
maehara.takanori@shizuoka.ac.jp

Abstract

We propose a scalable and efficient algorithm for co-clustering a higher-order tensor. Viewing tensors with hypergraphs, we propose formulating the co-clustering of a tensor as a problem of partitioning the corresponding hypergraph. Our algorithm is based on the random sampling technique, which has been successfully applied to graph cut problems. We extend a random sampling algorithm for the graph multiway cut problem to hypergraphs, and design a co-clustering algorithm based on it. Each iteration of our algorithm runs in polynomial on the size of hypergraphs, and thus it performs well even for higher-order tensors, which are difficult to deal with for state-of-the-art algorithm.

1 Introduction

Clustering is the task of grouping together similar objects, and it is one of the main tasks in machine learning and data mining. It is also considered to be an important technique in statistics, bioinformatics, database, pattern recognition, among others. A starting point for a clustering algorithm would apply to "matrix," because the most basic setting of data is represented by a matrix (e.g., words and documents, click and query in a search engine).

In a matrix, it is sometimes convenient to allow simultaneous clustering of rows and columns. Consider a matrix with the rows corresponding to words and the columns corresponding to documents. It has been observed that words appearing in two documents on similar topics should have similar concepts. Conversely, documents including words of similar concepts should cover similar topics. Therefore, it is useful for precise clustering to group both words and documents simultaneously. This clustering is called *biclustering* and was originally introduced by Hartigan (1972). Formally, given a set of n_1 rows in n_2 columns of a matrix, biclustering is the task of generating a subset of rows with similar behaviors across a subset of columns, or vice versa. Biclustering has several equivalent names in different areas, including co-clustering, two-order clustering, and block clustering.

Recently, data analysis has developed to need not only two-dimensional data but also higher-dimensional data. These higher-dimensional structures are often encoded in *tensors*, i.e., multi-dimensional matrices, which are becoming increasingly important in machine learning and data mining. The main challenges are the following: most of the problems for tensor computations are significantly harder than their matrix counterparts. For example, eigenvectors are a key concept for spectral clustering and it is not hard to find them for matrices. However, finding tensor eigenvectors is NP-hard (Hillar and Lim 2013).

A matrix can be viewed as a bipartite graph with the bipartition (V_1, V_2) of the vertex set, where V_1 and V_2 correspond to rows and columns, respectively. Therefore, it makes sense to apply some graph algorithms to matrices for biclustering. Indeed, several works have adapted this approach (Dhillon 2001; Wieling and Nerbonne 2011; Zha et al. 2001).

Viewing order-m tensors with m-uniform hypergraphs, we propose a hypergraph-based approach for co-clustering[1]. We formulate the co-clustering of a tensor as a problem of partitioning the corresponding hypergraph. Then, we find a k-way cut of the hypergraph by an algorithm based on *random sampling* (or random contraction) technique, which was introduced by Karger (1993) for the minimum cut problem of graphs, and was extended by Karger and Stein (1996) for the *k-way cut* problem. A k-way cut is an edge set that partitions a given (hyper)graph into k parts. Karger and Stein (1996) introduced a random sampling technique for a minimum cut computation of a given undirected graph. Simultaneously, they show that this technique can also be applied for computing a minimum k-way cut of a graph. Their algorithm iteratively contracts randomly sampled edges until only k vertices remain. These k vertices represent a k-way cut in the original graph. Karger and Stein proved that this random sampling approach leads to finding a minimum k-way cut with high probability, provided that we take $O(n^{2k-1})$ trials from this graph with n vertices. Karger and Stein's technique demonstrates the power of randomized algorithms, and their algorithm leads to an important building block for development of numerous randomize algorithms.

Karger and Stein's algorithm only deals with graphs, but this is not enough for our purpose. Thus, we extend their

[1]Biclustering is meant for only two dimensional structures (i.e., matrix). For higher dimensional structures, we shall use "co-clustering."

algorithm to hypergraphs. Based on this algorithm, we propose a new algorithm for co-clustering higher-order tensors. In summary, our contributions can be summarized as follows:

1. We consider the co-clustering problem for m-uniform hypergraphs (and hence, order-m tensors) as a problem of finding a hypergraph k-way cut.
2. We generalize the applicability of Karger and Stein's algorithm from graphs to hypergraphs.
3. We then present our co-clustering algorithm, which is based on Karger and Stein's algorithm for hypergraphs.
4. We adapt a few heuristics, which provides better performance for experiments.
5. We empirically show that our algorithm provides a better partition as the one given by state-of-the-art algorithms.
6. Finally, we empirically show that our algorithm is quite fast and scales for a large tensor.

For the computational issues, let us observe that each iteration of our algorithm samples $O(n)$ hyperedges, where n is the number of vertices (total number of dimensions in the corresponding tensor). As a theoretical result, we prove that $O(n^{mk})$ iterations suffice for computing a minimum k-way cut of a m-uniform hypergraph. However, we believe that $\Omega(n^{mk})$ iterations are not required in practice; in experiments, we observed that 1000-10000 iterations are enough for achieving enough quality, even for large hypergraphs. Since iterations of our algorithm can be executed independently, we can adjust the number of iterations when the computation time is restricted. Moreover, it is easy to parallelize our algorithm. These are useful features in practice. The runtime of state-of-the-art algorithms is an exponential of m for order-m tensors. In contrast, the runtime of each iteration in our algorithm is $O(nmp)$ for order-m tensors with n dimensions and p non-zero elements, and thus it performs well even for higher-order tensors. Therefore, our algorithm is much faster than any other state-of-the-art algorithms when we co-clustering higher-order tensors. This allows our algorithm to scale up to tensors with 0.2 billion nonzero elements, a feature not previously handled by any other algorithm.

The rest of this paper is organized as follows. Section 2 surveys related work, and Section 3 defines notations. Section 4 introduces Karger and Stein's algorithm for hypergraph k-way cut problem, and analyzes it. Section 5 presents our co-clustering algorithm, and Section 6 evaluates it through computational experiments.

2 Related work

Biclustering of data represented by a matrix has been studied since the 1970s (Hartigan 1972). There has been a huge number of algorithms proposed so far, and we name a few of them (Zha et al. 2001; Dhillon, Mallela, and Modha 2003; Shan and Banerjee 2008; O'Connor and Feizi 2014). These algorithms have been successfully applied to numerous unsupervised learning tasks (Dao et al. 2010; Zhu, Yang, and He 2015; Chen et al. 2015). In particular, clustering on gene expression data (Cheng and Church 2000; Madeira et al. 2010; Hussain 2011) and document classification (Dhillon 2001; Bisson and Hussain 2008; Hussain, Bisson, and Grimal 2010) are studied actively.

Compared with the biclustering of matrix data, co-clustering of higher-order tensors has not been extensively studied so far. Zhao and Zaki (2005) considered biclustering on three-dimensional microarray data, and proposed a graph-based algorithm. Other previous studies depend on tensor factorizations. Zhou et al. (2009) applied PARAFAC tensor decomposition for biclustering a data representing the records of web page visitors by a three-dimensional tensor. Peng and Li (2011) proposed PARATUCKER factorization, which combines the PARAFAC and Tucker factorizations, and applied it to biclustering. Papalexakis et al. (2013) formulated biclustering as a constrained multilinear decomposition problem and proposed an algorithm based on it.

As mentioned in Section 1, we formulate the co-clustering problem as a k-way cut problem in a hypergraph and apply the random sampling technique. The random sampling technique was introduced by Karger (1993) for the minimum cut problem of graphs, and was extended by Karger and Stein (1996) for the k-way cut problem for graphs. Later, Karger (2000) speeded up the minimum cut algorithm by combining random sampling and a tree-packing approach. Thorup (2008) presented an algorithm for the k-way cut problem. His algorithm is deterministic and is based on tree-packing. Thorup's algorithm was extended by Fukunaga (2013) for hypergraphs.

3 Preliminary

For an integer i, let $[i]$ be a set $\{1, 2, ..., i\}$. Let $A \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_m}$ be an order-m tensor. In other words, each element of A is specified by m indices $j_1, \ldots, j_m$, and the i-th index j_i ranges from 1 to n_i for each $i \in [m]$. $A(j_1, \ldots, j_m)$ denotes the element of A specified by the indices $j_1, \ldots, j_m$. In this study, we assume that each element is non-negative.

Let $G = (V, E)$ be the hypergraph with a vertex set V and a hyperedge set E. Each hyperedge consists of at least two vertices. The size of a hyperedge is called its *rank*. The rank of a hypergraph G is defined as the maximum rank of hyperedges in G. We denote the rank of a hypergraph G by m_G.

For a set F of hyperedges, we let $G - F$ and $G \backslash F$ respectively denote the hypergraphs obtained from G by deleting and contracting hyperedges in F. Here, contracting a hyperedge e stands for an operation whereby all vertices included in e are identified with a single new node.

A k-partition of the vertex set V denotes the set $\mathcal{U} := \{U_1, \ldots, U_k\}$ of k nonempty subsets of V such that $\bigcup_{i=1}^{k} U_i = V$ and $U_i \cap U_j = \emptyset$ for $1 \leq i < j \leq k$. For a k-partition $\mathcal{U}$, let $\delta(\mathcal{U})$ denote the set of hyperedges in E that intersect at least two members of $\mathcal{U}$.

A k-way cut of a hypergraph $G = (V, E)$ is a subset F of E such that $G - F$ has at least k connected components. If $\mathcal{U}$ is a k-partition of the vertex set V, then $\delta(\mathcal{U})$ is an inclusion-wise minimal k-way cut of the hypergraph G. Conversely,

Algorithm 1 Hypergraph version of Karger and Stein's algorithm

Input: a hypergraph $G = (V, E)$, nonnegative hyperedge weights $w\colon E \to \Re$, and an integer k
Output: a k'-way cut of G for some $k' \in \{k, \ldots, k+m_G-1\}$

1: **while** $|V| \geq k + m_G$ **do**
2: pick a hyperedge e in G with the probability $w(e)/\sum_{e' \in E(G)} w(e')$
3: $G \longleftarrow G \backslash e$, and remove all hyperedges of rank one from G
4: **end while**
5: output the set of hyperedges remaining in G

for any inclusion-wise minimal k-way cut F of G, there exists a k-partition $\mathcal{U}$ of V such that $F = \delta(\mathcal{U})$. Thus, when we discuss the problem of finding a minimum weight k-way cut of a hypergraph, k-way cuts and k-partitions of the vertex set can be identified.

4 Random sampling algorithm for hypergraph k-way cut

In our co-clustering algorithm, co-clustering of a tensor is identified with a k-way cut (or equivalently a k-partition of the vertex set) of the hypergraph constructed from the tensor in a certain way. To find an appropriate co-clustering, our algorithm adopts the random sampling technique, which was originally proposed by Karger and Stein (1996) for computing a minimum k-way cut of a graph. In this section, we show that their algorithm can be extended to k-way cuts in hypergraphs.

Algorithm

In the hypergraph k-way cut problem, we are given a hypergraph $G = (V, E)$ with a nonnegative weight $w(e)$ of each hyperedge $e \in E$. The problem seeks a k-way cut F of G that minimizes $\sum_{e \in F} w(e)$. By the equivalence between an inclusion-wise minimal k-way cut and a k-partition of V mentioned in Section 3, the problem is equivalent to finding a k-partition $\mathcal{U}$ of V that minimizes $\sum_{e \in \delta(\mathcal{U})} w(e)$.

In the setting discussed below, we allow an algorithm to output a partition of size between k and $k + m_G - 1$. This gives no issue because the size of the co-clustering is not known precisely in practice.

Hypergraph version of Karger and Stein's k-way cut algorithm iterates sampling a hyperedge in a given hypergraph. The probability that a hyperedge e is sampled is defined as $w(e)/\sum_{e' \in E} w(e')$. If hyperedge e is sampled, the algorithm contracts it and proceeds to the next iteration. The algorithm repeats this until the number of vertices becomes smaller than $k + m_G$. The precise description of the algorithm can be found in Algorithm 1.

Contracting a single hyperedge decreases the number of vertices by at least one and at most $m_G - 1$. Therefore, Algorithm 1 has at most $O(|V| - k)$ iterations.

Success probability of Algorithm 1

Algorithm 1 is a randomized Monte Carlo algorithm. In the rest of this section, we show that Algorithm 1 outputs a k'-way cut of small weight with a high probability. Throughout this subsection, we let $G = (V, E)$ be the input hypergraph with hyperedge weights w, and denote $|V|$ and $\sum_{e \in E} w(e)$ by n and W, respectively. Moreover, let W^* denote the minimum weight of the k-way cuts of G.

First, we analyze the minimum weight of k-way cuts.

Lemma 1.

$$W^* \leq \left(1 - \frac{\binom{n-m_G}{k-1}}{\binom{n}{k-1}}\right) W.$$

Proof. Choose $k-1$ vertices $v_1, v_2, \ldots, v_{k-1}$ uniformly at random from V. Then $\mathcal{U} := \{\{v_1\}, \{v_2\}, \ldots, \{v_{k-1}\}, V \setminus \{v_1, v_2, \ldots, v_{k-1}\}\}$ is a k-partition of V. A hyperedge e is contained by $\delta(\mathcal{U})$ if and only if $e \cap \{v_1, v_2, \ldots, v_{k-1}\} \neq \emptyset$. Hence, we have $\Pr[e \notin \delta(\mathcal{U})] \geq \binom{n-|e|}{k-1}/\binom{n}{k-1} \geq \binom{n-m_G}{k-1}/\binom{n}{k-1}$. Thus,

$$\Pr[e \in \delta(\mathcal{U})] \leq 1 - \frac{\binom{n-m_G}{k-1}}{\binom{n}{k-1}}.$$

It follows from this that

$$\begin{aligned} E\left[\sum_{e \in \delta(\mathcal{U})} w(e)\right] &= \sum_{e \in E} w(e) \Pr[e \in \delta(\mathcal{U})] \\ &\leq \sum_{e \in E} w(e) \left(1 - \frac{\binom{n-m_G}{k-1}}{\binom{n}{k-1}}\right) \\ &= \left(1 - \frac{\binom{n-m_G}{k-1}}{\binom{n}{k-1}}\right) W. \end{aligned}$$

$\mathcal{U}$ may not be a minimum k-way cut, but its weight $\sum_{e \in \delta(\mathcal{U})} w(e)$ is an upper bound on W^*. Therefore, the proof is completed. □

For notational convenience, we denote $\binom{n-m}{k-1}/\binom{n}{k-1}$ by $\epsilon_{n,m}$. We say that, for $\alpha \geq 1$, a k-way cut is *α-minimum* if its weight is at most αW^*. Note that a minimum k-way cut is 1-minimum. We say that a k-way cut F *survives* if Algorithm 1 outputs a k'-way cut F' such that $F \subseteq F'$. In the following theorem, we analyze the probability that an α-minimum k-way cut survives.

For $\alpha \geq 1$, define $g(\alpha)$ as

$$\alpha - \frac{\alpha - 1}{\epsilon_{m_G+k,m_G}} = \alpha - \frac{\alpha - 1}{k}\binom{m_G + k}{k}.$$

Since ϵ_{n,m_G} is increasing as n increases, $g(\alpha)$ is at most $\alpha - (\alpha - 1)/\epsilon_{n,m_G}$ for any $n \geq m_G + k$.

Theorem 1. *Let F be a minimal k-way cut. Let α be a real number at least 1 such that $g(\alpha)$ defined from α is positive. If F is α-minimum for some $\alpha \geq 1$, the probability that F survives is at least $\Omega(g(\alpha)n^{-m_G(k-1)})$.*

Proof. Below, we assume that $m_G \leq |V|-k+1$ holds. This condition can be assumed without loss of generality because if the rank of a hyperedge e is larger than $|V|-k+1$, then e belongs to any k-way cuts, and hence it suffices to choose e into a solution and solve the problem in the sub-hypergraph obtained by deleting e. Let $\tilde{W}$ denote the weight of F. Let n' denote the number of vertices in G', and let W' denote the total weight of hyperedges remaining in G'.

For notational convenience, we let m denote m_G. Notice that $m_{G'} \leq m$. Hence the weight of the minimum k-way cuts of G' is at most $(1-\epsilon_{n'm_{G'}})W' \leq (1-\epsilon_{n'm})W'$ by Lemma 1. If all hyperedges in F remain in G', then F is an α-minimum k-way cut of G'. Hence, $\tilde{W} \leq \alpha(1-\epsilon_{n'm})W'$ in this case. Provided that all edges in F remain in G', the probability that Algorithm 1 chooses a hyperedge outside F in this iteration is at least

$$1-\frac{\tilde{W}}{W'} \geq \alpha\epsilon_{n'm} - (\alpha-1) \geq g(\alpha)\epsilon_{n'm}.$$

Hence, the probability that Algorithm 1 chooses no hyperedge in F in any iteration is at least

$$g(\alpha)\prod_{n'=k+m}^{n}\frac{\binom{n'-m}{k-1}}{\binom{n'}{k-1}} = \Omega(g(\alpha)n^{-m(k-1)}).$$

□

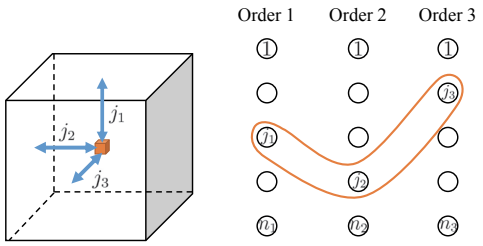
By Theorem 1, if we apply the algorithm $\Omega(n^{m_G(k-1)}/g(\alpha))$ times, the α-minimum k-way cut F survives in at least one trial with a constant probability.

$g(\alpha)$ may seem a mysterious number, but it is a small constant in many cases. For example, let us consider the case with $\alpha = 1.05$, $m_G = 3$, and $k = 5$; as we see in the following section, this corresponds to the co-clustering of an order-3 tensor into 5 co-clusters with allowing 5% fitting error. In this case, $g(\alpha) = 1.05-(1.05-1)/5\times\binom{5+3}{3} = 0.49$.

5 Proposed co-clustering algorithm

Co-clustering as a hypergraph k-way cut

We define a hypergraph from a given tensor $A \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_m}$ as follows. For each $i \in [m]$, let V_i be a set of n_i vertices, each of which corresponds to a dimension of the i-th order of A. We denote by v_j the vertex corresponding to the dimension j. The hypergraph is defined over the vertex set $V := \bigcup_{i\in[m]} V_i$. It contains a hyperedge for each non-zero element of A; if an element $A(j_1,\ldots,j_m)$ is non-zero, the corresponding hyperedge consists of $v_{j_1},\ldots,v_{j_m}$, and the weight associated with it is defined as $A(j_1,\ldots,j_m)$. For example, Figure 1 illustrates the transformation of a tensor to a hypergraph. Here, the (j_1,j_2,j_3)-th element of the order-3 tensor is regarded as a hyperedge (denoted by the orange line) including vertices j_1, j_2, and j_3. In the rest of this paper, we let $G = (V,E)$ denote the resulting hypergraph and w denote the weights associated with the hyperedges. We note that G is an m-uniform hypergraph over $\sum_{i\in[m]} n_i$ vertices, and the number of hyperedges in it is equal to the number of non-zero elements of A.



(a) Order-3 tensor (b) 3-uniform hypergraph

Figure 1: Definition of the 3-uniform hypergraph from an order-3 tensor

A partition of the vertex set of hypergraph G can be identified with the co-clustering of a tensor A. Namely, if $\mathcal{U}$ is a partition of V, then the clustering of the i-th order dimensions is defined from $\mathcal{U}$ as $\mathcal{U}_i := \{U\cap V_i \colon U \in \mathcal{U}\}$. Thus, we formulate the co-clustering problem as the problem of finding a k-partition of a hypergraph.

For an integer k, let $\mathcal{P}_k$ be the family of all k-partitions of the vertex set V. We define two loss functions $f_{\text{cut}}, f_{\text{equal}} \colon \mathcal{P}_k \to \mathbb{R}_+$. Roughly speaking, f_{cut} represents how the co-clustering represented by the k-partition fits into the given tensor data; as $f_{\text{cut}}(\mathcal{U})$ is smaller, the k-partition $\mathcal{U}$ represents a co-clustering fitting into the tensor better. However, a k-partition minimizing f_{cut} is often unbalanced, which is not desirable in practice. Hence we use the other loss function f_{equal}, which represents how balanced the k-partition is. Then, for a parameter $\theta \geq 0$, we formulate the co-clustering problem as

$$\begin{array}{ll}\text{minimize} & f_{\text{equal}}(\mathcal{U}) \\ \text{subject to} & f_{\text{cut}}(\mathcal{U}) \leq \theta, \\ & \mathcal{U} \in \mathcal{P}_k.\end{array} \tag{1}$$

In what follows, we introduce the functions f_{cut} and f_{equal}.

Cut function The cut function denoted by f_{cut} returns the total weight of hyperedges in $\delta(\mathcal{U})$ for a k-partition $\mathcal{U}$. Namely, $f_{\text{cut}}(\mathcal{U}) = \sum_{e\in\delta(\mathcal{U})} w(e)$ holds for all $\mathcal{U} \in \mathcal{P}_k$.

Let us explain an intuition behind this loss function. Let $\mathcal{U} = \{U_1,\ldots,U_k\}$ be a k-partition. As mentioned above, we view $\mathcal{U}$ as a co-clustering of tensor A. We observe that each hyperedge in $\delta(\mathcal{U})$ corresponds to a non-zero element whose indices belong to more than one cluster of $\mathcal{U}$. If $\mathcal{U}$ is the ground truth co-clustering, each $U_i \in \mathcal{U}$ consists of vertices whose corresponding dimensions are highly correlated. We represent this by the condition that tensor A has many large-value elements whose indices are included in single clusters, and the values of elements whose indices are included in more than one cluster are relatively small. We note that $f_{\text{cut}}(\mathcal{U})$ is equal to the sum of the values of the latter elements. Therefore, a k-partition minimizing f_{cut} can be expected to be a good co-clustering.

By the equivalence between the k-partitions and the inclusion-wise minimal k-way cuts, the optimization problem (1) with the loss function f_{cut} is equivalent to the hy-

Algorithm 2 Co-clustering algorithm

Input: a hypergraph $G = (V, E)$ with nonnegative hyperedge weights $w\colon E \to \mathbb{R}$, an integer k, the number of iterations l, and a positive threshold $\theta \in \mathbb{R}$.
Output: a k-partition $\mathcal{U}$

1: initialize $\mathcal{U}^*$ by an arbitrary k-partition
2: **for** $i = 1$ to l **do**
3: $\quad \mathcal{U} \leftarrow \mathrm{KS}(G, w, k)$
4: $\quad$ **if** $f_{\mathrm{cut}}(\mathcal{U}^*) > \theta \geq f_{\mathrm{cut}}(\mathcal{U})$ or $f_{\mathrm{equal}}(\mathcal{U}^*) > f_{\mathrm{equal}}(\mathcal{U})$ **then**
5: $\quad\quad \mathcal{U}^* \longleftarrow \mathcal{U}$
6: $\quad$ **end if**
7: **end for**
8: return $\mathcal{U}^*$

pergraph k-way cut problem. Therefore, Algorithm 1 can be used for finding a k-partition minimizing f_{cut}.

Equal-size function Although minimizing f_{cut} makes each cluster correlated well, it often gives many meaningless clusters. For example, when a hypergraph is dense, the minimizer of f_{cut} is likely to be a partition mostly made up of singletons.

To avoid this phenomenon, we consider another loss function, f_{equal}, which balances the sizes of clusters. Here, f_{equal} is defined by $f_{\mathrm{equal}}(\mathcal{U}) = \sum_{i\in[k]} |U_i|^2$ for each $\mathcal{U} \in \mathcal{P}_k$. We note that if a k-partition $\mathcal{U} = \{U_1, \ldots, U_k\}$ is a minimizer of f_{equal} and $n \bmod k \equiv 0$, then $|U_1| = \cdots = |U_k|$.

Algorithm

Our algorithm is based on the Karger and Stein's random sampling algorithm given in Algorithm 1. As shown in Theorem 1, Algorithm 1 outputs a k-partition $\mathcal{U}$ with small $f_{\mathrm{cut}}(\mathcal{U})$ with high probability. Hence, our algorithm applies Algorithm 1 a certain number of times to sample a set of k-partitions. If at least one k-partition $\mathcal{U}$ such that $f_{\mathrm{cut}}(\mathcal{U}) \leq \theta$ is sampled, our algorithm outputs the one minimizing f_{equal} among such k-partitions. If Algorithm 1 samples no such a k-partition, the proposed algorithm outputs the one with minimum f_{equal} value among all sampled k-partitions. See Algorithm 2 for precise description of the proposed algorithm. Here, $\mathrm{KS}(G, w, k)$ denotes the output of Algorithm 1 when a hypergraph G, hyperedge weights w, and a positive integer k are given.

The following analysis on Algorithm 2 is an easy corollary of Theorem 1. The notations n, m_G, W^*, and $g(\alpha)$ are defined as in Section 4.

Theorem 2. *Let α be a real number at least 1 such that $g(\alpha)$ defined from α is positive. If $\theta \leq \alpha W^*$ and $l = \Omega(n^{m_G(k-1)}/g(\alpha))$, then Algorithm 2 outputs an optimal solution of* (1) *with a constant probability.*

In addition, for obtaining more reasonable co-clusterings, we introduce heuristics, so-called *the distorted sampling heuristic* and *the balancing merge heuristic*, into Algorithm 1. We explain these heuristics below. In these modified versions, $\mathrm{KS}(G, w, k)$ in Algorithm 2 is replaced by the output of Algorithm 1 with these heuristics.

Distorted sampling heuristic In sampling a hyperedge $e \in E$, we wish to prohibit the case where two large-size co-clusters in the current partition $\mathcal{U}$ are merged by the contraction of e. To this end, depending on the size of co-clusters intersected by e, we cancel contracting e, and then resample another hyperedge in the same manner of Step 2 of Algorithm 1. Let $\mathcal{U}_e$ denote $\{U \in \mathcal{U}\colon e \cap U \neq \emptyset\}$. When we cancel a hyperedge e, we define the probability $p_{\mathrm{ca}}(e)$ to cancel the hyperedge e so that it is proportional to

$$1 - \frac{1}{|\mathcal{U}_e|} \sum_{U\in\mathcal{U}_e} \frac{1}{\log(|U| + \max\{1, |U| - |V|/k\})}.$$

This cancellation is recursively applied to a resampled hyperedge. This definition implies that a hyperedge e is canceled in higher probability as the average size of co-clusters in $\mathcal{U}_e$ is larger. Due to this heuristic, Algorithm 2 tends to produce a partition $\mathcal{U}^*$ minimizing the function f_{equal}.

Balancing merge heuristic In Algorithm 1, a particular k-way cut survives with higher probability in earlier iterations. Hence, we introduce a new parameter $\gamma \geq k + m_G$, and stop the iterations (Steps 1–4) of Algorithm 1 when the number of vertices becomes smaller than γ. Let $\mathcal{U} = \{U_1, \ldots, U_{k'}\}$ be the partition when iterations of Algorithm 1 terminate. Then we convert $\mathcal{U}$ into a k-partition as follows. Sort the members of $\mathcal{U}$ so that $|U_1| \geq |U_2| \geq \cdots \geq |U_{k'}|$. We merge each $U_i \in \{U_{k+1}, \cdots, U_{k'}\}$ with a co-cluster chosen from $\{U_1, \cdots, U_k\}$ uniformly at random. This heuristic balances the size of co-clusters in the k-partition output by the algorithm.

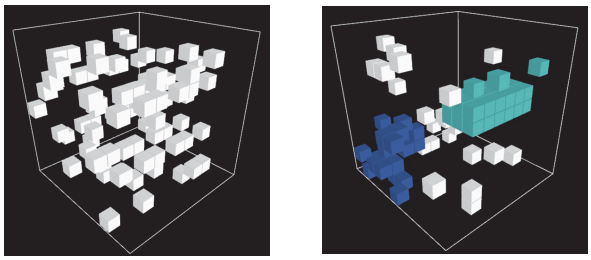
6 Experiments

In this section, we evaluate the performance of the proposal algorithms through experiments.

We conducted experiments on an Ubuntu server with an Intel Xeon E5-2690, 2.9GHz processor and 512GB memory, and implemented our algorithm in Java 1.7.0_79.

We verify the clustering qualities of the proposal algorithms through synthetic data, which is created as follows. This data set consists of tensors A, which has n dimensions in each order, and includes k ground truth co-clusters for some integers k, n, and m. A tensor A is created as follows. For all pairs of $i \in [m]$ and $j \in [n]$, we first choose an integer randomly from $[k]$, and denote it by $c_{i,j}$. This indicates that the j-th dimension j of the i-th order belongs to the $c_{i,j}$-th co-cluster in the ground truth co-clustering. For each $(j_1, \ldots, j_m) \in [n]^m$, we set $A(j_1, \ldots, j_m)$ to 1 with probability 0.5 if all of $\{j_1, \ldots, j_m\}$ belong to the same ground truth co-cluster, and change the probability for the others so that the number of $\delta(\mathcal{U})$ is accounted for 5% of all non-zero elements.

As for the threshold θ in Algorithm 2, we compute the minimum weight W^* of k-way cuts computed by iterating $\mathrm{KS}(G, w, k)$ 1000 times, and define θ as αW^* for some parameter $\alpha \geq 1$. We denote the proposed algorithm without heuristics by *Prop*. If it is combined with the distorted sampling heuristic and the balancing merge heuristic, we represent by adding "+D" and "+B," respectively.



(a) Input (b) Output

Figure 2: An order-3 tensor and its clustering result

Figure 2 represents an example of input and its clustering result computed by our algorithm. In Figure 2(a), elements colored by white represent those associated with positive values. In Figure 2(b), we rearrange dimensions so that dimensions in each cluster appear consecutively. Blue- and green-colored elements are those whose dimensions belong to the same clusters.

We evaluate the clustering quality by the normalized mutual information (NMI for short) (Lancichinetti, Fortunato, and Kertész 2009). NMI is defined as follows. Let $U^{\mathrm{t}} = \{U^{\mathrm{t}}_i \colon i \in [k]\}$ be a ground truth co-clustering, and let $U^{\mathrm{a}} = \{U^{\mathrm{a}}_j \colon j \in [k']\}$ be a co-clustering obtained by an algorithm. We denote the mutual information and the entropy by $I(U^{\mathrm{a}}; U^{\mathrm{t}}) = \sum_{i=1}^{k} \sum_{j=1}^{k'} \frac{|U^{\mathrm{a}}_i \cup U^{\mathrm{t}}_j|}{|V|} \log \frac{|V||U^{\mathrm{a}}_i \cup U^{\mathrm{t}}_j|}{|U^{\mathrm{a}}_i||U^{\mathrm{t}}_j|}$ and $H(U^*) = -\sum_{i=1}^{k} \frac{|U^*_i|}{|V|} \log \frac{|U^*_i|}{|V|}$ for each $* \in \{\mathrm{a}, \mathrm{t}\}$, respectively. Then, NMI of co-clustering U^{t} and U^{a} is defined as $\mathrm{NMI}(U^{\mathrm{a}}, U^{\mathrm{t}}) = \frac{I(U^{\mathrm{a}}; U^{\mathrm{t}})}{(H(U^{\mathrm{a}}) + H(U^{\mathrm{t}}))/2}$.

We compare the proposed algorithms with existing algorithms proposed by Papalexakis et al. (2013), from which we can define three variants of the algorithm: (1) standard PARAFAC decomposition (PARAFAC), (2) PARAFAC decomposition with nonnegative latent factors (PARAFAC-N), and (3) PARAFAC decomposition with sparse latent factors (PARAFAC-S). We downloaded a Matlab implementation of the algorithms from https://www.cs.cmu.edu/~epapalex/.

m, n, k in Table 1 indicate the number of order, dimension, and ground truth co-clustering, respectively. Table 1 indicates NMI and the runtime in seconds of the proposed and the existing algorithms over the synthetic data. The number of iterations l of the proposed algorithms set to 1000 and $\alpha = 1.00$, and k is fixed to the number of co-clusters in the ground truth co-clustering. In the table, type of "e" represents that the dimensions in the same order are evenly distributed to k ground truth co-clusters, while the dimensions are unevenly distributed in the type of "u".

From the results described in Table 1, we can observe that the proposal algorithms with the distorted sampling heuristic achieves better clustering quality than the algorithms with other heuristics in the type of "e". The algorithm with the both heuristics is better in the last two instances of the type "u". The reason behind this is clear because the balancing merge heuristic balances the sizes of co-clusters.

Compared with the existing algorithms, the proposed algorithm is competitive for lower-order tensors, and outperforms for higher-order tensors, with respect to both the clustering quality and the runtime. This is clearly a benefit of our algorithm, in which the runtime of each iteration is polynomial on the size of the corresponding hypergraph while the runtime of the existing algorithms exponentially increases in the order of a tensor.

In addition, we assess the scalability of proposed algorithm using real-world data generated from Amazon review data. Table 2 shows profiles of real-world data and the runtime of the algorithms. The runtime of the proposed algorithm is obtained by setting $l = 1$ and $k = 3$. We exploited Amazon review data which list over 30 millions of reviews including product's ID, name, price, reviewers ID, score, time stamp, and review text. From this data set, we created the following order-3 tensor. The tensor is the word co-occurrence tensors $A \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ where $A(j_1, j_2, j_3)$ is the number of review texts that contains word j_1, j_2, j_3 simultaneously. Decomposing co-occurrence tensor is well studied in natural language processing (Pennington, Socher, and Manning 2014). We generated seven tensors in this class by using the first 10, 20, 50, 100, 200, 500, and 1000 review data in the data set.

To confirm the quality of co-clusterings produced by the proposed algorithm, we applied it to a tensor generated from data of sharing bike system in New York City available at https://www.citibikenyc.com/system-data. The data consists of over 28 millions of trip data, each of which has 11 attributes including trip duration, start time and date, stop time and date, start station name, end station name, and so on. From this data, we created a tensor $A \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ where $A(j_1, j_2, j_3)$ is the number of trips that starts at station j_1 and reaches at station j_2 at time of j_3. The time j_3 is combined with the day of week $t_1 \in \{$Mon., Tue.,Wed., Thu., Fri., Sat., Sun.$\}$, starting hour $t_2 = \{0, 1, \ldots, 23\}$, and the trip length in minutes $t_3 = \{$0-10, 10-20, $\ldots$, 50-60, $>$60$\}$. This results in a tensor with $n_1 = 527, n_2 = 551, n_3 = 1176$. As a preprocessing, we omitted elements (j_1, j_2, j_3) whose weight $A(j_1, j_2, j_3)$ is less than 50, that is we reduce the weight of the elements to 0. Then, the number of nonzero elements is 18228. The result described in Table 3 and Figure 3 is obtained by setting $l = 1000$ and $k = 25$. Table 3 shows the most frequent three trips of the top five clusters where the clusters are sorted by the number of included trips. Its last column represents the days and the hours when the trips start and the length of trips in minutes. In Figure 3, we plotted the number of trips over the starting time in those five clusters. We can observe that most of the trips in clusters 1 and 2 start in the evening, while those in clusters 3 and 4 start in the morning. In cluster 5, we can find no characterization from the starting time, but most of the trips use a station at Central Park.

7 Conclusion

In this paper, we proposed a new co-clustering algorithm of tensors. Our algorithm is based on a formulation of the co-clustering problem as a problem of computing a k-way cut of a hypergraph. We extended Karger and Stein's random sampling algorithm for the k-way cut problem from graphs

Table 1: Clustering quality and the runtime in seconds for synthetic tensors

Instance	Prop		Prop+D		Prop+B		Prop+D+B		PARAFAC		PARAFAC-N		PARAFAC-S	
type, m, n, k	NMI	time	NMI	time	NMI	time	NMI	time	NMI	time	NMI	time	NMI	time
e, 3, 100, 3	0.06	4.8	0.77	5.0	0.09	4.8	0.87	5.1	**1.00**	1.7	0.91	**1.5**	0.08	2.1
e, 3, 100, 4	0.63	2.7	0.98	3.0	0.82	2.7	0.85	3.0	**1.00**	**0.3**	**1.00**	0.8	0.11	2.0
e, 3, 100, 5	0.11	2.1	**0.98**	2.4	0.55	2.1	0.91	2.4	0.59	**1.1**	0.77	1.8	0.02	2.0
e, 4, 50, 3	**0.98**	10.6	0.95	10.6	0.97	10.7	0.90	10.9	0.90	13.2	0.54	10.3	0.56	**9.8**
e, 4, 50, 4	0.63	**4.1**	**0.95**	5.0	0.81	4.9	**0.95**	5.1	0.70	10.0	0.72	13.2	0.67	11.9
e, 4, 50, 5	0.05	**2.1**	0.04	2.6	0.70	2.4	**0.90**	2.7	0.55	14.2	0.74	5.5	0.53	13.7
e, 5, 50, 3	0.99	216	0.99	214	0.96	214	0.93	**211**	**1.00**	246	**1.00**	287	0.95	320
e, 5, 50, 4	0.06	**74**	**1.00**	78	0.96	79	0.97	76	0.71	282	0.40	333	0.97	278
e, 5, 50, 5	0.03	**27**	0.78	28	0.14	28	**0.79**	29	**0.79**	292	0.60	273	0.52	317
u, 3, 100, 3	0.05	7.0	0.97	7.6	0.77	7.0	0.93	8.0	**1.00**	**1.6**	0.87	2.4	0.02	2.2
u, 3, 100, 4	0.60	3.3	0.07	3.9	0.57	3.3	0.93	3.7	0.81	**0.3**	**0.97**	1.4	0.02	2.0
u, 3, 100, 5	0.04	2.6	**1.00**	3.0	0.17	2.6	0.89	3.0	0.78	**0.7**	0.75	1.4	0.03	2.1

Table 2: Details of amazon review data and the runtime in seconds. MLE means that the memory limitation is exceeded.

Instance profile		Prop+D+B	PARAFAC	PARAFAC-N	PARAFAC-S
$\|E\|$	n_1, n_2, n_3	time	time	time	time
9351985	778, 778, 778	11.5	1039	1116	>3600
10073914	996, 996, 996	12.9	2155	2046	>3600
17252294	1589, 1589, 1589	25.4	>3600	>3600	>3600
34009370	2888, 2888, 2888	71.3	>3600	>3600	>3600
66280817	4595, 4595, 4595	147.1	MLE	MLE	MLE
138246119	7882, 7882, 7882	371.0	MLE	MLE	MLE
269751185	11965, 11965, 11965	996.2	MLE	MLE	MLE

Table 3: Top 3 trips that are frequently used

No.	Start station	Goal station	Starting time & length
1	Willoughby St & Fleet St	Adelphi St & Myrtle Ave	Mon., 0-10, 18
	Broad St & Bridge St	Barclay St & Church St	Mon., 0-10, 17
	Willoughby St & Fleet St	Adelphi St & Myrtle Ave	Mon., 0-10, 17
2	Greenwich St & N Moore St	Vesey Pl & River Terrace	Wed., 0-10, 17
	Greenwich St & N Moore St	Vesey Pl & River Terrace	Tue., 0-10, 17
	Greenwich St & N Moore St	Vesey Pl & River Terrace	Wed., 0-10, 16
3	E 32 St & Park Ave	1 Ave & E 30 St	Wed., 0-10, 09
	E 32 St & Park Ave	1 Ave & E 30 St	Mon., 0-10, 09
	8 Ave & W 31 St	9 Ave & W 18 St	Wed., 0-10, 09
4	Adelphi St & Myrtle Ave	DeKalb Ave & Hudson Ave	Fri., 0-10, 08
	Barclay St & Church St	6 Ave & Canal St	Thu., 0-10, 09
	Adelphi St & Myrtle Ave	DeKalb Ave & Hudson Ave	Tue., 0-10, 08
5	Central Park S & 6 Ave	Central Park S & 6 Ave	Sun., >60, 12
	Central Park S & 6 Ave	Central Park S & 6 Ave	Sun., >60, 13
	Central Park S & 6 Ave	Central Park S & 6 Ave	Sun., >60, 15

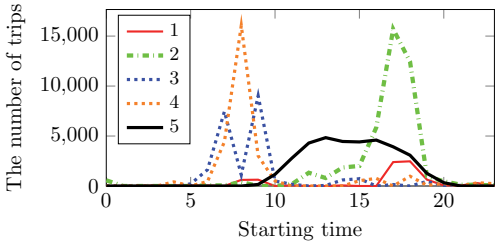


Figure 3: The number of trips when changing the starting time

to hypergraphs, and applied it to the co-clustering of tensors. In our algorithm, the runtime of each iteration is polynomial on the size of the hypergraph. Hence it performs well particularly for higher-order tensors.

References

Bisson, G., and Hussain, S. F. 2008. Chi-sim: A new similarity measure for the co-clustering task. In *Proceedings of the Seventh International Conference on Machine Learning and Applications, ICMLA 2008*, 211–217.

Chen, X.; Ritter, A.; Gupta, A.; and Mitchell, T. M. 2015. Sense discovery via co-clustering on images and text. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015*, 5298–5306.

Cheng, Y., and Church, G. M. 2000. Biclustering of expression data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, 93–103.

Dao, P.; Colak, R.; Salari, R.; Moser, F.; Davicioni, E.; Schönhuth, A.; and Ester, M. 2010. Inferring cancer subnetwork markers using density-constrained biclustering. *Bioinformatics* 26(18).

Dhillon, I. S.; Mallela, S.; and Modha, D. S. 2003. Information-theoretic co-clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 89–98.

Dhillon, I. S. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 269–274.

Fukunaga, T. 2013. Computing minimum multiway cuts in hypergraphs. *Discrete Optimization* 10(4):371–382.

Hartigan, J. A. 1972. Direct clustering of a data matrix. *Journal of the American Statistical Association* 67(337):123–129.

Hillar, C. J., and Lim, L. 2013. Most tensor problems are np-hard. *Journal of the ACM* 60(6):45.

Hussain, S. F.; Bisson, G.; and Grimal, C. 2010. An improved co-similarity measure for document clustering. In *Proceedings of the Ninth International Conference on Machine Learning and Applications, ICMLA 2010*, 190–197.

Hussain, S. F. 2011. Bi-clustering gene expression data using co-similarity. In *Advanced Data Mining and Applications - 7th International Conference, ADMA 2011, Proceedings, Part I*, 190–200.

Karger, D. R., and Stein, C. 1996. A new approach to the minimum cut problem. *Journal of the ACM* 43(4):601–640.

Karger, D. R. 1993. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, 21–30.

Karger, D. R. 2000. Minimum cuts in near-linear time. *Journal of the ACM* 47(1):46–76.

Lancichinetti, A.; Fortunato, S.; and Kertész, J. 2009. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics* 11(3):033015.

Madeira, S. C.; Teixeira, M. C.; Sá-Correia, I.; and Oliveira, A. L. 2010. Identification of regulatory modules in time series gene expression data using a linear time biclustering algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7(1):153–165.

O'Connor, L., and Feizi, S. 2014. Biclustering usinig message passing. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, 3617–3625.

Papalexakis, E. E.; Sidiropoulos, N. D.; and Bro, R. 2013. From K-means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *IEEE Transactions on Signal Processing* 61(2):493–506.

Peng, W., and Li, T. 2011. Temporal relation co-clustering on directional social network and author-topic evolution. *Knowledge Information Systems* 26(3):467–486.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, 1532–1543.

Shan, H., and Banerjee, A. 2008. Bayesian co-clustering. In *Proceedings of the 8th IEEE International Conference on Data Mining, ICDM 2008*, 530–539.

Thorup, M. 2008. Minimum k-way cuts via deterministic greedy tree packing. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, 159–166.

Wieling, M., and Nerbonne, J. 2011. Bipartite spectral graph partitioning for clustering dialect varieties and detecting their linguistic features. *Computer Speech & Language* 25(3):700–715.

Zha, H.; He, X.; Ding, C. H. Q.; Gu, M.; and Simon, H. D. 2001. Bipartite graph partitioning and data clustering. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management*, 25–32.

Zhao, L., and Zaki, M. J. 2005. Tricluster: An effective algorithm for mining coherent clusters in 3d microarray data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 694–705.

Zhou, Q.; Xu, G.; and Zong, Y. 2009. Web co-clustering of usage network using tensor decomposition. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Web Intelligence and International Conference on Intelligent Agent Technology — Workshops*, 311–314.

Zhu, Y.; Yang, H.; and He, J. 2015. Co-clustering based dual prediction for cargo pricing optimization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1583–1592.