General Tensor Spectral Co-clustering for Higher-Order Data

Tao Wu
Purdue University
wu577@purdue.edu

Austin R. Benson
Stanford University
arbenson@stanford.edu

David F. Gleich
Purdue University
dgleich@purdue.edu

Abstract

Spectral clustering and co-clustering are well-known techniques in data analysis, and recent work has extended spectral clustering to square, symmetric tensors and hypermatrices derived from a network. We develop a new tensor spectral co-clustering method that simultaneously clusters the rows, columns, and slices of a nonnegative three-mode tensor and generalizes to tensors with any number of modes. The algorithm is based on a new random walk model which we call the super-spacey random surfer. We show that our method out-performs state-of-the-art co-clustering methods on several synthetic datasets with ground truth clusters and then use the algorithm to analyze several real-world datasets.

1 Introduction

Clustering is a fundamental task in machine learning that aims to assign closely related entities to the same group. Traditional methods optimize some aggregate measure of the strength of pairwise relationships (e.g., similarities) between items. Spectral clustering is a particularly powerful technique for computing the clusters when the pairwise similarities are encoded into the adjacency matrix of a graph. However, many graph-like datasets are more naturally described by higher-order connections among several entities. For instance, multilayer or multiplex networks describe the interactions between several graphs simultaneously with node-node-layer relationships [21]. Nonnegative tensors are a common representation for many of these higher-order datasets. For instance the i, j, k entry in a third-order tensor might represent the similarity between items i and j in layer k.

Here we develop the General Tensor Spectral Co-clustering (GTSC) framework for clustering tensor data. The algorithm takes as input a nonnegative tensor, which may be sparse, non-square, and asymmetric, and outputs subsets of indices from each dimension (co-clusters). Underlying our method is a new stochastic process that models higher-order Markov chains, which we call a *super-spacey random walk*. This is used to generalize ideas from spectral clustering based on random walks. We introduce a variant on the well-known conductance measure from spectral graph partitioning [29] that we call *biased conductance* and describe how this provides a tensor partition quality metric; this is akin to Chung's use of circulations to spectrally-partition directed graphs [9]. Essentially, biased conductance is the exit probability from a set following our new super-spacey random walk model.

We use experiments on both synthetic and real-world problems to validate the effectiveness of our method[1]. For the synthetic experiments, we devise a "planted cluster" model for tensors and show that GTSC has superior performance compared to other state-of-the-art clustering methods in recovering the planted clusters. In real-world tensor data experiments, we find that our GTSC framework identifies stop-words and semantically independent sets in n-gram tensors as well as worldwide and regional airlines and airports in a flight multiplex network.

[1] Code and data for this paper are available at: https://github.com/wutao27/GtensorSC

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

1.1 Related work

The Tensor Spectral Clustering (TSC) algorithm [4], another generalization of spectral methods to higher-order graph data [4], is closely related. Both the perspective and high-level view are similar, but the details differ in important ways. For instance, TSC was designed for the case when the higher-order tensor recorded the occurrences of small subgraph patterns within the network. This imposes limitations, including how, because the tensor arose based on some underlying graph that the partitioning metric was designed explicitly for a graph. Thus, the applications are limited in scope and cannot model, for example, the airplane-airplane-airport multiplex network we analyze in Section 3.2. Second, for sparse data, the model used by TSC required a correction term with magnitude proportional to the sparsity in the tensor. In sparse tensors, this makes it difficult to accurately identify clusters, which we show in Section 3.1.

Most other approaches to tensor clustering proceed by using low-rank factorizations [18, 26]. or a k-means objective [20]. In contrast, our work is based on a stochastic interpretation (escape probabilities from a set), in the spirit of random walks in spectral clustering for graphs. There are also several methods specific to clustering multiplex networks [32, 25] and clustering graphs with multiple entities [13, 2]. Our method handles general tensor data, which includes these types of datasets as a special case. Hypergraphs clustering [17] can also model the higher-order structures of the data, and in the case of tensor data, it is approximated by a standard weighted graph.

1.2 Background on spectral clustering of graphs from the perspective of random walks

We first review graph clustering methods from the view of graph cuts and random walks, and then review the standard spectral clustering method using sweep cuts. In Section 2, we generalize these notions to higher-order data in order to develop our GTSC framework.

Let $\boldsymbol{A} \in \mathbb{R}_+^{n\times n}$ be the adjacency matrix of an undirected graph $G = (V, E)$ and let $n = |V|$ be the number of nodes in the graph. Define the diagonal matrix of degrees of vertices in V as $\boldsymbol{D} = \text{diag}(\boldsymbol{A}\mathbf{e})$, where $\mathbf{e}$ is the vector with all ones. The graph Laplacian is $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ and the transition matrix is $\boldsymbol{P} = \boldsymbol{A}\boldsymbol{D}^{-1} = \boldsymbol{A}^T\boldsymbol{D}^{-1}$. The transition matrix represents the transition probabilities of a random walk on the graph. If a walker is at node j, it transitions to node i with probability $P_{ij} = A_{ji}/D_{jj}$.

Conductance. One of the most widely-used quality metrics for partitioning a graph's vertices into two sets S and $\bar{S} = V\backslash S$ is conductance [29]. Intuitively, conductance measures the ratio of the number of edges in the graph that go between S and $\bar{S}$ to the number of edges in S or $\bar{S}$. Formally, we define conductance as:

$$\phi(S) = \text{cut}(S)/\min\big(\text{vol}(S), \text{vol}(\bar{S})\big), \tag{1}$$

where

$$\text{cut}(S) = \sum_{i\in S, j\in\bar{S}} A_{ij} \qquad \text{and} \qquad \text{vol}(S) = \sum_{i\in S, j\in V} A_{ij}. \tag{2}$$

A set S with small conductance is a good partition $(S, \bar{S})$. The following well-known observation relates conductance to random walks on the graph.

Observation 1 ([23]) *Let G be undirected, connected, and not bipartite. Start a random walk $(Z_t)_{t\in\mathbb{N}}$ where the initial state X_0 is randomly chosen following the stationary distribution of the random walk. Then for any set $S \in V$,*

$$\phi(S) = \max\big\{\Pr(Z_1 \in \bar{S} \mid Z_0 \in S), \Pr(Z_1 \in S \mid Z_0 \in \bar{S})\big\}.$$

This provides an alternative view of conductance—it measures the probability that one step of a random walk will traverse between S and $\bar{S}$. This random walk view, in concert with the super-spacey random walk, will serve as the basis for our biased conductance idea to partition tensors in Section 2.4.

Partitioning with a sweep cut. Finding the set of minimum conductance is an NP-hard combinatorial optimization problem [31]. However, there are real-valued relaxations of the problem that are tractable to solve and provide a guaranteed approximation [24, 11]. The most well known computes an eigenvector called the Fiedler vector and then uses a sweep cut to identify a partition based on this eigenvector.

The Fiedler eigenvector $\mathbf{z}$ solves $\boldsymbol{L}\mathbf{z} = \lambda \boldsymbol{D}\mathbf{z}$ where λ is the second smallest generalized eigenvalue. This can be equivalently formulated in terms of the random walk transition matrix $\boldsymbol{P}$. Specifically,

$$\boldsymbol{L}\mathbf{z} = \lambda \boldsymbol{D}\mathbf{z} \quad \Leftrightarrow \quad (\boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A})\mathbf{z} = \lambda \mathbf{z} \quad \Leftrightarrow \quad \mathbf{z}^T\boldsymbol{P} = (1-\lambda)\mathbf{z}^T.$$

The *sweep cut procedure* to identify a low-conductance set S from $\mathbf{z}$ is as follows:

1. Sort the vertices by $\mathbf{z}$ as $z_{\sigma_1} \leq z_{\sigma_2} \leq \cdots \leq z_{\sigma_n}$.
2. Consider the $n-1$ candidate sets $S_k = \{\sigma_1, \sigma_2, \cdots, \sigma_k\}$ for $1 \leq k \leq n-1$
3. Choose $S = \text{argmin}_{S_k} \phi(S_k)$ as the solution set.

The solution set S from this algorithm satisfies the celebrated Cheeger inequality [24, 10]: $\phi(S) \leq 2\sqrt{\phi_{opt}}$, where $\phi_{opt} = \min_{S \subset V} \phi(S)$ is the minimum conductance over any set of nodes. Computing $\phi(S_k)$ for all k only takes time linear in the number of edges in the graph because S_{k+1} and S_k differ only in the vertex σ_{k+1}.

To summarize, the spectral method requires two components: the second left eigenvector of $\boldsymbol{P}$ and the conductance criterion. We generalize these ideas to tensors in the following section.

2 A higher-order spectral method for tensor co-clustering

We now generalize the ideas from spectral graph partitioning to nonnegative tensor data. We first review our notation for tensors and then show how tensor data can be interpreted as a higher-order Markov chain. We briefly review Tensor Spectral Clustering [4] before introducing the new super-spacey random walk that we use here. This super-spacey random walk will allow us to compute a vector akin to the Fiedler vector for a tensor and to generalize conductance to tensors. Furthermore, we generalize the ideas from co-clustering in bipartite graph data [12] to rectangular tensors.

2.1 Preliminaries and tensor notation

We use $\underline{\boldsymbol{T}}$ to denote a tensor. As a generalization of a matrix, $\underline{\boldsymbol{T}}$ has m indices (making $\underline{\boldsymbol{T}}$ an mth-order or m-mode tensor), with the $(i_1, i_2, \cdot, i_m)$ entry denoted $\underline{T}_{i_1,i_2,\cdots,i_m}$. We will work with non-negative tensors where $\underline{T}_{i_1,i_2,\cdots,i_m} \geq 0$. We call a subset of the tensor entries with all but the first element fixed a *column* of the tensor. For instance, the j,k column of a three-mode tensor $\underline{\boldsymbol{T}}$ is $\underline{T}_{:,j,k}$. A tensor is *square* if the dimension of all the modes is equal and rectangular if not, and a square tensor is *symmetric* if it is equal for any permutation of the indices. For simplicity in the remainder of our exposition, we will focus on three-mode tensors. However, all of or ideas generalize to an arbitrary number of modes. (See, e.g., the work of Gleich et al. [15] and Benson et al. [5] for representative examples of how these generalizations work.) Finally, we use two operations between a tensor and a vector. First, a tensor-vector product with a three-mode tensor can output a vector, which we denote by:

$$\mathbf{y} = \underline{\boldsymbol{T}}\mathbf{x}^2 \quad \Leftrightarrow \quad y_i = \textstyle\sum_{j,k} \underline{T}_{i,j,k} x_j x_k.$$

Second, a tensor-vector product can also produce a matrix, which we denote by:

$$\boldsymbol{A} = \underline{\boldsymbol{T}}[\mathbf{x}] \quad \Leftrightarrow \quad A_{i,j} = \textstyle\sum_k \underline{T}_{i,j,k} x_k.$$

2.2 Forming higher-order Markov chains from nonnegative tensor data

Recall from Section 1.2 that we can form the transition matrix for a Markov chain from a square non-negative matrix $\boldsymbol{A}$ by normalizing the columns of the matrix $\boldsymbol{A}^T$. We can generalize this idea to define a higher-order Markov chain by normalizing a square tensor. This leads to a probability transition tensor $\underline{\boldsymbol{P}}$:

$$\underline{P}_{i,j,k} = \underline{T}_{i,j,k} / \textstyle\sum_i \underline{T}_{i,j,k} \tag{3}$$

where we assume $\sum_i \underline{T}_{i,j,k} > 0$. In Section 2.3, we will discuss the sparse case where the column $\underline{T}_{:,j,k}$ may be entirely zero. When that case does not arise, entries of $\underline{\boldsymbol{P}}$ can be interpreted as the transition probabilities of a second-order Markov chain $(Z_t)_{t \in \mathbb{N}}$:

$$\underline{P}_{i,j,k} = \Pr(Z_{t+1} = i \mid Z_t = j, Z_{t-1} = k).$$

In other words, If the last two states were j and k, then the next state is i with probability $\underline{P}_{i,j,k}$.

It is possible to turn any higher-order Markov chain into a first-order Markov chain on the product state space of all ordered pairs (i, j). The new Markov chain moves to the state-pair (i, j) from (j, k) with probability $\underline{P}_{i,j,k}$. Computing the Fiedler vector associated with this chain would be one approach to tensor clustering. However, there are two immediate problems. First, the eigenvector is of size n^2, which quickly becomes infeasible to store. Second, the eigenvector gives information about the product space—not the original state space. (In future work we plan to explore insights from marginals of this distribution.)

Recent work uses the *spacey random walk* and *spacey random surfer* stochastic processes to circumvent these issues [5]. The process is non-Markovian and generates a sequence of states X_t as follows. After arriving at state X_t, the walker promptly "spaces out" and forgets the state X_{t-1}, yet it still wants to transition according to the higher-order transitions $\underline{\boldsymbol{P}}$. Thus, it invents a state Y_t by drawing a random state from its history and then transitions to state X_{t+1} with probability $\underline{P}_{X_{t+1},X_t,Y_t}$. We denote $\text{Ind}\{\cdot\}$ as the indicator event and H_t as the history of the process up to time t,[2] then

$$\Pr(Y_t = j \mid H_t) = \tfrac{1}{t+n}\left(1 + \textstyle\sum_{r=1}^{t} \text{Ind}\{X_r = j\}\right). \tag{4}$$

In this case, we assume that the process has a non-zero probability of picking any state by inflating its history count by 1 visit. The spacey random surfer is a generalization where the walk follows the above process with probability α and teleports at random following a stochastic vector $\mathbf{v}$ with probability $1 - \alpha$. This is akin to how the PageRank random walk includes teleportation.

Limiting stationary distributions are solutions to the multilinear PageRank problem [15]:

$$\alpha \underline{\boldsymbol{P}}\mathbf{x}^2 + (1-\alpha)\mathbf{v} = \mathbf{x}, \tag{5}$$

and the limiting distribution $\mathbf{x}$ represents the stationary distribution of the transition matrix $\underline{\boldsymbol{P}}[\mathbf{x}]$ [5]. The transition matrix $\underline{\boldsymbol{P}}[\mathbf{x}]$ asymptotically approximates the spacey walk or spacey random surfer.

Thus, it is feasible to compute an eigenvector of $\underline{\boldsymbol{P}}[\mathbf{x}]$ matrix and use it with the sweep cut procedure on a generalized notion of conductance. However, this derivation assumes that all n^2 columns of $\underline{\boldsymbol{T}}$ were non-zero, which does not occur in real-world datasets. The TSC method adjusted the tensor $\underline{\boldsymbol{T}}$ and replaced any columns of all zeros with the uniform distribution vector [4]. Because the number of zero-columns may be large, this strategy dilutes the information in the eigenvector (see Appendix D.1). We deal with this issue more generally in the following section, and note that our new solution outperforms TSC in our experiments (Section 3).

2.3 A stochastic process for sparse tensors

Here we consider another model of the random surfer that avoids the issue of undefined transitions—which correspond to columns of $\underline{\boldsymbol{T}}$ that are all zero—entirely. If the surfer attempts to use an undefined transition, then the surfer moves to a random state drawn from history. Formally, define the set of feasible states by

$$\mathcal{F} = \{(j,k) \mid \sum_i \underline{T}_{i,j,k} > 0\}. \tag{6}$$

Here, the set $\mathcal{F}$ denotes all the columns in $\underline{\boldsymbol{T}}$ that are non-zero. The transition probabilities of our proposed stochastic process are given by

$$\begin{aligned} &\Pr(X_{t+1} = i \mid X_t = j, H_t) \\ &\quad = (1-\alpha)v_i + \alpha \textstyle\sum_k \Pr(X_{t+1} = i \mid X_t = j, Y_t = k, H_t)\Pr(Y_t = k \mid H_t) \end{aligned} \tag{7}$$

$$\Pr(X_{t+1} = i \mid X_t = j, Y_t = k, H_t) = \begin{cases} \underline{T}_{i,j,k} / \sum_i \underline{T}_{i,j,k} & (j,k) \in \mathcal{F} \\ \frac{1}{n+t}(1 + \sum_{r=1}^{t} \text{Ind}\{X_r = i\}) & (j,k) \notin \mathcal{F}, \end{cases} \tag{8}$$

where v_i is the teleportation probability. Again Y_t is chosen according to Equation (4). We call this process the *super-spacey random surfer* because when the transitions are not defined it picks a random state from history.

This process is a (generalized) vertex-reinforced random walk [3]. Let $\underline{\boldsymbol{P}}$ be the normalized tensor $\underline{P}_{i,j,k} = T_{i,j,k} / \sum_i T_{i,j,k}$ only for the columns in $\mathcal{F}$ and where all other entries are zero. Stationary distributions of the stochastic process must satisfy the following equation:

$$\alpha \underline{\boldsymbol{P}}\mathbf{x}^2 + \alpha(1 - \|\underline{\boldsymbol{P}}\mathbf{x}^2\|_1)\mathbf{x} + (1-\alpha)\mathbf{v} = \mathbf{x}, \tag{9}$$

[2] Formally, this is the σ-algebra generated by the states $X_1, \ldots, X_t$.

where $\mathbf{x}$ is a probability distribution vector (see Appendix A.1 for a proof). At least one solution vector $\mathbf{x}$ must exist, which follows directly from Brouwer's fixed-point theorem. Here we give a sufficient condition for it to be unique and easily computable.

Theorem 2.1 *If $\alpha < 1/(2m-1)$ then there is a unique solution $\mathbf{x}$ to (9) for the general m-mode tensor. Furthermore, the iterative fixed point algorithm*

$$\mathbf{x}_{k+1} = \alpha \underline{\boldsymbol{P}} \mathbf{x}_k^2 + \alpha(1 - \|\underline{\boldsymbol{P}} \mathbf{x}_k^2\|_1)\mathbf{x}_k + (1-\alpha)\mathbf{v}_k \tag{10}$$

will converge at least linearly to this solution.

This is a nonlinear setting and tighter convergence results are currently unknown, but these are unlikely to be tight on real-world data. For our experiments, we found that high values (e.g., 0.95) of α do not impede convergence. We use $\alpha = 0.8$ for all our experiments.

In the following section, we show how to form a Markov chain from $\mathbf{x}$ and then develop our spectral clustering technique by operating on the corresponding transition matrix.

2.4 First-order Markov approximations and biased conductance for tensor partitions

From Observation 1 in Section 1.2, we know that conductance may be interpreted as the exit probability between two sets that form a partition of the nodes in the graph. In this section, we derive an equivalent first-order Markov chain from the stationary distribution of the *super-spacey random surfer*. If this Markov chain was guaranteed to be reversible, then we could apply the standard definitions of conductance and the Fiedler vector. This will not generally be the case, and so we introduce a biased conductance measure to partition this non-reversible Markov chain with respect to starting in the stationary distribution of the super-spacey random walk. We use the second largest, real-valued eigenvector of the Markov chain as an approximate Fiedler vector. Thus, we can use the sweep cut procedure described in Section 1.2 to identify the partition.

Forming a first-order Markov chain approximation. In the following derivation, we use the property of the two tensor-vector products that $\underline{\boldsymbol{P}}[\mathbf{x}]\mathbf{x} = \underline{\boldsymbol{P}}\mathbf{x}^2$. The stationary distribution $\mathbf{x}$ for the super-spacey random surfer is equivalently the stationary distribution of the Markov chain with transition matrix

$$\alpha\big(\underline{\boldsymbol{P}}[\mathbf{x}] + \mathbf{x}(\mathbf{e}^T - \mathbf{e}^T\underline{\boldsymbol{P}}[\mathbf{x}])\big) + (1-\alpha)\mathbf{v}\mathbf{e}^T.$$

(Here we have used the fact that $\mathbf{x} \geq 0$ and $\mathbf{e}^T\mathbf{x} = 1$.) The above transition matrix denotes transitioning based on a first-order Markov chain with probability α, and based on a fixed vector $\mathbf{v}$ with probability $1-\alpha$. We introduce this following first-order Markov chain

$$\tilde{\boldsymbol{P}} = \underline{\boldsymbol{P}}[\mathbf{x}] + \mathbf{x}(\mathbf{e}^T - \mathbf{e}^T\underline{\boldsymbol{P}}[\mathbf{x}]),$$

which represents a useful (but crude) approximation of the higher-order structure in the data. First, we determine how often we visit states using the *super-spacey random surfer* to get a vector $\mathbf{x}$. Then the Markov chain $\tilde{\boldsymbol{P}}$ will tend to have a large probability of spending time in states where the higher-order information concentrates. This matrix represents a first-order Markov chain on which we can compute an eigenvector and run a sweep cut.

Biased conductance. Consider a random walk $(Z_t)_{t\in\mathbb{N}}$. We define the *biased conductance* $\phi_{\mathbf{p}}(S)$ of a set $S \subset \{1,\ldots,n\}$ to be

$$\phi_{\mathbf{p}}(S) = \max\big\{\Pr(Z_1 \in \bar{S} \mid Z_0 \in S),\ \Pr(Z_1 \in S \mid Z_0 \in \bar{S})\big\},$$

where Z_0 is chosen according to a fixed distribution $\mathbf{p}$. Just as with the standard definition of conductance, we can interpret biased conductance as an escape probability. However, the initial state Z_0 is not chosen following the stationary distribution (as in the standard definition with a reversible chain) but following $\mathbf{p}$ instead. This is why we call it *biased conductance*. We apply this measure to $\tilde{\boldsymbol{P}}$ using $\mathbf{p} = \mathbf{x}$ (the stationary distribution of the super-spacey walk). This choice emphasizes the higher-order information. Our idea of biased conductance is equivalent to how Chung defines a conductance score for a directed graph [9].

We use the eigenvector of $\tilde{\boldsymbol{P}}$ with the second-largest real eigenvalue as an analogue of the Fiedler vector. If the chain were reversible, this would be exactly the Fiedler vector. When it is not, then the vector coordinates still encode indications of state clustering [30]; hence, this vector serves as a principled heuristic. It is important to note that although $\tilde{\boldsymbol{P}}$ is a dense matrix, we can implement the two operations we need with $\tilde{\boldsymbol{P}}$ in time and space that depends only on the number of non-zeros of the sparse tensor $\underline{\boldsymbol{P}}$ using standard iterative methods for eigenvalues of matrices (see Appendix B.1).

2.5 Handling rectangular tensor data

So far, we have only considered square, symmetric tensor data. However, tensor data are often rectangular. This is usually the case when the different modes represent different types of data. For example, in Section 3.2, we examine a tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{p \times n \times n}$ of airline flight data, where $\underline{T}_{i,j,k}$ represents that there is a flight from airport j to airport k on airline i. Our approach is to embed the rectangular tensor into a larger square tensor and then symmetrize this tensor, using approaches developed by Ragnarsson and Van Loan [28]. After the embedding, we can run our algorithm to simultaneously cluster rows, columns, and slices of the tensor. This approach is similar in style to the symmetrization of bipartite graphs for co-clustering proposed by Dhillon [12].

Let $\underline{\boldsymbol{U}}$ be an n-by-m-by-ℓ rectangular tensor. Then we embed $\underline{\boldsymbol{U}}$ into a square three-mode tensor $\underline{\boldsymbol{T}}$ with $n + m + \ell$ dimensions and where $\underline{U}_{i,j,k} = \underline{T}_{i,j+n,k+n+m}$. This is illustrated in Figure 1 (left). Then we symmetrize the tensor by using all permutations of the indices Figure 1 (right). When viewed as a 3-by-3-by-3 block tensor, the tensor is

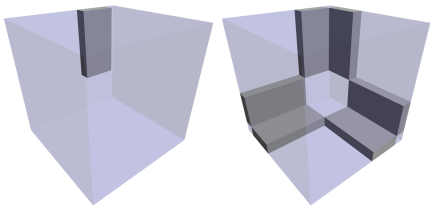

Figure 1: The tensor is first embedded into a larger square tensor (left) and then this square tensor is symmetrized (right).

$$\underline{\boldsymbol{T}} = \left[\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & \underline{\boldsymbol{U}}_{(1,3,2)} & 0 & \underline{\boldsymbol{U}}_{(1,2,3)} & 0 \\ 0 & 0 & \underline{\boldsymbol{U}}_{(2,3,1)} & 0 & 0 & 0 & \underline{\boldsymbol{U}}_{(2,1,3)} & 0 & 0 \\ 0 & \underline{\boldsymbol{U}}_{(3,2,1)} & 0 & \underline{\boldsymbol{U}}_{(3,1,2)} & 0 & 0 & 0 & 0 & 0 \end{array}\right],$$

where $\underline{\boldsymbol{U}}_{(1,3,2)}$ is a generalized transpose of $\underline{\boldsymbol{U}}$ with the dimensions permuted.

2.6 Summary of the algorithm

Our GTSC algorithm works by recursively applying the sweep cut procedure, similar to the recursive bisection procedures for clustering matrix-based data [7]. Formally for each cut, we:

1. Compute the super-spacey stationary vector $\mathbf{x}$ (Equation (9)) and form $\underline{\boldsymbol{P}}[\mathbf{x}]$.
2. Compute second largest left, real-valued eigenvector $\mathbf{z}$ of $\tilde{\boldsymbol{P}} = \underline{\boldsymbol{P}}[\mathbf{x}] + \mathbf{x}(\mathbf{e}^T - \mathbf{e}^T \underline{\boldsymbol{P}}[\mathbf{x}])$.
3. Sort the vertices by the eigenvector $\mathbf{z}$ as $z_{\sigma_1} \leq z_{\sigma_2} \leq \cdots \leq z_{\sigma_n}$.
4. Find the set $S_k = \{\sigma_1, \ldots, \sigma_k\}$ for which the biased conductance $\phi_{\mathbf{x}}(S_k)$ on transition matrix $\tilde{\boldsymbol{P}}$ is minimized.

We continue partitioning as long as the clusters are large enough or we can get good enough splits. Specifically, if a cluster has dimension less than a specified size minimum size, we do not consider it for splitting. Otherwise, the algorithm recursively splits the cluster if either (1) its dimension is above some threshold or (2) the biased conductance of a new split is less than a target value ϕ^*[3]. The overall algorithm is summarized in Appendix B as well as the algorithm complexity. Essentially, the algorithm scales linearly in the number of non-zeros of the tensor for each cluster that is produced.

3 Experiments

We now demonstrate the efficacy of our method by clustering synthetic and real-world data. We find that our method is better at recovering planted cluster structure in synthetically generated tensor data compared to other state-of-the-art methods. Please refer to Appendix C for the parameter details.

3.1 Synthetic data

We generate tensors with planted clusters and try to recover the clusters. For each dataset, we generate 20 groups of nodes that will serve as our planted clusters, where the number of nodes in each group from a truncated normal distribution with mean 20 and variance 5 so that each group has at least 4 nodes. For each group g we also assign a weight w_g where the weight depends on the group number. For group i, the weight is $(\sigma\sqrt{2\pi})^{-1} \exp(-(i - 10.5)^2/(2\sigma^2))$, where σ varies by experiment. Non-zeros correspond to interactions between three indices (triples). We generate t_w triples whose indices are within a group and t_a triples whose indices span across more than one group.

[3] We tested ϕ^* from 0.3 to 0.4, and we found the value of ϕ^* is not very sensitive to the experimental results.

Table 1: Adjusted Rand Index (ARI), Normalized Mutual Information (NMI), and F1 scores on various clustering methods for recovering synthetically generated tensor data with planted cluster structure. The $\pm$ entries are the standard deviation over 5 trials.

	ARI	NMI	F1	ARI	NMI	F1
	Square tensor with $\sigma = 4$			Rectangular tensor with $\sigma = 4$		
GTSC	$\mathbf{0.99}\pm0.01$	$\mathbf{0.99}\pm0.00$	$\mathbf{0.99}\pm0.01$	$\mathbf{0.97}\pm0.06$	$\mathbf{0.98}\pm0.03$	$\mathbf{0.97}\pm0.05$
TSC	0.42 ± 0.05	0.60 ± 0.04	0.45 ± 0.04	0.38 ± 0.17	0.53 ± 0.15	0.41 ± 0.16
PARAFAC	0.82 ± 0.05	0.94 ± 0.02	0.83 ± 0.04	0.81 ± 0.04	0.90 ± 0.02	0.82 ± 0.04
SC	$\mathbf{0.99}\pm0.01$	$\mathbf{0.99}\pm0.01$	$\mathbf{0.99}\pm0.01$	0.91 ± 0.06	0.94 ± 0.04	0.91 ± 0.06
MulDec	0.48 ± 0.05	0.66 ± 0.03	0.51 ± 0.05	0.27 ± 0.06	0.39 ± 0.05	0.32 ± 0.05
	Square tensor with $\sigma = 2$			Rectangular tensor with $\sigma = 2$		
GTSC	$\mathbf{0.78}\pm0.13$	$\mathbf{0.89}\pm0.06$	$\mathbf{0.79}\pm0.12$	$\mathbf{0.96}\pm0.06$	$\mathbf{0.97}\pm0.04$	$\mathbf{0.96}\pm0.06$
TSC	0.41 ± 0.11	0.60 ± 0.09	0.44 ± 0.10	0.28 ± 0.08	0.44 ± 0.10	0.32 ± 0.08
PARAFAC	0.48 ± 0.08	0.67 ± 0.04	0.50 ± 0.07	0.10 ± 0.04	0.24 ± 0.05	0.15 ± 0.04
SC	0.43 ± 0.07	0.66 ± 0.04	0.47 ± 0.06	0.38 ± 0.07	0.52 ± 0.05	0.41 ± 0.07
MulDec	0.19 ± 0.01	0.37 ± 0.01	0.24 ± 0.01	0.08 ± 0.01	0.19 ± 0.02	0.14 ± 0.01

The t_w triples are chosen by first uniformly selecting a group g and then uniformly selecting three indices i, j, and k from group g and finally assigning a weight of w_g. For the t_a triples, the sampling procedure first selects an index i from group g_i with a probability proportional to the weights of the group. In other words, indices in group g are chosen proportional to w_g. Two indices j and k are then selected uniformly at random from groups g_j and g_k other than g_i. Finally, the weight in the tensor is assigned to be the average of the three group weights. For rectangular data, we follow a similar procedure where we distinguish between the indices for each mode of the tensor.

For our experiments, $t_w = 10,000$ and $t_a = 1,000$, and the variance σ that controls the group weights is 2 or 4. For each value of σ, we create 5 sample datasets. The value of σ affects the concentration of the weights and how certain groups of nodes interact with others. This skew reflects properties of the real-world networks we examine in the next section.

Our GTSC method is compared with Tensor Spectral Clustering (TSC) [4], the Tensor Decomposition PARAFAC [1], Spectral Clustering (SC) via Multilinear SVD [14] and Multilinear Decomposition (MulDec) [26]. Table 1 depicts the performances of the four algorithms in recovering the planted clusters. In all cases, GTSC has the best performance. We note that the running time is a few seconds for GTSC, TSC and SC and nearly 30 minutes for PARAFAC and MulDec per trial. Note that the tensors have roughly $50,000$ non-zeros. The poor scalability prohibits the later two methods from being applied to the real-world tensors in the following section.

3.2 Case study in airline flight networks

We now turn to studying real-world tensor datasets. We first cluster an airline-airport multimodal network which consists of global air flight routes from 539 airlines and $2,939$ airports[4]. In this application, the entry $\underline{T}_{i,j,k}$ of the three-mode tensor $\underline{\boldsymbol{T}}$ is 1 if airline i flies between airports j and k and 0 otherwise. Figure 2 illustrates the connectivity of the tensor with a random ordering of the indices (left) and the ordering given by the popularity of co-clusters (right). We can see that after the co-clustering, there is clear structure in the data tensor.

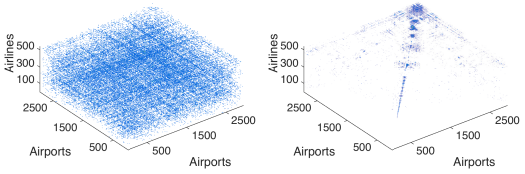

Figure 2: Visualization of the airline-airport data tensor. The x and y axes index airports and the z axis indexes airlines. A dot represents that an airline flies between those two airports. On the left, indices are sorted randomly. On the right, indices are sorted by the co-clusters found by our GTSC framework, which reveals structure in the tensor.

One prominent cluster found by the method corresponds to large international airports in cities such as Beijing and New York City. This group only accounts for 8.5% of the total number of airports, but it is responsible for 59% of the total routes. Figure 2 illustrates this result—the airports with the highest

[4]Data were collected from http://openflights.org/data.html#route.

indices are connected to almost every other airport. This cluster is analogous to the "stop word" group we will see in the n-gram experiments. Most other clusters are organized geographically. Our GTSC framework finds large clusters for Europe, the United States, China/Taiwan, Oceania/SouthEast Asia, and Mexico/Americas. Interestingly, Cancún International Airport is included with the United States cluster, likely due to large amounts of tourism.

3.3 Case study on n-grams

Next, we study data from n-grams (consective sequences of words in texts). We construct a square mode-n tensor where indices correspond to words. An entry in the tensor is the number of occurrences this n-gram. We form tensors from both English and Chinese corpora for $n = 3, 4$.[5] The non-zeros in the tensor consist of the frequencies of the one million most frequent n-grams.

English n-grams. We find several conclusions that hold for both tensor datasets. Two large groups in both datasets consist of stop words, i.e., frequently occuring connector words. In fact, 48% (3-gram) and 64% (4-gram) of words in one cluster are prepositions (e.g., *in, of, as, to*) and link verbs (e.g., *is, get, does*). In the another cluster, 64% (3-gram) and 57% (4-gram) of the words are pronouns (e.g., *we, you, them*) and link verbs. This result matches the structure of English language where link verbs can connect both prepositions and pronouns whereas prepositions and pronouns are unlikely to appear in close vicinity. Other groups consist of mostly semantically related English words, e.g.,

{*cheese, cream, sour, low-fat, frosting, nonfat, fat-free*} and
{*bag, plastic, garbage, grocery, trash, freezer*}.

The clustering of the 4-gram tensor contains some groups that the 3-gram tensor fails to find, e.g.,

{*german, chancellor, angela, merkel, gerhard, schroeder, helmut, kohl*}.

In this case, Angela Merkel, Gerhard Schroeder, and Helmut Kohl have all been German chancellors, but it requires a 4-gram to make this connection strong. Likewise, some clusters only appear from clustering the 3-gram tensor. One such cluster is

{*church, bishop, catholic, priest, greek, orthodox, methodist, roman, episcopal*}.

In 3-grams, we may see phrases such as "catholic church bishop", but 4-grams containing these words likely also contain stop words, *e.g.*, "bishop of the church". However, since stop words already form their own cluster, this connection is destoryed.

Chinese n-grams. We find that many of the conclusions from the English n-gram datasets also hold for the Chinese n-gram datasets. This includes groups of stop words and semantically related words. For example, there are two clusters consisting of mostly stop words (200 most frequently occurring words) from the 3-gram and 4-gram tensors. In the 4-gram data, one cluster of 31 words consists entirely of stop words and another cluster contains 36 total words, of which 23 are stop words.

There are some words from the two groups that are not typically considered as stop words, e.g.,

社会 *society,* 经济 *economy,* 发展 *develop,* 主义 *-ism,* 国家 *nation,* 政府 *government*

These words are also among the top 200 most common words according to the corpus. This is a consequence of the dataset coming from scanned Chinese-language books and is a known issue with the Google Books corpus [27]. In this case, it is a feature as we are illustrating the efficacy of our tensor clustering framework rather than making any linguistic claims.

4 CONCLUSION

In this paper we developed the General Tensor Spectral Co-clustering (GTSC) method for co-clustering the modes of nonnegative tensor data. Our method models higher-order data with a new stochastic process, the super-spacey random walk, which is a variant of a higher-order Markov chain. With the stationary distribution of this process, we can form a first-order Markov chain which captures properties of the higher-order data and then use tools from spectral graph partitioning to find co-clusters. In future work, we plan to create tensors that bridge information from multiple modes. For instance, clusters in the n-gram data depended on n, e.g., the names of various German chancellors only appeared as a 4-gram cluster. It would be useful to have a holistic tensor to jointly partition both 3- and 4-gram information.

Acknowledgements. TW and DFG are supported by NSF IIS-1422918 and DARPA SIMPLEX. ARB is supported by a Stanford Graduate Fellowship.

[5] English n-gram data were collected from http://www.ngrams.info/intro.asp and Chinese n-gram data were collected from https://books.google.com/ngrams.

A Super-spacey random surfer

A.1 Stationary distribution

Suppose the process has run for a very long time and that $\mathbf{x}_t$ is the current empirical distribution. From equation (7), we have

$$\Pr(X_{t+1} = i | X_t = j) = (1-\alpha)\mathbf{v}_i + \alpha\Big(\sum_{(j,k)\in\mathcal{F}} \underline{P}_{i,j,k}\mathbf{x}_t(k) + \sum_{(j,k)\notin\mathcal{F}} \mathbf{x}_t(k)\mathbf{x}_t(i)\Big). \tag{11}$$

The above transition process can be treated as a Markov chain with transition matrix depending on the current empirical distribution $\mathbf{x}_t$. Let $\boldsymbol{M}_t$ denote this Markov chain transition matrix, and we have:

$$\boldsymbol{M}_t = (1-\alpha)\mathbf{v}\mathbf{e}^T + \alpha\underline{\boldsymbol{P}}[\mathbf{x}_t] + \alpha\mathbf{x}_t(\mathbf{e}^T - \mathbf{e}^T\underline{\boldsymbol{P}}[\mathbf{x}_t]).$$

To understand the above expression, the first term of $\boldsymbol{M}_t$ comes from the first term of equation (11) and so does the second term. For the third term, the j-th entry of $\mathbf{e}^T - \mathbf{e}^T\underline{\boldsymbol{P}}[\mathbf{x}_t]$ is exactly $\sum_{(j,k)\notin\mathcal{F}} \mathbf{x}_t(k)$, so that $\boldsymbol{M}_t$ is a column stochastic matrix. At stationarity of the super-spacey random walk, the stationary distribution of this Markov chain must satisfy $\mathbf{x} = \boldsymbol{M}_t\mathbf{x}$. When the random walk is at its stationary distribution $\mathbf{x}$, $\boldsymbol{M}_t$ will not change but be fixed as:

$$\boldsymbol{M}_t = \boldsymbol{M} = (1-\alpha)\mathbf{v}\mathbf{e}^T + \alpha\underline{\boldsymbol{P}}[\mathbf{x}] + \alpha\mathbf{x}(\mathbf{e}^T - \mathbf{e}^T\underline{\boldsymbol{P}}[\mathbf{x}]).$$

And we have:

$$\begin{aligned}
\boldsymbol{M}\mathbf{x} &= (1-\alpha)\mathbf{v}\mathbf{e}^T\mathbf{x} + \alpha\underline{\boldsymbol{P}}[\mathbf{x}]\mathbf{x} + \alpha\mathbf{x}(\mathbf{e}^T - \mathbf{e}^T\underline{\boldsymbol{P}}[\mathbf{x}])\mathbf{x} \\
&= (1-\alpha)\mathbf{v} + \alpha\underline{\boldsymbol{P}}\mathbf{x}^2 + \alpha\mathbf{x}(1 - \mathbf{e}^T\underline{\boldsymbol{P}}\mathbf{x}^2) \\
&= (1-\alpha)\mathbf{v} + \alpha\underline{\boldsymbol{P}}\mathbf{x}^2 + \alpha(1 - \|\underline{\boldsymbol{P}}\mathbf{x}^2\|_1)\mathbf{x}.
\end{aligned}$$

Thus equation 9 gives us the necessary condition for $\mathbf{x}$ being the stationary distribution. A more formal proof is to use the results from [3] to show that (9) is the necessary condition of the stationary distribution in a vertex reinforced random walk.

A.2 Proof of Theorem 2.1

Let $\boldsymbol{R}$ denote the mode-1 unfolding of $\underline{\boldsymbol{P}}$:

$$\boldsymbol{R} = [\underline{\boldsymbol{P}}(:,:,1) \mid \underline{\boldsymbol{P}}(:,:,2) \mid \cdots \mid \underline{\boldsymbol{P}}(:,:,n)].$$

Note that $\boldsymbol{R}(\mathbf{x}\otimes\mathbf{x}) = \underline{\boldsymbol{P}}\mathbf{x}^2$ where $\otimes$ is the Kronecker product. Assume $\mathbf{x}$ and $\mathbf{y}$ are two solutions of (9). Let $r_x = \|\boldsymbol{R}(\mathbf{x}\otimes\mathbf{x})\|_1$ and $r_y = \|\boldsymbol{R}(\mathbf{y}\otimes\mathbf{y})\|_1$. Then

$$\|\mathbf{x}-\mathbf{y}\|_1 \le \alpha\|\boldsymbol{R}(\mathbf{x}\otimes\mathbf{x} - \mathbf{y}\otimes\mathbf{y})\|_1 + \alpha\|(1-r_x)\mathbf{x} - (1-r_y)\mathbf{y}\|_1.$$

By Lemma 4.5 of [15], the first term

$$\alpha\|\boldsymbol{R}(\mathbf{x}\otimes\mathbf{x} - \mathbf{y}\otimes\mathbf{y})\|_1 \le \alpha\|\boldsymbol{R}\|_1\|\mathbf{x}\otimes\mathbf{x} - \mathbf{y}\otimes\mathbf{y}\|_1 \le 2\alpha\|\mathbf{x}-\mathbf{y}\|_1.$$

The second term satifies

$$\begin{aligned}
&\alpha\|(1-r_x)\mathbf{x} - (1-r_y)\mathbf{y}\|_1 \\
&= \alpha\|(1-r_x)(\mathbf{x}-\mathbf{y}) - (r_x - r_y)\mathbf{y}\|_1 \\
&\le \alpha\|(1-r_x)(\mathbf{x}-\mathbf{y})\|_1 + \alpha|r_y - r_x| \\
&\le \alpha\|\mathbf{x}-\mathbf{y}\|_1 + \alpha\|\boldsymbol{R}(\mathbf{x}\otimes\mathbf{x} - \mathbf{y}\otimes\mathbf{y})\|_1 \le 3\alpha\|\mathbf{x}-\mathbf{y}\|_1.
\end{aligned}$$

Combining the above two facts, we know when $\alpha < 1/5$ the solution is unique. For an m-mode tensor, this idea generalizes to $\alpha < 1/(2m-1)$.

For the convergence of the fixed point algorithm (10), the same analysis shows that $\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_1 \le 5\alpha\|\mathbf{x}_k - \mathbf{x}^*\|_1$, and so the iteration converges at least linearly when the solution is unique.

B Algorithm discussion

The overall algorithm is summarized in Algorithm 1.

We also have a couple of pre-processing steps. First, we have to symmetrize the data if the tensor is rectangular. Second, we look for "empty" indices that do not participate in the tensor structure. Formally, index i is empty if $\underline{\boldsymbol{T}}_{ijk} = 0$ for all j and k.

Algorithm 1 General Tensor Spectral Co-clustering

Require:
Symmetric square tensor $\underline{\boldsymbol{T}} \in \mathbb{R}_{+}^{n\times n\times n}$, $\alpha \in (0,1)$
Stopping criterion `max-size`, `min-size`, ϕ^*

Ensure:
Partitioning C of indices $\{1,\ldots,n\}$.

$C = \{\{1,\ldots,n\}\}$
IF $n \leq$ `min-size`: **RETURN**
Generate transition tensor $\underline{\boldsymbol{P}}$ by

$$\underline{P}_{ijk} = \begin{cases} \underline{T}_{ijk} / \sum_{i=1}^{n} \underline{T}_{ijk} & \text{if } \sum_{i=1}^{n} \underline{T}_{ijk} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Compute super-spacey stationary vector $\mathbf{x}$ (Equation (9)) and form $\underline{\boldsymbol{P}}[\mathbf{x}]$.
Compute second largest left, real-valued eigenvector $\mathbf{z}$ of
$\tilde{\boldsymbol{P}} = \underline{\boldsymbol{P}}[\mathbf{x}] + \mathbf{x}(\mathbf{e}^T - \mathbf{e}^T\underline{\boldsymbol{P}}[\mathbf{x}])$ (that is, $\mathbf{z}^T\tilde{\boldsymbol{P}} = \lambda\mathbf{z}^T$).
$\sigma \leftarrow$ Sort eigenvector $\mathbf{z}$
$(S,\phi) \leftarrow$ Biased Conductance Sweep Cut$(\sigma, \underline{\boldsymbol{P}}[\mathbf{x}])$ with bias $\mathbf{p} = \mathbf{x}$.
if $n \geq$ `max-size` or $\phi \leq \phi^*$ **then**
  $C_S =$ Algorithm 1 on sub-tensor $\underline{\boldsymbol{T}}_{S,S,S}$.
  $C_{\bar{S}} =$ Algorithm 1 on sub-tensor $\underline{\boldsymbol{T}}_{\bar{S},\bar{S},\bar{S}}$.
  $C = C_S \cup C_{\bar{S}}$.
end if
RETURN C

B.1 Linear time sweep cut

In this section we prove that the operations we need for computing the eigenvector and conducting the sweep cut for $\tilde{\boldsymbol{P}}$ (possibly a dense matrix) depend only on the number of non-zeros of the sparse tensor $\underline{\boldsymbol{P}}$.

For computing the eigenvector of $\tilde{\boldsymbol{P}}$, it only involves the mat-vec operation (i.e., $\tilde{\boldsymbol{P}}\mathbf{b}$):

$$\tilde{\boldsymbol{P}}\mathbf{b} = \underline{\boldsymbol{P}}[\mathbf{x}]\mathbf{b} + \mathbf{x}\big(\mathbf{e}^T\mathbf{b} - \mathbf{e}^T(\underline{\boldsymbol{P}}[\mathbf{x}]\mathbf{b})\big)$$

Since $\underline{\boldsymbol{P}}[\mathbf{x}]$ is a sparse matrix with number of non-zeros up to the number of non-zeros in $\underline{\boldsymbol{P}}$, so the mat-vec operation $\tilde{\boldsymbol{P}}\mathbf{b}$ also only depends on the number of non-zeros in $\underline{\boldsymbol{P}}$.

For the sweep cut procedure, let $\mathbf{z}$ be the eigenvector we computed from $\tilde{\boldsymbol{P}}$ and without a loss of generality, we assume $z_1 \leq z_2 \leq \cdots \leq z_n$. We want to see what is the computational complexity for calculating $\Pr(X_1 \in \bar{S}_{k+1} | X_0 \in S_{k+1})$ when the value of $\Pr(X_1 \in \bar{S}_k | X_0 \in S_k)$ is given.

Denote the row vector $\mathbf{h} = \mathbf{e}^T - \mathbf{e}^T\mathbf{P}[\mathbf{x}]$, $H_k = \sum_{i=1}^{k} x_i h_i$ and $Q_k = \sum_{i=k}^{n} x_k$ Then given the value of:

$$\Pr(X_1 \in \bar{S}_k | X_0 \in S_k) = \frac{\Pr(X_1 \in \bar{S}_k, X_0 \in S_k)}{\Pr(X_0 \in S_k)} = \frac{p_k}{1 - Q_{k+1}}$$

To compute $\Pr(X_1 \in \bar{S}_{k+1} | X_0 \in S_{k+1})$

$$
\begin{aligned}
&= \frac{\Pr(X_1 \in \bar{S}_{k+1}, X_0 \in S_k) + \Pr(X_1 \in \bar{S}_{k+1}, X_0 = k+1)}{\Pr(X_0 \in S_k) + \Pr(X_0 = k+1)} \\
&= \frac{\Pr(X_1 \in \bar{S}_k, X_0 \in S_k) - \Pr(X_1 = k+1, X_0 \in S_k) + \Pr(X_1 \in \bar{S}_{k+1}, X_0 = k+1)}{1 - Q_{k+1} + x_{k+1}} \\
&= \frac{p_k - \sum_{i=1}^{k} x_i \tilde{\boldsymbol{P}}[\mathbf{x}]_{k+1,i} + x_{k+1} \sum_{i=k+2}^{n} \tilde{\boldsymbol{P}}[\mathbf{x}]_{i,k+1}}{1 - Q_{k+2}} \\
&= \frac{p_k - \sum_{i=1}^{k} x_i \boldsymbol{P}[\mathbf{x}]_{k+1,i} - \sum_{i=1}^{k} x_i x_{k+1} h_i + x_{k+1} \sum_{i=k+2}^{n} \boldsymbol{P}[\mathbf{x}]_{i,k+1} + x_{k+1} \sum_{i=k+2}^{n} x_i h_{k+1}}{1 - Q_{k+2}} \\
&= \frac{-\sum_{i=1}^{k} x_i \boldsymbol{P}[\mathbf{x}]_{k+1,i} + x_{k+1} \sum_{i=k+2}^{n} \boldsymbol{P}[\mathbf{x}]_{i,k+1}}{1 - Q_{k+2}} + \frac{p_k - x_{k+1} H_k + x_{k+1} h_{k+1} Q_{k+2}}{1 - Q_{k+2}}
\end{aligned}
$$

The first term only involves the $(k+1)$-th row and column of $\boldsymbol{P}[\mathbf{x}]$, and the second term cost constant computation as long as the arraies $\boldsymbol{H}$ and $\boldsymbol{Q}$ are precomputed. Since computing $\boldsymbol{H}$ and $\boldsymbol{Q}$ is $O(n)$, the total complexity of computing the above probability for all $1 \leq k \leq n$ is linear to the number of non-zeros in $\boldsymbol{P}[\mathbf{x}]$, and the same conclusion holds for $\Pr(X_1 \in \bar{S}_k | X_0 \in \bar{S}_k)$. So in summary the Sweep Cut procedure costs order of total number of non-zeros in $\boldsymbol{P}[\mathbf{x}]$.

B.2 Computational complexity

We now provide an analysis of the running time of our algorithm. Let N be the number of non-zeros in the tensor $\underline{\boldsymbol{T}}$. First, note that the pre-processing (tensor symmetrization and finding empty nodes) takes $O(N)$ time. Now, we examine the computational complexity of a single partition:

1. Generating the transition tensor $\underline{\boldsymbol{P}}$ costs $O(N)$.
2. Each step of the fixed-point algorithm to find the stationary distribution is $O(N)$.
3. Constructing $\underline{\boldsymbol{P}}[\mathbf{x}]$ costs $O(N)$. (The matrix $\underline{\tilde{\boldsymbol{P}}}$ is not formed explicitly).
4. Each iteration of the eigenvector computation takes time linear in the number of non-zeros in $\underline{\boldsymbol{P}}[\mathbf{x}]$, which is $O(N)$.
5. Sorting the eigenvector takes $O(n \log n)$ computations, which is negligible considering N is big compared to n.
6. The sweep cut takes time $O(n + N)$, which is $O(N)$.

In practice, we find that only a few iterations are needed to compute the stationary distribution, which is consistent with past results [4, 15]. For these systems, we do not know how many iterations are needed for the eigenvector computations. However, for the datasets we analyze in this paper, the eigenvector computation is not prohibitive. Thus, we can think of the time for each cut as roughly linear in the number of non-zeros. Provided that the cuts are roughly balanced, the depth of the recursion tree is $O(\log N)$, and the total time is $O(N \log N)$. Again, in our experiments, this is the case.

To backup our analysis, we tested the scalability of our algorithm on a data tensor of English 3-grams. We varied the number of non-zeros in the data tensor from five million down to a few hundred by removing non-zeroes uniformly at random. We used a laptop with 8GB of memory and 1.3GHz of CPU to run Algorithm 1 on these tensors with `max-size` $= 100$, `min-size` $= 5$, and $\phi^* = 0.4$. Figure 3 shows the results, and we see that the scaling is roughly linear.

C Experiment discussion

C.1 Clustering methods and evaluation for synthetic experiment

We compared the results of our GTSC framework to several other state-of-the-art methods for clustering tensor data.

GTSC. This is the method presented in this paper (Algorithm 1). We use the parameters $\alpha = 0.8$, `max-size` $= 100$, `min-size` $= 5$, and $\phi^* = 0.35$.[6]

[6] We tested several values $\phi^* \in [0.3, 0.4]$ and obtained roughly the same results.

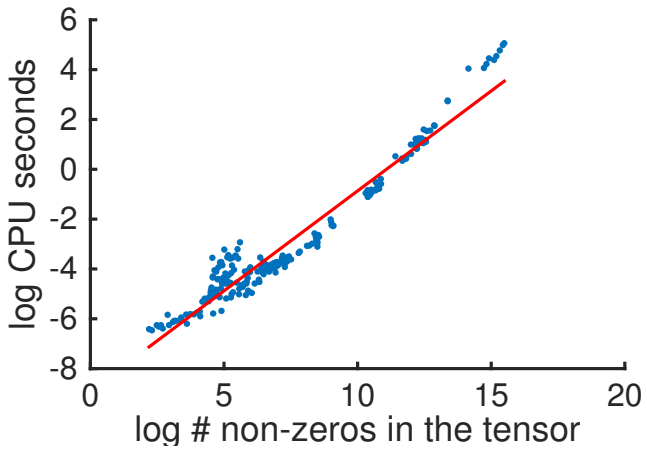

Figure 3: Time to compute a partition on the English 3-grams as a function of the number of non-zeros in the tensors. We see that the algorithm scales roughly linearly in the number of non-zeros (the red line).

TSC. This is the original tensor spectral clustering algorithm [4]. We use the algorithm with recursive bisection to find 20 clusters.

PARAFAC. The PARAFAC method is a widely used tensor decomposition procedure [16] that finds an approximation to the tensor by the sum of outer-products of vectors. We compute a rank-20 decomposition using the Tensor Toolbox [1, 8], and then assign nodes to clusters by taking the index of the vector with highest value in the nodes index. We use the default tolerance of 10^{-4} and a maximum of 1000 iterations.

Spectral Clustering (SC). Our clustering framework (Algorithm 1) works on mode-2 tensors, i.e., matrices. In this case, with $\alpha = 1$, our algorithm reduces to a standard spectral clustering method. We create a matrix $\boldsymbol{M}$ from the tensor data $\underline{\boldsymbol{T}}$ by summing along the third mode: $M_{ij} = \sum_{k=1}^{n} \underline{T}_{i,j,k}$. We then run Algorithm 1 with the same parameters as GTSC.

Multilinear Decomposition (MulDec). This is the higher-way co-clustering method for tensor decomposition [26]. The parameter λ is set to be 0. We compute the rank-20 decomposition and recover the clusters.

Evaluation metrics. We evaluate the clustering results using the Adjusted Rand Index (ARI) [19], Normalized Mutual Information (NMI) [22], and F1 score. The ground truth labels correspond to the generated groups.

C.2 Real-world dataset

In all of our experiments, we use the stopping criterion $\alpha = 0.8$, $\phi^* = 0.4$, `max-size` $= 100$ and `min-size` $= 5$ for Algorithm 1.

Table 2 shows the airline-airport co-clusters our GTSC finds. The other two methods (i.e., SC and TSC) cannot find the *Worldwide metropolises* group, which is analogous to the "stop word" group in the n-gram experiments. In particular TSC Even has trouble accurately identifying the regional clusters. Other methods like PARAFAC and MulDec cannot handle the dataset of this size. Although the number of non-zeros in this dataset is only $51,982$, PARAFAC and MulDec are unable to utilize the sparse structure of the tensor. As a result the data size is $4,655,731,619$ (i.e.,$539 \times 2939 \times 2939$) to them.

C.3 Extra experiemt - Enron email tensor

Due to space limit of the paper, we put our experiment results for Enron dataset here in Appendix.

Enron email. This dataset is constructed from emails between Enron employees with labeled topics [6]. The tensor data represents the volume of communication between two employees discussing

Table 2: High-level descriptions of the larger co-clusters found by our GTSC framework on the Airline-airport dataset. The algorithm finds one co-cluster of international hubs and large commercial airlines and several geographically coherent groups.

Name	# Airports	# Airlines	Airports description	Airlines description
Worldwide metropolises	250	77	Large hubs, e.g., Beijing Capital and JFK in New York	Large commercial airlines, e.g., United, Air Canada, Air China
Europe	184	32	177 in Europe, rest in Morocco	29 European airlines
United States	137	9	136 in U.S., Cancún International	29 all U.S. airlines
China/Taiwan	170	33	136 in China or Taiwan,	21 in China/Taiwan 14 in S. Korea and Thailand
Oceania/S.E. Asia	302	52	231 in Oceania or S.E. Asia,	41 in East Asia or Canada 66 in China, Japan, or Canada
Mexico/Americas	399	68	396 in Mexico or Central and South America	43 in Mexico or Central and South America

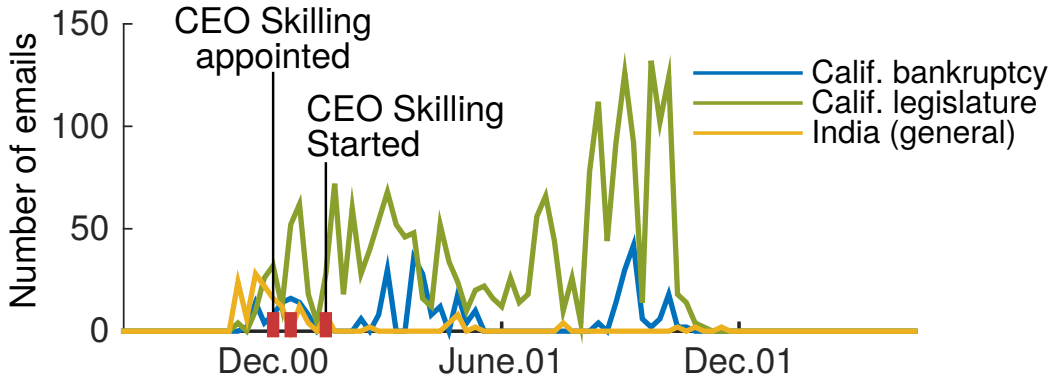

Figure 4: Enron email volume on three labeled topics. Our GTSC framework finds a co-cluster consisting of these three topics at the time points labeled in red, which seems to correlate with various events involving the CEO.

a given topic during a particular week. In total, there are 185 weeks of data, 184 employees, and 34 topics, leading to a $185 \times 184 \times 184 \times 34$ tensor where $\underline{T}_{ijkl}$ is the number of emails between employee j and k on topic l during week i.

In total, the algorithm finds 23 co-clusters of topics, people, and time. The most popular group corresponds to three topics, 19 people, and 0 time intervals. Similar to the n-grams and airport-airline data, this cluster corresponds to high-volume entities, in this case common topics and people who send a lot of emails. The three topics are “Daily business", “too few words", and “no matching topic", which account for roughly 90% of the total email volume. (The latter two topics are capturing outliers and emails that do not fall under an obvious category). The 19 employees include 11 managers: the CEO, (vice) presidents, and directors. These employees are involved in 42% of the total emails. There is no time interval in this co-cluster because these high-volume topics and employees are balanced throughout time.

We also found several interesting co-clusters. One consists of the topics “California bankruptcy", “California legislature", and “India (general)", during three weeks in December 2000 and January 2001, and 13 employees. These time points correspond to various events involving CEO Skilling (Figure 4). Each of the 13 employees in the co-cluster sent at least one email from at least one of the topics. Another co-cluster consists of the topics “General newsfeed", “Downfall newsfeed", and

"Federal Energy Regulatory Commission/Department of Energy" and several weeks from March 2001 and December 2001. These time intervals coincide with investor James Chanos finding problems with Enron in early 2001 and the serious financial troubles encountered by the company in late 2001.

D Other discussion

D.1 Dangling correction effect

The TSC framework [4] used a pre-defined stochastic dangling vector $\mathbf{u}$ when encountering a zero column of the tensor $\underline{\mathbf{P}}$. So the transition tensor is

$$\underline{\tilde{P}}_{ijk} = \begin{cases} \underline{P}_{ijk} & \text{if } \sum_{i=1}^{n} \underline{T}_{ijk} > 0 \\ u_i & \text{otherwise} \end{cases}$$

Denote $\tilde{\mathbf{R}}$ and $\mathbf{R}$ the mode-1 unfolding of tensor $\underline{\tilde{\mathbf{P}}}$ and $\underline{\mathbf{P}}$ respectively. Then we have

$$\tilde{\mathbf{R}} = \mathbf{R} + \mathbf{u}(\mathbf{e}^T - \mathbf{e}^T\mathbf{R}).$$

The multilinear Pagerank vector $\mathbf{x}$ of $\tilde{\mathbf{R}}$ from Equation (5) satisfies:

$$\begin{aligned} \mathbf{x} &= \alpha\big(\mathbf{R} + \mathbf{u}(\mathbf{e}^T - \mathbf{e}^T\mathbf{R})\big)\mathbf{x} \otimes \mathbf{x} + (1-\alpha)\mathbf{v} \\ &= \alpha\mathbf{R}\mathbf{x} \otimes \mathbf{x} + \alpha\mathbf{u}(1 - \mathbf{e}^T\mathbf{R}\mathbf{x} \otimes \mathbf{x} + (1-\alpha)\mathbf{v} \\ &= \alpha\mathbf{R}\mathbf{x} \otimes \mathbf{x} + \alpha(1 - \|\mathbf{R}\mathbf{x} \otimes \mathbf{x}\|_1)\mathbf{u} + (1-\alpha)\mathbf{v} \end{aligned}$$

where $\|\cdot\|_1$ denotes the $1-$norm of a vector.

When the tensor is very sparse, there are lots of zero columns in $\mathbf{R}$, and $\|\mathbf{R}\mathbf{x} \otimes \mathbf{x}\|_1$ can be quite small. Thus, the contribution of the dangling vector $\mathbf{u}$ becomes significant. In prior work, $\mathbf{u}$ is simply chosen as the uniform vector, and thus it washes out information about the structure in $\mathbf{R}$.

With the dangling vector, the spacey random surfer ends up guessing some history steps that are not feasible, i.e., states corresponding to zero columns in $\mathbf{R}$.

References

[1] B. W. Bader, T. G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015.

[2] B.-K. Bao, W. Min, K. Lu, and C. Xu. Social event detection with robust high-order co-clustering. In *ICMR*, pages 135–142, 2013.

[3] M. Benaïm. Vertex-reinforced random walks and a conjecture of pemantle. *Ann. Prob.*, 25(1):361–392, 1997.

[4] A. R. Benson, D. F. Gleich, and J. Leskovec. Tensor spectral clustering for partitioning higher-order network structures. In *SDM*, pages 118–126, 2015.

[5] A. R. Benson, D. F. Gleich, and L.-H. Lim. The spacey random walk: a stochastic process for higher-order data. *arXiv*, cs.NA:1602.02102, 2016.

[6] M. W. Berry, M. Browne, and B. Signer. Topic annotated enron email data set. *LDC*, 2001.

[7] D. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[8] E. C. Chi and T. G. Kolda. On tensors, sparsity, and nonnegative factorizations. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1272–1299, 2012.

[9] F. Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005. 10.1007/s00026-005-0237-z.

[10] F. Chung. Four proofs for the Cheeger inequality and graph partition algorithms. In *ICCM*, 2007.

[11] F. R. L. Chung. *Spectral Graph Theory*. AMS, 1992.

[12] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD*, pages 269–274, 2001.

[13] B. Gao, T.-Y. Liu, X. Zheng, Q.-S. Cheng, and W.-Y. Ma. Consistent bipartite graph co-partitioning for star-structured high-order heterogeneous data co-clustering. In *KDD*, pages 41–50, 2005.

[14] D. Ghoshdastidar and A. Dukkipati. Spectral clustering using multilinear svd: Analysis, approximations and applications. In *AAAI*, pages 2610–2616, 2015.

[15] D. F. Gleich, L.-H. Lim, and Y. Yu. Multilinear PageRank. *SIAM J. Matrix Ann. Appl.*, 36(4):1507–1541, 2015.

[16] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[17] M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram. The total variation on hypergraphs-learning on hypergraphs revisited. In *Advances in Neural Information Processing Systems*, pages 2427–2435, 2013.

[18] H. Huang, C. Ding, D. Luo, and T. Li. Simultaneous tensor subspace selection and clustering: the equivalence of high order SVD and k-means clustering. In *KDD*, pages 327–335, 2008.

[19] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[20] S. Jegelka, S. Sra, and A. Banerjee. Approximation algorithms for tensor clustering. In *Algorithmic learning theory*, pages 368–383. Springer, 2009.

[21] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 2014.

[22] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New J. Phys.*, 11(3):033015, 2009.

[23] M. Meilă and J. Shi. A random walks view of spectral segmentation. In *AISTATS*, 2001.

[24] M. Mihail. Conductance and convergence of markov chains—a combinatorial treatment of expanders. In *FOCS*, pages 526–531, 1989.

[25] J. Ni, H. Tong, W. Fan, and X. Zhang. Flexible and robust multi-network clustering. In *KDD*, pages 835–844, 2015.

[26] E. E. Papalexakis and N. D. Sidiropoulos. Co-clustering as multilinear decomposition with sparse latent factors. In *ICASSP*, pages 2064–2067. IEEE, 2011.

[27] E. A. Pechenick, C. M. Danforth, and P. S. Dodds. Characterizing the Google Books corpus: strong limits to inferences of socio-cultural and linguistic evolution. *PloS one*, 10(10):e0137041, 2015.

[28] S. Ragnarsson and C. F. Van Loan. Block tensors and symmetric embeddings. *Linear Algebra Appl.*, 438(2):853–874, 2013.

[29] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[30] W. J. Stewart. *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press, 1994.

[31] D. Wagner and F. Wagner. Between min cut and graph bisection. In *MFCS*, pages 744–750, 1993.

[32] D. Zhou and C. J. Burges. Spectral clustering and transductive learning with multiple views. In *ICML*, pages 1159–1166, 2007.