EXTRACTING CLUSTERS FROM LARGE DATASETS WITH MULTIPLE SIMILARITY MEASURES USING IMSCAND

TERESA M. SELEE[*], TAMARA G. KOLDA[†], W. PHILIP KEGELMEYER[‡], AND JOSHUA D. GRIFFIN[§]

Abstract. We consider the problem of how to group information when multiple similarities are known. For a group of people, we may know their education, geographic location and family connections and want to cluster the people by treating all three of these similarities simultaneously. Our approach is to store each similarity as a slice in a tensor. The similarity measures are generated by comparing features. Generally, the object similarity matrix is dense. However it can be stored implicitly as the product of a sparse matrix, representing the object-feature matrix, and its transpose. For this new type of tensor where dense slices are stored implicitly, we have created a new decomposition called Implicit Slice Canonical Decomposition (IMSCAND). Our decomposition is equivalent to the tensor CANDECOMP/PARAFAC decomposition, which is a higher-order analogue of the matrix Singular Value decomposition (SVD) and Principal Component Analysis (PCA). From IMSCAND we obtain compilation feature vectors which are clustered using *k*-means. We demonstrate the applicability of IMSCAND on a set of journal articles with multiple similarities.

1. Introduction.

Datasets naturally have mulitple similarities. For a group of people, we might know their age, education, geographic location, and social and family connections. For a set of published papers, we know the authors, citations, and terms in the abstract, title, and keywords. Even for a computer hard drive, we know the names of the files, their saved location, their time stamp, and their contents. For each of these examples, if we wanted to find a way to cluster the people, documents, or computer files, our approach is to treat all the similarities concurrently.

For each similarity, a similarity matrix is formed, with objects (people, documents, files, etc.) as both the rows and columns. Each of the similarity matrices is a slice in a tensor, and a tensor decomposition is used to assemble the multiple adjacency matrices into a set of compilation feature vectors. These feature vectors can then be clustered using the *k*-means clustering algorithm. This approach was used by Dunlavy et al. [6].

Our approach to this problem is special because of how we form the similarity matrices. In general, similarity matrices are dense, limiting the number of objects and features because of the large amount of work required to compute a decomposition for a dense tensor. We form the dense similarity matrices implicitly by storing only sparse object-feature matrices. For example, an article-author matrix is a sparse object-feature matrix with documents as rows and authors as columns of the matrix. Since most articles have just a few authors, we can see how this is a sparse matrix. Then the adjacency matrix is stored implicitly as the product of the sparse object-feature matrix and its transpose (a feature-object matrix).

Our new decomposition is called Implicit Slice Canonical Decomposition (IMSCAND) and does all of its computations on the sparse matrices only. This allows us to treat much larger problems than if we stored the full similarity matrices. We illustrate the effectiveness of our decomposition on a set of journal publications from the Society of Industrial and Applied Mathematics (SIAM). Our long-term goal is to use this approach to cluster files on computer hard drives.

2. Tensors.

In this paper we focus on third-order tensors. These are denoted by boldface Euler script letters, e.g., $\mathcal{X}$. When we say order, way, or mode we are referring to the number of dimensions of the tensor. A third-order tensor, $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is illustrated in Figure 2.1.

[*]Department of Mathematics, North Carolina State University, tmselee@ncsu.edu
[†]Sandia National Laboratories, tgkolda@sandia.gov
[‡]Sandia National Laboratories, wpk@sandia.gov
[§]Sandia National Laboratories, jdgriffi@sandia.gov

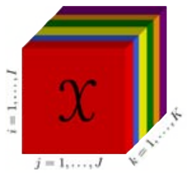
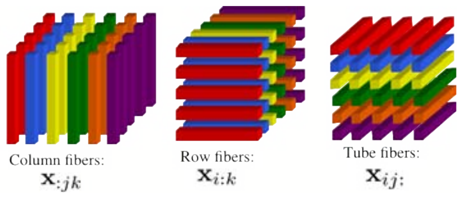


FIG. 2.1. *Third-order tensor* $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$

Matrices are denoted by boldface capital letters, e.g., $\mathbf{A}$. Vectors are denoted by boldface lowercase letters, e.g., $\mathbf{a}$. Scalars are denoted by lowercase letters, e.g., a. The i^{th} entry of $\mathbf{a}$ is a_i. The i^{th} row of $\mathbf{A}$ is $\mathbf{a}_{i:}$, the j^{th} column of $\mathbf{A}$ is $\mathbf{a}_{:j}$ or sometimes just $\mathbf{a}_j$. Finally, the (i, j) entry of $\mathbf{A}$ is a_{ij}.

We have similar notation for tensors. The higher-order analogue of rows and columns are fibers. A column vector is a mode-1 or column fiber, denoted $\mathbf{x}_{:jk}$. A row vector is a mode-2 or row fiber, denoted $\mathbf{x}_{i:k}$. Vectors in the third dimension are tube fibers, written $\mathbf{x}_{ij:}$. These are illustrated in Figure 2.2.



Column fibers: $\mathbf{x}_{:jk}$
Row fibers: $\mathbf{x}_{i:k}$
Tube fibers: $\mathbf{x}_{ij:}$

FIG. 2.2. *Fibers of a third order tensor* $\mathcal{X}$.

We must also discuss the notation for two-dimensional slices. Slices of a tensor are matrices and can be horizontal, lateral, or frontal. They are denoted as $\mathbf{X}_{i::}, \mathbf{X}_{:j:}$ and $\mathbf{X}_{::k}$, respectively. The notation $\mathbf{X}_k$ is also used to denote the k^{th} frontal slice of a tensor. These are displayed in Figure 2.3.

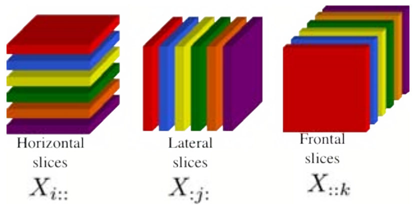


FIG. 2.3. *Slices of a third order tensor.*

In addition, there are tensor symbols we need to define. The symbol $\circ$ denotes the outer product of vectors. For example, for $\mathbf{a} \in \mathbb{R}^I$, $\mathbf{b} \in \mathbb{R}^J$ and $\mathbf{c} \in \mathbb{R}^K$, we can form a tensor $\mathcal{X}$ from the outer product, $\mathcal{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ where $x_{ijk} = a_i b_j c_k$ for all $i = 1, \ldots, I$, $j = 1, \ldots, J$, and $k = 1, \ldots, K$. We define the Hadamard (i.e., elementwise) matrix product using the symbol $*$. The Khatri-Rao product [18, 22, 2, 24] is a columnwise Kronecker product. For two matrices $\mathbf{A} \in \mathbb{R}^{I \times P}$ and $\mathbf{B} \in \mathbb{R}^{J \times P}$, the Khatri-Rao product is

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_P \otimes \mathbf{b}_P \end{bmatrix},$$

where $\mathbf{a} \otimes \mathbf{b}$ denotes the matrix Kronecker product.

For some computations, it is necessary to treat the entire tensor in matrix form. To do this we go through a process called matricization. We define $\mathbf{X}_{(1)}$ to the be the mode-1 matricization of $\mathcal{X}$. This means the mode-1 fibers (the column fibers) are aligned to form a matrix. Specifically, the mode-1 fibers are mapped to the rows of a matrix, and the modes-2 and -3 fibers are mapped to the columns of the matrix. See Figure 2.4 for an illustration of mode-1 matricization. We also look at $\mathbf{X}_{(2)}$ and $\mathbf{X}_{(3)}$, the mode-2 (row) and mode-3 (tube) matriciza-

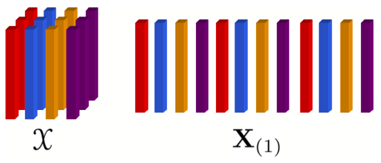


FIG. 2.4. *Example of a mode-1 matricization, $\mathbf{X}_{(1)}$. The column fibers are aligned to form a matrix, with the mode-1 fibers as rows in our matrix, and the modes-2 and -3 fibers as the columns of the matrix.*

tions, respectively. The ordering of the grouped modes makes a difference. Although we do not indicate it explicitly, we assume $\mathbf{X}_{(1)} = \mathbf{X}_{(\{1\}\times\{2,3\})}$, $\mathbf{X}_{(2)} = \mathbf{X}_{(\{2\}\times\{1,3\})}$, and $\mathbf{X}_{(3)} = \mathbf{X}_{(\{3\}\times\{1,2\})}$, per the notation of [14]. This ordering is not universal; see, e.g., [5, 13].

The tensor norm is defined as the square root of the sum of the squares of all the elements of a tensor. For $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$,

$$\|\mathcal{X}\|^2 = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} x_{ijk}^2.$$

This is the higher-order analogue of the matrix Frobenius norm.

For more information on tensor notation, see [13, 9, 1, 14].

3. Special types of tensors.

We will discuss two special types of tensors in this paper: Kruskal tensors and sp3way tensors.

3.1. Kruskal tensors.

Kruskal tensors, named for Kruskal [15, 16], are tensors that are stored as the sum of R rank-1 tensors. If $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ is a Kruskal tensor, it is stored as:

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r,$$

where $\mathbf{A} \in \mathbb{R}^{I\times R}$, $\mathbf{B} \in \mathbb{R}^{J\times R}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$. The notation $\mathbf{a}_r$ denotes the r^{th} column of a matrix $\mathbf{A}$. An illustration of this idea is in Figure 3.1. This type of tensor results from a CAN-

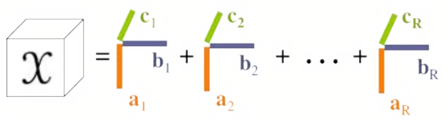


FIG. 3.1. *A third-order Kruskal tensor $\mathcal{X}$ is written as the sum of R rank-1 tensors.*

DECOMP/PARAFAC decomposition [3, 7], described in section 4. We use the shorthand notation from [14]:

$$\mathcal{X} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!],$$

though other notation can be used. For instance, Kruskal [15] uses

$$\mathcal{X} = (\mathbf{A}, \mathbf{B}, \mathbf{C}) .$$

It is also possible to have explicit weights $\lambda \in \mathbb{R}^R$. Then

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \, \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \quad \text{and} \quad \mathcal{X} = [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!].$$

3.2. A new class of tensors: sp3way tensors. We have created a new class of tensors that are third-order (3-way) with special structure. For this type of tensor, each slice is dense, but formed from the product of a sparse matrix and its transpose. Specifically, each (frontal) slice of $\mathcal{X} \in \mathbb{R}^{N \times N \times P}$ is written, $\mathbf{X}_p = \mathbf{Y}_p \mathbf{Y}_p^T$ for sparse $\mathbf{Y}_p$, with $p = 1, \ldots, P$. This is shown in Figure 3.2.

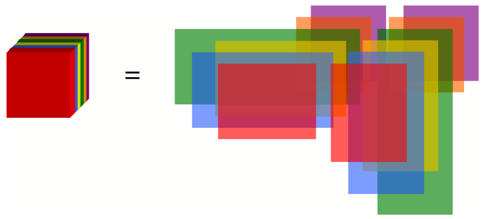


FIG. 3.2. *An sp3way tensor, in which each slice of the tensor, $\mathcal{X}$, is formed from the product of a sparse matrix and its transpose, $\mathbf{X}_p = \mathbf{Y}_p \mathbf{Y}_p^T$ for sparse $\mathbf{Y}_p$, with $p = 1, \ldots, P$.*

The motivation for this type of tensor came from wanting to store multiple similarity matrices simultaneously. Our idea is to view the similarity (object-object) matrices as formed from the product of an object-feature matrix and its transpose. For the tensor $\mathcal{X} \in \mathbb{R}^{N \times N \times P}$ we have N objects and P features. In general, object-feature matrices are sparse. The slices of our final tensor are object-object matrices which can be dense, and each slice is a different similarity matrix. When we do computations on sp3way tensors, everything is done on the sparse matrices directly.

4. CANDECOMP/PARAFAC (CP) and INDSCAL. Canonical Decomposition (CANDECOMP) [3] and Parallel Factors (PARAFAC) [7] are two different names for the same tensor decomposition, first published in 1970, and often abbreviated CP so as to give credit to both names. This decomposition is a higher-order analogue of the matrix Singular Value Decomposition (SVD) or matrix Principal Component Analysis (PCA). For a general third-order tensor, a rank-R CP decomposition approximates a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ by a Kruskal tensor $\mathcal{K}$. Since $\mathcal{X}$ is a 3-way tensor, the Kruskal tensor is written $\mathcal{K} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$.

The standard algorithm for computing a CP decomposition employs Alternating Least Squares (ALS) [3]. The CP-ALS algorithm requires two matrices to begin, say $\mathbf{A}$ and $\mathbf{B}$. We then iterate through three steps to compute the matrices $\mathbf{C}$, $\mathbf{B}$, and $\mathbf{A}$ which form the Kruskal tensor. The algorithm proceeds as follows.

1. Holding $\mathbf{A}$ and $\mathbf{B}$ constant, solve for $\mathbf{C}$:

$$\mathbf{C} = \mathbf{X}_{(3)} \left(\mathbf{B} \odot \mathbf{A}\right) \left(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A}\right)^{\dagger} .$$

2. Holding $\mathbf{A}$ and $\mathbf{C}$ constant, solve for $\mathbf{B}$:

$$\mathbf{B} = \mathbf{X}_{(2)} \left(\mathbf{C} \odot \mathbf{A}\right) \left(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A}\right)^{\dagger} .$$

3. Holding **B** and **C** constant, solve for **A**:

$$\mathbf{A} = \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B}) \left(\mathbf{C}^T \mathbf{C} * \mathbf{B}^T \mathbf{B} \right)^{\dagger}.$$

We iterate through the three steps, computing new values for $\mathbf{C}, \mathbf{B}$, and $\mathbf{A}$, until the fit of the Kruskal tensor to the original tensor ceases to improve or we reach our maximum number of allowed iterations.

4.1. Symmetric CP. In the original paper by Carroll and Chang [3], in addition to introducing the CANDECOMP decomposition of a tensor, they also introduce individual differences in scaling (INDSCAL), which is a symmetric CP decomposition. To employ INDSCAL, the frontal slices of the tensor $\mathcal{X} \in \mathbb{R}^{N \times N \times P}$ must be symmetric. The major difference is that, unlike normal (not necessarily symmetric) CP-ALS iterations, there is no constraint to make $\mathbf{A} = \mathbf{B}$. However, when we have symmetry in the frontal slices ($x_{ijk} = x_{jik}$ for all i, j, k), we are guaranteed to have **A** and **B** equal, or at least related by a diagonal transformation, **D**, by the time the algorithm has converged [3]. Specifically,

$$\mathbf{A} = \mathbf{D}\mathbf{B}$$
$$\mathbf{B} = \mathbf{D}^{-1}\mathbf{A}$$

where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is a diagonal matrix with nonzero diagonal elements.

Two alternatives to the CP-ALS algorithm exist to explicitly obtain $\mathbf{A} = \mathbf{B}$, even if the frontal slices of the tensor are not symmetric [3]. The first option is not recommended [3] since its properties are not well understood. In this option, the third step becomes:

3. Set $\mathbf{A} = \mathbf{B}$.

Then, as before, we iterate through steps 1, 2, and 3. A better alternative is to follow the initially laid out steps until the tolerance is reached. Then set $\mathbf{A} = \mathbf{B}$, and solve for **C** one final time.

4.2. Uniqueness Conditions for CP. There is a long history on uniqueness conditions for decompositions, and Stegman, et al. [25] give a good summary. The first uniqueness results on CP are credited to Jennrich (in the Acknowledgments section by Harshman [7], 1970) and Harshman [8], 1972. The most general sufficient condition for uniqueness is due to Kruskal [15] in 1977. Specifically, Kruskal defined the k-rank of a matrix, which is the largest number k such that every subset of k columns of the matrix is linearly independent. We denote the k-rank of a matrix **A** as $k_{\mathbf{A}}$. Kruskal's condition states:

$$k_{\mathbf{A}} + k_{\mathbf{B}} + k_{\mathbf{C}} \geq 2R + 2$$

is sufficient for uniqueness of CP, where $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, written $\mathcal{X} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, is rank R.

Some additional uniqueness conditions have been developed more recently. In 2000, Sidiropoulos and Bro [23] illustrated a short-cut proof for the uniqueness conditions and generalized the result to n-way tensors (for $n > 3$). In 2002, ten Berge and Sidiropoulos [26] proved that Kruskal's sufficiency condition is also necessary for $R = 2$ and $R = 3$, but not for $R > 3$, (uniqueness for $R = 1$ was proved by Harshman [8]). Some alternate uniqueness conditions are given by Jiang and Sidiropoulos, and De Lathauwer in [12, 4]. These two references separately examine the case where one of the component matrices is of full column rank. Each paper assumes three conditions (though not the same conditions). Stegman, et al. [25], take the same approach as in [4], however [25] should be more accessible since there are no fourth order tensors and the only requirement is basic linear algebra. In addition, Stegman et al. [25] state distinct general uniqueness conditions for CP and INDSCAL. For

INDSCAL, stricter uniqueness conditions in terms of R are obtained because the model has less parameters. The conditions for CP are related to those from [12, 4].

The major idea from Stegman, et al. [25] is that we almost surely have uniqueness for the CP and INDSCAL decompositions when $\mathbf{A}$ and $\mathbf{B}$ are randomly sampled from a continuous distribution and $\mathbf{C}$ has full rank. Further, in a majority of situations, the CP and INDSCAL algorithms can be thought of as randomly sampled from a continuous distribution. Thus, the uniqueness conditions should, but do not always, apply.

5. Implicit Slice Canonical Decomposition (IMSCAND). The motivation for a new decomposition comes from the new sp3way tensors, which allow us to work with a dense tensor, but only requires us to store sparse matrices. We call this new decomposition IMSCAND for Implicit Slice Canonical Decomposition. This decomposition gives a canonical decomposition of a tensor whose slices are formed implicitly as the product of a sparse matrix and its transpose.

Both CP and IMSCAND produce the same decomposition of a tensor into a Kruskal tensor using the same number of iterations. The decompositions have a major difference however. IMSCAND stores sparse matrices, $\mathbf{Y}_i$, which are implicitly multiplied by their transpose to form the frontal slices of the tensor. Specifically, the frontal slices of $\mathcal{X} \in \mathbb{R}^{N\times N\times P}$ are

$$\mathbf{X}_p = \mathbf{Y}_p\mathbf{Y}_p^T \text{ for } p = 1, \ldots, P.$$

All computations are done on the sparse matrices directly. CP, on the other hand, stores fully formed slices, which can be dense. Because all the computations are done for sparse matrices, IMSCAND is capable of handling much larger problems than CP in less time.

The algorithm for computing IMSCAND differs from the CP-ALS algorithm at three main points. The difference occurs in the calculations of $\mathbf{X}_{(3)}(\mathbf{B}\odot\mathbf{A})$, $\mathbf{X}_{(2)}(\mathbf{C}\odot\mathbf{A})$, and $\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})$. The CP-ALS algorithm follows:

1. Holding $\mathbf{A}$ and $\mathbf{B}$ constant, solve for

$$\mathbf{C} = \widehat{\mathbf{C}}\left(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A}\right)^{\dagger}, \text{ with } \widehat{\mathbf{C}} = \mathbf{X}_{(3)}(\mathbf{B}\odot\mathbf{A}).$$

2. Holding $\mathbf{A}$ and $\mathbf{C}$ constant, solve for

$$\mathbf{B} = \widehat{\mathbf{B}}\left(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A}\right)^{\dagger}, \text{ with } \widehat{\mathbf{B}} = \mathbf{X}_{(2)}(\mathbf{C}\odot\mathbf{A}).$$

3. Holding $\mathbf{B}$ and $\mathbf{C}$ constant, solve for

$$\mathbf{A} = \widehat{\mathbf{A}}\left(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B}\right)^{\dagger} \text{ with } \widehat{\mathbf{A}} = \mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B}).$$

5.1. Computing the product of a tensor and 2 vectors in 2 modes. The entire point of the IMSCAND decomposition is to not form any of the $\mathbf{X}_i$ directly. Thus we have different ways to compute $\widehat{\mathbf{C}}$, $\widehat{\mathbf{B}}$ and $\widehat{\mathbf{A}}$. These are all computed using the sparse $\mathbf{Y}_i$ object-feature matrices directly.

We compute $\widehat{\mathbf{C}} \in \mathbb{R}^{P\times R}$ elementwise using $\mathbf{A}$ and $\mathbf{B}$. Define $\mathbf{b}_r$ as the r^{th} column of $\mathbf{B}$, and $\mathbf{a}_r$ as the r^{th} column of $\mathbf{A}$. Then the (p, r) element of $\widehat{\mathbf{C}}$ is

$$\widehat{c}_{pr} = \left(\mathbf{Y}_p^T\mathbf{b}_r\right)^T\left(\mathbf{Y}_p^T\mathbf{a}_r\right), \text{ for } p = 1, \ldots, P \text{ and } r = 1, \ldots, R.$$

Both P and R are relatively small.

The values for $\widehat{\mathbf{B}}$ and $\widehat{\mathbf{A}}$ are computed with the same formula. Unlike $\widehat{\mathbf{C}}$ which is computed elementwise, these are both computed columnwise. We assume there are P different features and thus P slices in our tensor. Then, using $\mathbf{C}$, $\mathbf{A}$, and notation from $\widehat{\mathbf{C}}$,

$$\widehat{\mathbf{b}}_r = \sum_{p=1}^{P} c_{pr} \left[\mathbf{Y}_p \left(\mathbf{Y}_p^T \mathbf{a}_r \right) \right].$$

And using $\mathbf{B}$ and $\mathbf{C}$,

$$\widehat{\mathbf{a}}_r = \sum_{p=1}^{P} c_{pr} \left[\mathbf{Y}_p \left(\mathbf{Y}_p^T \mathbf{b}_r \right) \right].$$

6. Numerical Results. We have analyzed both real and simulated data. Our main goal is to gain intuition for the optimal choice of the IMSCAND rank and to increase our understanding of the k-means clustering algorithm.

Our data sets are too large and dense for previous tensor decompositions. Thus we have established the new IMSCAND decomposition that exploits the structure of the data to make the necessary computations possible. Specifically, instead of storing the slices, which are dense object-object similarity matrices, we instead store the sparse object-feature matrices which are multiplied by their transpose to form the slices. From the IMSCAND decomposition, we obtain matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. We start with symmetric data so that our output has the property $\mathbf{A} \approx \mathbf{B}$ (we only get equality when CP converges). This is important, since the rows of both of these matrices contain feature vectors, which are a compilation of all of the different similarities. We want the same feature vectors since $k-$means clustering is used on these vectors to obtain a clustering of our data. We use the matrix $\mathbf{A}$ to obtain our feature vectors.

6.1. Randomly Generated Data. Our simulated data was created to produce an sp3way tensor with pre-determined clusters. Specifically, the user determines the number of nodes, number of different similarities (= number of frontal slices), number of clusters, number of objects in each cluster, number of features in each cluster for each different type of feature, and the probabilities that two nodes are in the same cluster or in different clusters for each type of feature.

For example, consider the arranged $5 \times 5 \times 2$ sp3way tensor, $\mathcal{X}$ with 3 clusters. The tensor is formed from object-feature matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$, with

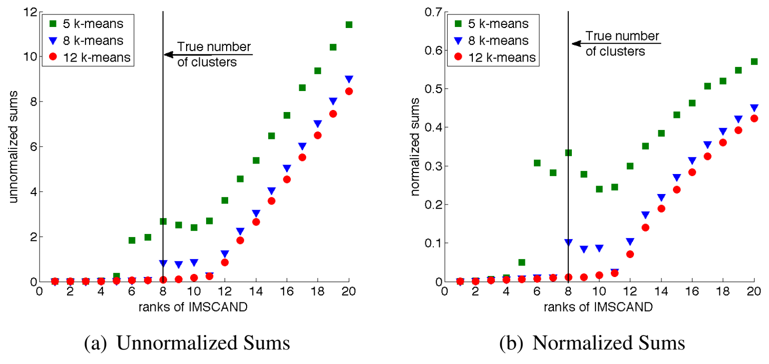
$$\mathbf{Y}_1 = \left(\begin{array}{c|cc|c} * & 0 & 0 & 0 \\ * & 0 & 0 & 0 \\ \hline 0 & * & * & 0 \\ \hline 0 & 0 & 0 & * \\ 0 & 0 & 0 & * \end{array} \right) \qquad \mathbf{Y}_2 = \left(\begin{array}{c|ccc|cc} * & 0 & 0 & 0 & 0 & 0 \\ * & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & * & * & * & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * & * \end{array} \right).$$

For both of these matrices, the stars are in-cluster values, and the zeros are out-of-cluster values. The out of cluster values do not have to be zero, and the in-cluster values can be zero. The code is designed so the the clusters are denser than the non-cluster areas. After creating the slices, we rearrange the rows to make the clusters less obvious and the algorithm nontrivial.

Both object-feature matrices illustrate that there are two objects in the first cluster, one in the second cluster, and two in the third cluster. For the first similarity, $\mathbf{Y}_1$, we see that there is one feature in the first cluster, two features in the second cluster, and one feature in the third cluster. For the second similarity, $\mathbf{Y}_2$ we see that there is one feature in the first cluster, three features in the second cluster, and two features in the third cluster.

As an numerical example, we have generated an sp3way tensor $\mathcal{X} \in \mathbb{R}^{1000\times1000\times6}$ with 8 clusters. The sizes for $\mathbf{Y}_i$ range for the number of features goes from 678 columns to 21342 columns. The number of objects per cluster ranges from 76 to 195, and the number of features per cluster ranges from 8% to 18% of the total number of features for that similarity. The in-cluster probabilities range from 9% to 37%, and the out-of-cluster probabilities range from 0.4% to 21%.

We considered many outputs in hopes that something would be a good indicator of an acceptable choice for the IMSCAND rank. The fit value from IMSCAND does not indicate the correct number of clusters, or the number of clusters requested from *k*-means. We also plotted 2 values from the *k*-means algorithm. The first graph in Figure 6.1(a) is the total sum of the distance between each point and its centroid. The second graph in Figure 6.1(b) is the same sum of distances, only we divide by the rank of IMSCAND or number of dimensions. The second graph is essentially a normalization of the first graph on the number of dimensions. For each graph, we computed multiple IMSCAND decompositions with ranks ranging from 1 to 20. We did three separate *k*-means clusterings, asking for the correct number of clusters (8), more (12), and less (5). In all of these, we can see a jump in the graph at the *k*-means clustering value, but not at the actual number of clusters.



(a) Unnormalized Sums (b) Normalized Sums

FIG. 6.1. *Both graphs look at the total sum of the distances from each points to its k-means centroid. The normalized sums are divided by the number of dimensions (which also corresponds to the rank of IMSCAND.*

6.2. The SIAM Journal Data. The data set we are using with IMSCAND is a set of approximately 4700 articles from eleven Society of Industrial and Applied Mathematics (SIAM) journals and SIAM proceedings (SIAM PROC S) for a five-year period from 1999 to 2004. The names of the publications used throughout this paper are the ISI abbreviations[1] for the journals. We will use the terms *article* and *document* interchangeably.

There are both explicit links and implicit links built in to the journal data. An explicit link exists when one paper cites another paper. The implicit links include connections between papers through similar authors, title words, abstract words and author-specified keywords.

For this application, we have a tensor with six frontal slices, each of which is formed

[1]`http://www.isiknowledge.com/`

from the product of a sparse article-feature matrix and its transpose. The slices are:

$$\begin{aligned}
\mathbf{X}_1 &= \text{similarity between words in the abstract}\\
\mathbf{X}_2 &= \text{similarity between names of authors}\\
\mathbf{X}_3 &= \text{similarity between author-specified keywords}\\
\mathbf{X}_4 &= \text{similarity between words in the title}\\
\mathbf{X}_5 &= \text{co-citation information}\\
\mathbf{X}_6 &= \text{co-reference information}
\end{aligned}$$

We discuss these in more detail. The first four slices are formed as $\mathbf{X}_i = \mathbf{Y}_i\mathbf{Y}_i^T$ for $i = 1, \ldots, 4$.

1. Let $\mathbf{Y}_1$ be the adjacency matrix formed using information about the abstracts of the papers. Specifically, we form a document-term matrix where the words in the abstracts are used to determine the terms. The documents are listed in the rows, and there is a word bank for the abstracts that is used to form the columns. The word bank includes all word appearing in abstracts, except for common "stop-list" words. There are more than 8000 columns in $\mathbf{Y}_1$.
2. Let $\mathbf{Y}_2$ be the adjacency matrix formed using the documents as rows and the authors as columns. There are more than 6000 authors associated with the data.
3. Let $\mathbf{Y}_3$ be the adjacency matrix formed using the documents as rows and authors-specified keywords as columns. There are less keywords than documents (just over 3000).
4. Let $\mathbf{Y}_4$ be the adjacency matrix formed using the documents as rows and a word bank of terms from the titles as columns. There are less than 3000 distinct words that appear in the titles.
5. For $\mathbf{X}_5$ (and $\mathbf{X}_6$), we need the sparse citation matrix, $\mathbf{C}$. The citation matrix is a binary matrix defined such that
   $$c_{ij} = \begin{cases} 1, & \text{if paper } i \text{ cites paper } j \\ 0, & \text{otherwise} \end{cases}.$$
   Then the co-citation matrix is defined $\mathbf{X}_5 \equiv \mathbf{C}^T\mathbf{C}$ with
   $$(\mathbf{X}_5)_{ij} = \text{ the number of papers citing both documents } i \text{ and } j.$$
6. For $\mathbf{X}_6$, as with $\mathbf{X}_5$, the sparse citation matrix, $\mathbf{C}$ is used (see 5. above). The co-reference matrix is defined $\mathbf{X}_6 \equiv \mathbf{C}\mathbf{C}^T$ with
   $$(\mathbf{X}_6)_{ij} = \text{ the number of papers cited by both documents } i \text{ and } j.$$

Although the original dataset contained 11 journals and SIAM proceedings, after initial trials we decided to remove two of the journals and SIAM proceedings. We removed the SIAM Journal on Applied Dynamical Systems because it had only 32 eligible papers in the five year span. We did not believe there would be enough information from these papers to be able to identify these documents in the clustering. We also removed SIAM Review and SIAM proceedings since they had documents spanning a variety of different topics, but lacked one unifying area by which they might later be clustered together.

Before running IMSCAND and *k*-means, we first normalized the slices by dividing each row of each slice by its 2-norm. The first four slices have naturally higher overall weights (measured by the Frobenius norm), due to more links in the matrices.

To give an example of some results, we computed an IMSCAND rank-20 decomposition on the nine SIAM journals. The confusion matrix follows with the clusters from *k*-means as the columns, and the actual journals in which the papers were published as rows:

	1	2	3	4	5	6	7	8	9
SIAM J Appl Math	390	114	13	0	19	0	1	0	8
SIAM J Comput	0	377	11	142	0	4	0	4	0
SIAM J Control Optim	57	311	171	1	3	5	4	16	8
SIAM J Discrete Math	0	126	2	128	0	1	1	1	0
SIAM J Math Anal	234	115	4	0	41	3	2	0	18
SIAM J Matrix Anal A	12	135	2	7	1	255	3	6	1
SIAM J Numer Anal	98	176	7	1	59	26	197	3	44
SIAM J Opitmiz	0	173	12	8	0	8	1	142	0
SIAM J Sci Comput	85	244	7	5	37	118	103	7	48

An associated 3-D bar chart is illustrated in Figure 6.2.

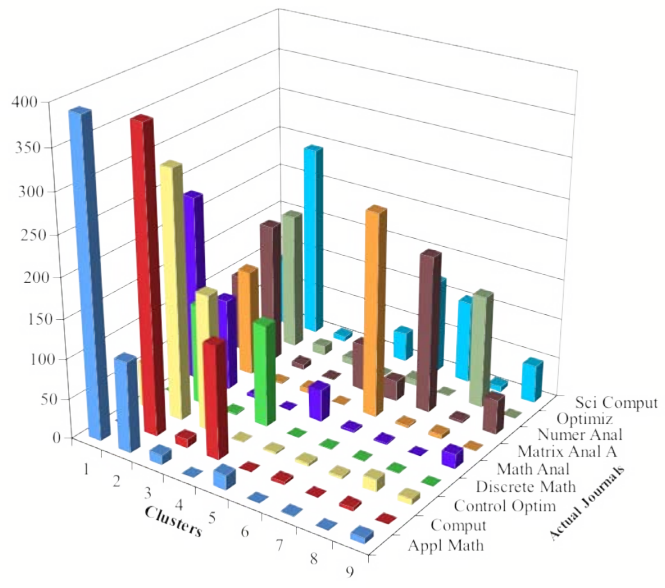


FIG. 6.2. *This chart contains the same information as the confusion matrix. The papers come from 9 SIAM journals, and we have asked k-means to find 9 clusters. The height of each column is the number of documents.*

We can see by looking at the columns of the confusion matrix that two of the clusters, #3 and #8, find papers mostly from one journal. However, if we look at these specific journals, these clusters are not getting all the papers for the journal, just a large portion of them. Looking at clusters (columns) #1, #4, #6, and #7 we are mostly getting papers from two journals. We can easily make an argument that there are documents with related authors; title, abstract, and key words; and citations in each of these different pairs of journal, and thus it is logical that we have clusters containing documents from two journals. Additionally, the SIAM J Sci Comput is a rather diverse journal, and thus occurs in many different predicted journals.

We have more than forty percent of the papers being clusters together in cluster #2. We do not have an explanation for this occurrence. It may be that the relatively sparse co-citation and co-reference slices have some impact on this. When IMSCAND and *k*-means were run only on these two slices, more than 90% of the documents were clustered together. However, IMSCAND and *k*-means on the other four slices only (abstract, author, keyword, and title) still yields a cluster containing almost 50% of the data.

7. Future Work: Clustering Attributed Relational Graphs for Information Organization (CARGIO). CARGIO is a multi-year project whose goal is to determine clusters from a set of files on a computer hard drive. Images of several hard drives are currently being examined in the groundtruthing process, to determine how the files are clustered. This groundtruth test data will be used to help identify the correct procedure for future hard drives where the clustering is unknown. For more information, see Appendix A.

8. Conclusions. We addressed the problem of clustering objects from datasets with multiple similarity measures. Our approach includes the creation of a new class of tensors, sp3way tensors, which allow the dense similarity matrices to be stored implicitly as the product of a sparse object-feature matrix and its transpose. In addition, we have created a new tensor decomposition, IMSCAND, which is identical in output to CP, but is computed on the sparse object-feature matrices only. This new decomposition allows us to handle much larger problems than if we stored the full similarity matrices. We have computed IMSCAND on both generated and real data and tried to form some conclusions on the optimal ranks of IMSCAND with regard to identifying the best number of clusters to represent a group of data.

REFERENCES

[1] B. W. Bader and T. G. Kolda, *Algorithm 862: MATLAB tensor classes for fast algorithm prototyping*, ACM Transactions on Mathematical Software, 32 (2006).

[2] R. Bro, *Multi-way analysis in the food industry: Models, algorithms, and applications*, PhD thesis, University of Amsterdam, 1998.

[3] J. D. Carroll and J.-J. Chang, *Analysis of individual differences in multidimensional scaling via an N−way generalization of "Eckart-Young" decomposition*, Psychometrika, 35 (1970), pp. 283–319.

[4] L. De Lathauwer, *A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization*, SIAM Journal for Matrix Analysis and its Applications, 28 (2006), pp. 642–666.

[5] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM Journal for Matrix Analysis and its Applications, 21 (2000), pp. 1253–1278.

[6] D. M. Dunlavy, T. G. Kolda, and W. P. Kegelmeyer, *Multilinear algebra for analyzing data with multiple linkages*, Tech. Report SAND2006-2079, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2006.

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

[8] ———, *Determination and proof of minimum uniqueness conditions for PARAFAC-1*, UCLA Working Papers in Phonetics, 22 (1972), pp. 111–117.

[9] ———, *An index formulism that generalizes the capabilities of matrix notation and algebra to n-way arrays*, J. Chemometr., 15 (2001), pp. 689–714.

[10] M. A. Jaro, *Advances in record linking methodology as applied to the 1985 census of Tampa, Florida*, Journal of the American Statistical Society, 64 (1989), pp. 1183–1210.

[11] ———, *Probabilistic linkage of large public health data file*, Statistics in Medicine, 14 (1995), pp. 491–498.

[12] T. Jiang and N. D. Sidiropoulos, *Kruskal's permutation lemma and the identification of CANDECOMP/-PARAFAC and bilinear models with constant modulus constraints*, IEEE Transactions on Signal Processing, 52 (2004), pp. 2625–2636.

[13] H. A. L. Kiers, *Towards a standardized notation and terminology in multiway analysis*, J. Chemometr., 14 (2000), pp. 105–122.

[14] T. G. Kolda, *Multilinear operators for higher-order decompositions*, Tech. Report SAND2006-2081, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2006.

[15] J. B. Kruskal, *Three-way arrays: Rank and uniqueness of trilinear decompositions, with applications to arithmetic complexity and statistics*, Linear Algebra and its Applications, 18 (1977), pp. 95–138.

[16] ———, *Rank, decomposition, and uniqueness for 3-way and n-way arrays*, in Multiway Data Analysis, Elsevier Science Publishers B.V., 1989, pp. 7–18.

[17] H. W. Kuhn, *The Hungarian method for the assignment problem*, Naval Research Logistics Quarterly, 2 (1955), pp. 83–87.

[18] R. P. McDonald, *A simple comprehensive model for the analysis of covariance structures*, Brit J. Math. Stat. Psy., 33 (1980), p. 161.

[19] M. Meilă, *Comparing clusterings*, Tech. Report 418, University of Washington, Dept. of Statistics, 2002.

[20] ———, *Comparing clustering by the variation of information*, in Lecture notes in Computer Science, B. Schölkopf and M. K. Warmuth, eds., vol. 2777, Springer, 2003, pp. 173–187.
[21] ———, *Comparing clustering – an axiomatic view*, in Proceedings of the 22nd International Conference on Machine Learning, 2005, pp. 577–584.
[22] C. R. Rao and S. Mitra, *Generalized inverse of matrices and its applications*, Wiley, New York, 1971.
[23] N. D. Sidiropoulos and R. Bro, *On the uniqueness of multilinear decompositions of n-way arrays*, Journal of Chemometrics, 14 (2000), pp. 229–239.
[24] A. Smilde, R. Bro, and P. Geladi, *Multi-way analysis: Applications in the chemical sciences*, Wiley, 2004.
[25] A. Stegeman, J. M. F. ten Berge, and L. D. Lathauwer, *Sufficient conditions for uniqueness in CANDECOMP/PARAFAC and INDSCAL with random component matrices*, Psychometrika, 71 (2006), pp. 219–229.
[26] J. M. F. ten Berge and N. D. Sidiropoulos, *On uniqueness in CANDECOMP/PARAFAC*, Psychometrika, 67 (2002), pp. 399–409.
[27] S. van Dongen, *Performance criteria for graph clustering and Markov cluster experiments*, Tech. Report INS-R0012, Center for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands, 2000.
[28] W. E. Winkler, *The state of record linkage and current research problems*, Internal Revenue Service Publication R99/04, 1999.

Appendix A. Clustering Attributed Relational Graphs for Information Organization (CARGIO). We are currently considering seven different attributes of the files, each of which produces an adjacency matrix which becomes one slice of a tensor. Three of the edges (*parent, ancestry*, and *symbolic link*) are directed, and the other four (*time delta, sibling, name match*, and *text match*) are undirected.

- *time delta:* This data yields a completely dense adjacency matrix. Specifically, for any set of two nodes, the difference in seconds between time stamps is used to compute the weight. If d_{ij} is the difference in seconds, then the weight is $1/(1 + d_{ij})$.
- *parent:* This is binary data. If file i is contained within folder j, then the (i, j) entry of the adjacency matrix is 1. Otherwise, it has value 0.
- *sibling:* Files i and j are siblings if they are contained inside the same directory. The integer edge weight between them gives their depth in the directory.
- *ancestry:* Perhaps the easiest way to think of this is that the ancestry distance between 2 points is how many of the Unix "cd .." commands need to be executed to move from node i to node j. If it is not possible to move from i to j in this way, then the distance is ∞. The weight, w_{ij} is computed as $1/(1 + d_{ij})$.
- *name match:* This is a measurement of how much two filenames match. Higher weights are given to files whose prefix letters are more similar than those whose suffix letters are similar. The weighting is done using the Jaro-Winkler distance metric [28] which is a measure of similarity between two strings. This is a variant of the Jaro distance metric [10, 11].
- *text match:* This measures how well the text matches within two files for normal ASCII text files only. For each file, a word bag is created, then compared to other word bags. The similarity measurement being used is the Tanimoto coefficient (Extended Jaccard Coefficient — see `http://en.wikipedia.org/wiki/Jaccard\_index`).
- *symbolic link:* This is currently a zero matrix, or an extremely sparse binary matrix. If i is a symbolic link to j, then the (i, j) element is 1.

A.1. Adjusting CARGIO data for use with IMSCAND. In its current form, this problem is not computationally possible for the test hard drives. Every sample hard drive has at least 4500 files. A tensor, $\mathcal{X}$ to store this data would be $\in \mathbb{R}^{4500\times4500\times7}$. The slices range from having 3 to more than 21 million nonzero elements. There are more than 40 million nonzero elements total, which becomes too much for an average computer during the CP-ALS algorithm.

We are currently working to adjust the edge types so that they may be treated by the IMSCAND decomposition. We have extended the IMSCAND decomposition for this application to include the ability to treat both sparse, symmetric similarity (object-object) matrices and sparse object-feature matrices (which are multiplied by their transpose to form a tensor slice).

It is not immediately obvious how to adjust *time delta*. One possibility is to form a sparse, binary file-time matrix, where the columns are different timeframes, e.g., 1 minute, 10 minutes, 1 hour, 10 hours, 1 day, 1 week, 1 month, 3 months, 6 months, 1 year, > 1 year. Then if file i was altered in the last 7 minutes, there would be a 1 in row i in the 10 minute column. The similarity matrix is formed as the product of the file-time matrix and its transpose.

The *parent*, *sibling*, and *ancestry* matrices are all formed from the same starting matrix. Consider the set of 8 nodes with edges illustrated in Figure A.1.

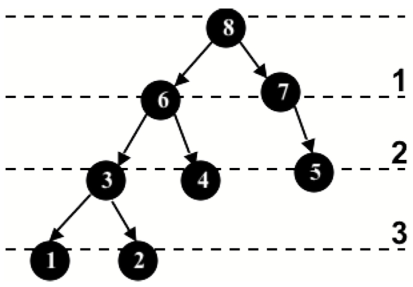


Fig. A.1. *Each node represents a computer file. Node 3 is a parent to nodes 1 and 2. Node 6 is a parent to nodes 3 and 4. Node 7 is a parent to node 5, and node 8 is a parent to nodes 6 and 7.*

We represent this graph as a matrix where the (i, j) element of the matrix is 1 if file j is the parent of file i.

$$\mathbf{P} = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \left(\begin{array}{cccccccc} & & 1 & & & & & \\ & & 1 & & & & & \\ & & & & & 1 & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ & & & & & & & 1 \\ & & & & & & & \end{array}\right) \end{array}$$

Technically, the matrix $\mathbf{P}$ is the parent matrix. However, for implementation we need symmetry, thus we could use $\mathbf{P} + \mathbf{P}^T$ to represent the parent-child relationship.

We have two options for the sibling matrix. We could compute $\mathbf{P}\mathbf{P}^T$ directly and get sibling information without weights. The other option is to compute a new matrix $\widehat{\mathbf{P}}$ whose edges are weighted corresponding to the square root of their depth in the tree, then compute $\widehat{\mathbf{P}\mathbf{P}^T}$. With $\widehat{\mathbf{P}}$ defined as:

1 2 3 4 5 6 7 8

$$\widehat{\mathbf{P}} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \left(\begin{array}{cccccccc} & & \sqrt{3} & & & & & \\ & & \sqrt{3} & & & & & \\ & & & & & \sqrt{2} & & \\ & & & & & \sqrt{2} & & \\ & & & & & & \sqrt{2} & \\ & & & & & & & 1 \\ & & & & & & & 1 \\ & & & & & & & \end{array} \right)$$

we compute two options for the *sibling* matrices:

$$\mathbf{PP}^T = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 1 & 1 & & & & & & \\ 2 & 1 & 1 & & & & & & \\ 3 & & & 1 & 1 & & & & \\ 4 & & & 1 & 1 & & & & \\ 5 & & & & & 1 & & & \\ 6 & & & & & & 1 & 1 & \\ 7 & & & & & & 1 & 1 & \\ 8 & & & & & & & & \end{array}, \quad \widehat{\mathbf{PP}^T} = \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 3 & 3 & & & & & & \\ 3 & 3 & & & & & & \\ & & 2 & 2 & & & & \\ & & 2 & 2 & & & & \\ & & & & 2 & & & \\ & & & & & 1 & 1 & \\ & & & & & 1 & 1 & \\ & & & & & & & \end{array}$$

Finally, we can get binary ancestry information by first looking at powers of the original matrix **P**. Matrix $\mathbf{P}^2$ gives grandparent information, $\mathbf{P}^3$ gives great-grandparent information, etc. For the current example, all powers of $\mathbf{P} \geq 4$ are zero.

$$\mathbf{P}^2 = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & & & & & & 1 & & \\ 2 & & & & & & 1 & & \\ 3 & & & & & & & & 1 \\ 4 & & & & & & & & 1 \\ 5 & & & & & & & & 1 \\ 6 & & & & & & & & \\ 7 & & & & & & & & \\ 8 & & & & & & & & \end{array} \quad \mathbf{P}^3 = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & & & & & & & & 1 \\ 2 & & & & & & & & 1 \\ 3 & & & & & & & & \\ 4 & & & & & & & & \\ 5 & & & & & & & & \\ 6 & & & & & & & & \\ 7 & & & & & & & & \\ 8 & & & & & & & & \end{array}$$

With these powers of the matrix **P**, we describe how to form the matrix **A**, which has a sparse structure and can be multiplied by its transpose to form the full *ancestry* similarity matrix. We must first decide how many levels of ancestry are important. A value of 8 may be reasonable, since that would give ancestry information for up to eight levels of files (or up to 7 subfolders). For this example, however, we can choose 3 since all larger powers produce a zero matrix. Then, define the *ancestry* matrix as:

$$\mathbf{A} = \sum_{k=0}^{3} \frac{\mathbf{P}^k}{k!}.$$

Notice this is an approximation of the matrix exponential, $e^A = \sum_{k=0}^{\infty} \mathbf{P}^k/k!$. The matrix **A** is our ancestry object-feature matrix, and can be multiplied by its transpose to form a dense ancestry similarity matrix.

Both *name match* and *text match* are computed similarly. For each of these we compute a different list of appropriate terms that are used as the columns of a file-term matrix. This produced a binary matrix in which a value of 1 indicates that the word associated with that column is found in the name (or the text inside) of the file.

The final slice, *symbolic link* is an extremely sparse binary matrix. We create its slice by adding the binary matrix to its transpose in order to obtain a symmetric matrix.

A.2. Numerical Results. We have analyzed both real and simulated data. Our main goal is to gain intuition for the choice of the CP rank and to increase our understanding of the k-means clustering algorithm. We believe our first attempt at generating data was too simple. We have not yet determined the effectiveness of our second code at generating useful data. Both of these datasets were only run using CP.

A.2.1. Performance Metrics. For real or simulated data when we know the true clustering, we can compare the truth to our k-means clustering output. The four metrics investigated are the variation of information (VIC), the Scaled Coverage measure (Dongen), classification error (CE), and Mirkin's metric (Mirkin). While discussing these clusters, we will use the notation C and C' to denote two different clusters. As previously defined N is the number of nodes and K is the number of clusters. We now also define n_k to be the number of nodes in cluster k (we also have k', K' and $n_{k'}$ for the cluster C'). Finally, we define $n_{kk'}$ to be the number of nodes in both cluster k in C and cluster k' in C'.

The variation of information [19, 20, 21] measures the amount of information gained by moving from C to C' and adds that to the information lost by moving from C' to C. Specifically, this function is composed of entropy functions for each of the two clusters, $H(C)$ and $H(C')$, and a function of both clusters which gives their mutual information, $I(C, C')$. We compute the VIC metric as

$$d_{VIC} = H(C) + H(C') - 2I(C, C')$$

where

$$H(C) = -\sum_{k=1}^{K} \frac{n_k}{N} \log \frac{n_k}{N} \qquad \text{and} \qquad I(C, C') = \sum_{k=1}^{K} \sum_{k'=1}^{K'} \frac{n_{kk'}}{N} \log \frac{N \cdot n_{kk'}}{n_k \cdot n_{k'}}.$$

The Scaled Coverage measure is also called the van Dongen metric [27]. This metric gives the number of node substitutions needed to convert clustering C to clustering C'. Then, per a suggestion from Meilă in [21], the metric is normalized by dividing by $2N$. Specifically, the normalized metric is computed:

$$d_{dongen} = 1 - \frac{1}{2N} \sum_{k=1}^{K} \max_{k'} n_{kk'} - \frac{1}{2N} \sum_{k'=1}^{K'} \max_{k} n_{kk'}.$$

Classification error (CE)[2] is based on the Kuhn-Munkres (Hungarian) algorithm [17, 21]. It is similar to van Dongen's metric, however it is looking for a unique mapping, whereas van Dongen's approach allows multiple clusters to be mapped to one cluster. The metric is defined by

$$d_{CE}(C, C') = 1 - \frac{1}{N} \max_{\sigma} \sum_{k=1}^{K} n_{k,\sigma(k)}.$$

It is assumed that $K < K'$ and we define σ to be an injective map from $\{1, \ldots, K\}$ to $\{1, \ldots, K'\}$. Essentially, CE looks at all possible one-to-one comparisons of clusters in C to clusters in C' and chooses the assignment that minimizes the distance between the two clusterings.

[2]Although this algorithm seems to be a good measure of the clustering accuracy, it can take an extremely long time to compute when we are comparing two clustering schemes that are not very similar. For the time being, we are not computing this metric.

The final metric we consider is Mirkin's metric. This metric gives twice the number of points that are in disagreement between any two clusterings C and C'. Another thought is that it gives twice the number of point pairs that are in the same cluster in C, but in different clusters in C', or vice versa. This metric is rescaled by dividing by N^2. We compute this as

$$d_{mirkin}(C, C') = \frac{1}{N^2}\left(\sum_k n_k^2 + \sum_{k'} n_{k'}^2 - 2\sum_k \sum_{k'} n_{kk'}^2\right).$$

A.2.2. Our initial set of generalized data. We are generating data with a predetermined clustering. Our goal is to run CP and compute several performance metrics to determine which CP ranks are best at identifying the correct number of clusters. To run the code we must input the number of nodes, N (= number of rows = number of columns), the number of tensor slices, P, and the number of clusters, K.

Specifically, we generate a vector of N elements that gives us the clustering labels. We then take the first value in *pcvec* (which we will call *pcvec*(1)) and the first value in *pwvec* (called *pwvec*(1)) and create the adjacency matrix that will be the first slice of our tensor. For any two points, if they are in the same cluster, the probability that there is a link between them is in the range $[pcvec(1), 1]$. If there is a link we can either choose for it to have a random weight in that same range, or we can decide to create a binary adjacency matrix and store a weight of 1. For any two points in different clusters, the probability of a link between them is in the range $[0, pwvec(1)]$. Again, this can either have a random link with the weight falling in the specified range, or a weight of 1. After we have completed this adjacency matrix, we move on to the second elements in *pcvec* and *pwvec* to compute the second slice, and continue until we have all the matrices to produce our tensor.

With the tensor produced, we begin the process of computing CP of various ranks and the corresponding clusters and performance metrics. We initialize each CP using the eigenvectors of $\mathbf{X}_{(2)}\mathbf{X}_{(2)}^T$, where $\mathbf{X}_{(2)}$ is the mode-2 matricization of $\mathcal{X}$. For each CP decomposition computed, we call the (built-in with the Statistics package) k-means algorithm to run on the rows of the output matrix $\mathbf{A}$. One option we choose in k-means clustering is to run the algorithm multiple times and choose the clustering with the smallest normwise error. This should reduce the odds of k-means settling on a false minimum. In addition, if it naturally occurs that a cluster is lost in the k-means process, we allow the algorithm to simply continue with one less cluster.

Initially, we plotted multiple outputs, in addition to the metrics, in hopes of seeing some strong indicator for the "best" choice for the rank of CP. In addition to the four performance metrics, we also considered fit, or normwise distance between the original tensor and the Kruskal tensor, from the CP algorithm, the number of iterations it took CP to converge, and a measure from k-means of the distance of all points to their centroid for each cluster After several trials, we decided to not compute the CE performance metric for each iteration, since it was taking an extremely long time, especially when the clustering from k-means was not very close to the true clustering. We also realized that fit and iteration from CP did not indicate that any rank of CP was better than any other. However, the distance measure of points to their centroid from k-means turned out to be quite helpful when divided by the rank of CP. The value of the CP rank is the number of dimension, so dividing the sum by this value is essentially a normalization of the sum over the dimensions.

The included results correspond to a $1000 \times 1000 \times 5$ tensors with approximately 300,000 nonzero entries total and ten true clusters. Simulations were run, asking k-means for 7, 10, and 14 clusters. I computed CP ranks for 1 to 30, computing k-means clustering at each rank.

For all three values of k-means, the performance metrics did not indicate that any rank for CP is better than any other. In general, Mirkin's metric was very low (less than 0.36 for

7 *k*-means, and less than 0.16 for the other two *k*-means), which indicates that the *k*-means clusters were good approximations to the true clusters according to that metric. Dongen's metric was also quite low, never ranging above 0.33. The VIC metric can grow quite large when looking at two very different clusters. We did get some values to be quite low (less than 0.2), and it achieved a maximum value at 1.46. The CE took an extremely long time to run, and in our limited experience, we did not find it to be a good indicator of the best CP rank, so we have omitted computing it.

We do have 2 decent indicators of the true number of clusters. These both occur using an output from *k*-means that gives the sum of the distances from each point to its centroid. We consider the sum directly as well as a normalization in which we divide each sum by the number of dimensions (rank of CP). As long as we ask *k*-means to find a number of clusters greater than or equal to the true number, this jump occurs. When *k*-means is asked to find less than the true number of clusters, the jump occurs at the number of clusters asked for by *k*-means.

The major reason we believe this to be too simplistic is that similar results can be generated from just one slice of the tensor.

A.2.3. Our second set of generalized data. One of the major goals with this data was that we wanted to ensure that all (or at least more than one) of the slices were needed to obtain all of the information. We generate data with a predetermined clustering and create the adjacency matrices the same way as the original code. The difference is that we can determine, for each slice (or adjacency matrix), how many of the clusters to ignore. When we ignore a cluster, all of its nodes are treated with the out-of-cluster probabilities of occurring. This code also allows us to restrict the number of nonzero elements per slice.

In the trials we ran using this dataset, we were not able to identify an optimal CP rank. Both normalized and unnormalized sums of distances of *k*-means points to their centroids could indicate the number of clusters requested from *k*-means. In addition, the performance metrics were better once we were computing a CP rank greater than the number of clusters requested from *k*-means. These results coincided with the results from the sp3way trials.