# Tensor Spectral Clustering for Partitioning Higher-order Network Structures

Austin R. Benson[*] David F. Gleich[†] Jure Leskovec[‡]

**Abstract**

Spectral graph theory-based methods represent an important class of tools for studying the structure of networks. Spectral methods are based on a first-order Markov chain derived from a random walk on the graph and thus they cannot take advantage of important higher-order network substructures such as triangles, cycles, and feed-forward loops. Here we propose a *Tensor Spectral Clustering* (TSC) algorithm that allows for modeling higher-order network structures in a graph partitioning framework. Our TSC algorithm allows the user to specify which higher-order network structures (cycles, feed-forward loops, etc.) should be preserved by the network clustering. Higher-order network structures of interest are represented using a tensor, which we then partition by developing a multilinear spectral method. Our framework can be applied to discovering layered flows in networks as well as graph anomaly detection, which we illustrate on synthetic networks. In directed networks, a higher-order structure of particular interest is the directed 3-cycle, which captures feedback loops in networks. We demonstrate that our TSC algorithm produces large partitions that cut fewer directed 3-cycles than standard spectral clustering algorithms.

## 1 Introduction

Spectral graph methods investigate the structure of networks by studying the eigenvalues and eigenvectors of matrices associated to the graph, such as its adjacency matrix or Laplacian matrix. Arguably the most important spectral graph algorithms are the spectral graph partitioning methods that identify partitions of nodes into low conductance communities in undirected networks [1]. While the simple matrix computations and strong mathematical theory behind spectral clustering methods makes them appealing, the methods are inherently limited to *two-dimensional* structures, for example, undirected edges connecting *pairs* nodes. Thus, it is a natural question whether spectral methods can be generalized to higher-order network structures. For example, traditional spectral clustering attempts to minimize (appropriately normalized) number of first-order structures (*i.e.*, edges) that need to be cut in order to split the graph into two parts. In a similar spirit, a higher-order generalization of spectral clustering would try to minimize cutting *higher-order structures* that involve multiple nodes (*e.g.*, triangles).

Incorporating higher-order graph information (that is, network motifs/graphlets) into the partitioning process can significantly improve our understanding of the underlying network. For example, triangles (three-dimensional network structures involving *three* nodes) have proven fundamental to understanding social networks [14, 21] and their community structure [10, 26, 29]. Most importantly, higher-order spectral clustering would allow for greater modeling flexibility as the application would drive which higher-order network structures should be preserved by the network clustering. For example, in financial networks, directed cycles might indicate money laundering and higher-order spectral clustering could be used to identify groups of nodes that participate in such directed cycles. As directed cycles involve multiple edges, current spectral clustering tools would not be able to identify groups with such structural signatures.

Generalizing spectral clustering to higher-order structures involves several challenges. The essential challenge is that higher-order structures are often encoded in tensors, i.e., multi-dimensional matrices. Even simple computations with tensors lack the traditional algorithmic guarantees of two-dimensional matrix computations such as existence and known runtimes. For instance, eigenvectors are a key component to spectral clustering, and finding tensor eigenvectors is NP-hard [15]. An additional challenge is that the number of higher-order structures increases exponentially with the size of the structure. For example, in a graph with n nodes, the number of possible triangles is $O(n^3)$. However, real-world networks have far fewer triangles.

While there exist several extensions to the spectral method, including the directed Laplacian [5], the asymmetric Laplacian [4], and co-clustering [9, 28], these methods are all limited to two-dimensional graph representations. A simple work-around would be to weight edges that occur in higher-order structures [19]. However, this heuristic is unsatisfactory because the optimization is still on edges, and not on the higher-order patterns we aim to cluster.

Here, we propose a *Tensor Spectral Clustering (*TSC*)* framework that is directly based on higher-order network structures, *i.e.*, network information beyond edges connect-

[*]Institute for Computational and Mathematical Engineering, Stanford University. Supported by Stanford Graduate Fellowship.

[†]Department of Computer Science, Purdue University. Supported by NSF CAREER CCF-1149756 and IIS-1422918.

[‡]Department of Computer Science, Stanford University. Supported by NSF IIS-1016909, CNS-1010921, IIS-1149837, ARO MURI, DARPA GRAPHS, PayPal, Docomo, Volkswagen, and Yahoo.

ing two nodes. Our framework operates on a tensor of network data and allows the user to specify which higher-order network structures (cycles, feed-forward loops, etc.) should be preserved by the clustering. For example, if one aims to obtain a partitioning that does not cut triangles, then this can be encoded in a third-order tensor $\underline{\boldsymbol{T}}$, where $\underline{\boldsymbol{T}}(i, j, k)$ is equal to 1 if nodes $i$, $j$, and $k$ form a triangle and 0 otherwise.

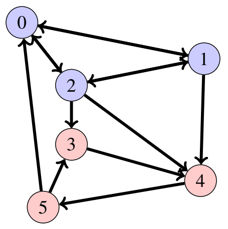
Given a tensor representation of the desired higher-order network structures, we then use a mutlilinear PageRank vector [13] to reduce the tensor to a two-dimensional matrix. This dimensionality reduction step allows us to use efficient matrix algorithms while approximately preserving the higher-order structures represented by the tensor. Our resulting TSC algorithm is a spectral method that partitions the network to minimize the number of higher-order structures cut. This way our algorithm finds subgraphs that contain many instances of the higher-order structure described by the tensor. Figure 1 illustrates a directed network, and our goal is to identify clusters of directed 3-cycles. That is, we aim to partition the nodes into two sets such that few directed 3-cycles get cut. Our TSC algorithm finds a partition that does not cut any of the directed 3-cycles, while a standard spectral partitioner (the directed Laplacian [5]) does.

Clustering networks based on higher-order structures has many applications. For example, the TSC algorithm allows for identifying layered flows in networks, where the network consists of several layers that contain many feedback loops. Between layers, there are many edges, but they flow in one direction and do not contribute to feedback. We identify such layers by clustering a tensor that describes small feedback loops (*e.g.*, directed 3-cycles and reciprocated edges). Similarly, TSC can be applied to anomaly detection in directed networks, where the tensor encodes directed 3-cycles that have no reciprocated edges. Our TSC algorithm can find subgraphs that have many instances of this pattern, while other spectral methods fail to capture these higher-order network structures.

Our contributions are summarized as follows:

- In Sec. 3, we develop a tensor spectral clustering framework that computes directly on higher-order graph structures. We provide theoretical justifications for our framework in Sec. 4.
- In Sec. 5, we provide two applications—layered flow networks and anomaly detection—where our tensor spectral clustering algorithm outperforms standard spectral clustering on small, illustrative networks.
- In Sec. 6, we use tensor spectral clustering to partition large networks so that directed 3-cycles are not cut. This provides additional empirical evidence that our algorithm out-performs state-of-the-art spectral methods.

Code used for this paper is available at https://github.com/arbenson/tensor-sc, and all networks used in experiments are available from SNAP [23].



Tensor spectral clustering:
{0, 1, 2}, {3, 4, 5}

Directed Laplacian:
{1, 2, 5}, {0, 3, 4}

Figure 1: (Left) Network where directed 3-cycles only appear within the blue or red nodes. (Right) Partitioning found by our proposed tensor spectral clustering algorithm and the directed Laplacian. Our proposed algorithm doesn't cut any directed 3-cycles. Directed 3-cycles are just one higher-order structure that can be used within our framework.

## 2 Preliminaries and background

We now review spectral clustering and conductance cut. The key ideas are a Markov chain representing a random walk on a graphs, a second left eigenvector of the Markov chain, and a sweep cut that uses the ordering of the eigenvector to compute conductance scores. In Sec. 3, we generalize these ideas to tensors and higher-order structures on graphs.

**2.1 Notation and the transition matrix** Consider an undirected, weighted graph $G = (V, E)$, where $n = |V|$ and $m = |E|$. Let $\boldsymbol{A} \in \mathbb{R}_{+}^{n \times n}$ be the weighted adjacency matrix of $G$, *i.e.*, $A_{ij} = w_{ij}$ if $(i, j) \in E$ and $A_{ij} = 0$ otherwise. Let $\boldsymbol{D}$ be the diagonal matrix with generalized degrees of the vertices of $G$. In other words, $\boldsymbol{D} = \operatorname{diag}(\boldsymbol{A}\boldsymbol{e})$, where $\boldsymbol{e}$ is the vector of all ones. The *combinatorial Laplacian* or *Kirchoff* matrix is $\boldsymbol{K} = \boldsymbol{D} - \boldsymbol{A}$. The matrix $\boldsymbol{P} = \boldsymbol{A}^{\top}\boldsymbol{D}^{-1}$ is a column stochastic matrix, which we call the *transition matrix*. We now interpret this matrix as a Markov chain.

**2.2 Markov chain interpretation** Since $\boldsymbol{P}$ is column stochastic, we can interpret the matrix as a Markov chain with states $S_t$, for each time step $t$. Specifically, the states of the Markov chain are the vertices on the graph, *i.e.*, $S_t \in V$. The transition probabilities are given by $\boldsymbol{P}$:

$$\text{Prob}(S_{t+1} = i \mid S_t = j) = \boldsymbol{P}_{ij} = A_{ji}/D_{jj}.$$

This Markov chain represents a *random walk* on the graph $G$. In Sec. 3.2, we will generalize this idea to tensors of graph data. We now show how the second left eigenvector of the Markov chain described here is key to spectral clustering.

**2.3 Second left eigenvector for conductance cut** The *conductance* of a set $S \subset V$ of nodes is

$$\phi(S) = \text{cut}(S) \,/\, \min\left(\text{vol}(S), \text{vol}(\bar{S})\right), \tag{2.1}$$

where $\text{cut}(S) = \left|\{(u, v) \mid u \in S, v \in \bar{S}\}\right|$, and $\text{vol}(S) = |\{(u, v) \mid u \in S\}|$. Small conductance indicates a good partition of the graph: the number of cut edges must be small and neither $S$ nor $\bar{S}$ can be too small. Let $z \in \{-1, 1\}^n$ be an indicator vector over the nodes in $G$, where $z_i = 1$ if the $i$th node is in $S$. Then

$$z^{\mathsf{T}} \boldsymbol{K} z = \sum_{(i,j)\in E} 4\mathbb{I}\left(z_i = z_j\right) \propto \operatorname{cut}(S)\,. \tag{2.2}$$

The conductance cut eigenvalue problem is an approximation for the NP-hard problem of minimizing conductance:

$$\begin{array}{ll} \underset{z\in\mathbb{R}^n}{\text{minimize}} & z^{\mathsf{T}}\boldsymbol{K}z / z^{\mathsf{T}}\boldsymbol{D}z \\ \text{subject to} & \boldsymbol{e}^{\mathsf{T}}\boldsymbol{D}z = 0, \quad \|z\| = 1 \end{array} \tag{2.3}$$

The idea of the real-valued relaxation in Eqn. (2.3) is that positive and negative values of $z$ correspond to the $\pm 1$ indicator vector for the cut in Eqn. (2.2). In Sec. 2.4 we will review how to convert the real-valued solution to a cut.

The matrices $\boldsymbol{K}$ and $\boldsymbol{D}$ are positive semi-definite, and Eqn. (2.3) is a generalized eigenvalue problem. In particular, the solution is the vector $z$ such that $\boldsymbol{K}z = \lambda\boldsymbol{D}z$, where $\lambda$ is the second smallest generalized eigenvalue (the smallest eigenvalue is 0 and corresponds to the trivial solution $z = \boldsymbol{e}$). To get the solution $z$, we observe that

$$\begin{aligned} \boldsymbol{K}z = \lambda\boldsymbol{D}z &\iff (\boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A})z = \lambda z \\ &\iff z^T\boldsymbol{P} = (1-\lambda)z^T \end{aligned}$$

where $1 - \lambda$ is the second largest left eigenvalue of $\boldsymbol{P}$. We know that $\boldsymbol{e}^T\boldsymbol{P} = \boldsymbol{e}^T$, so we are looking for the dominant left eigenvector that is orthogonal to the trivial one.

Here, we call the above partitioning algorithm for undirected graphs the "undirected Laplacian" method. One generalization to directed graphs is due to Chung [5]. For this method, we use the undirected Laplacian method on the following symmetrized network: $\boldsymbol{A}_{sym} := \frac{1}{2}\left(\boldsymbol{\Pi}\boldsymbol{P}^{\mathsf{T}} + \boldsymbol{P}\boldsymbol{\Pi}\right)$, where $\boldsymbol{P} = \boldsymbol{A}^{\mathsf{T}}\boldsymbol{D}^{-1}$ and $\boldsymbol{\Pi} = \operatorname{diag}(\pi)$ for $\boldsymbol{P}\pi = \pi$, the stationary distribution of $\boldsymbol{P}$. Note that $\boldsymbol{D}_{sym} = \operatorname{diag}\left(\boldsymbol{A}_{sym}\boldsymbol{e}\right) = \boldsymbol{\Pi}$, so we are interested in the second left eigenvector of

$$\boldsymbol{P}_{sym} = \frac{1}{2}\left(\boldsymbol{\Pi}\boldsymbol{P}^{\mathsf{T}}\boldsymbol{\Pi}^{-1} + \boldsymbol{P}\right). \tag{2.4}$$

By "directed Laplacian", we refer to the method that uses the second left eigenvector of $\boldsymbol{P}_{sym}$.

**2.4 Sweep cuts** In order to round the real-valued solution $z$ to a solution set $S$ to evaluate Eqn. (2.1), we sort the vertices by the values $z_i$ and consider vertex sets $S_k$ that consist of the first $k$ nodes in the sorted vertex list. In other words, if $\sigma_i$ is equal to the index of the $i$th smallest element of $z$, then $S_k = \{\sigma_1, \sigma_2, \ldots \sigma_k\}$. We then choose $S = \arg\min_{S_k} \phi(S_k)$. The set of nodes $S$ satisfies the celebrated Cheeger inequality [1]: $\phi_*^2/2 \le \phi(S) \le 2\phi_*$, where $\phi_*$ is the minimum conductance over all cuts. The sweep cut computation is fast, since $S_{k+1}$ differs from $S_k$ by only one node, and the sequence of scores $\phi(S_1), \ldots, \phi(S_n)$ can be computed in $O(n+m)$ time.

In addition to conductance, other scores can also be computed in the same sweeping fashion. Of particular interest are the normalized cut, $ncut(S) = \operatorname{cut}(S)\left(1/\operatorname{vol}(S) + 1/\operatorname{vol}(\bar{S})\right)$, and the expansion, $\rho(S) = \operatorname{cut}(S) / \min(|S|, |\bar{S}|)$. The normalized cut differs by at most a factor of two from conductance, so we will limit ourselves to conductance and expansion in this paper.

## 3 Tensor spectral clustering framework

The key ingredients for spectral clustering discussed in Sec. 2 were a transition matrix from an undirected graph, a Markov chain interpretation of the transition matrix, and the second left eigenvector of the Markov chain. We now generalize these ideas for higher-order network structures.

**3.1 Transition tensors** Our first goal is to represent the higher-order network stuctures of interest. For example, to represent structures on three nodes (*i.e.*, directed cycles, or feed-forward loops) we required a three-dimensional tensor. In particular, we want a *symmetric* order-3 tensor $\underline{\boldsymbol{T}} \in \mathbb{R}_+^{n\times n\times n}$ such that the entry at index $(i, j, k)$ contains information about nodes $i, j, k \in V$. (Here, symmetric means that the value of $\underline{\boldsymbol{T}}(i, j, k)$ remains the same under any permutation of the three indices.) A tensor describing triangles in $G$ is:

$$\underline{\boldsymbol{T}}(i, j, k) = \mathbb{I}\left(i, j, k \in V \text{ distinct and form a triangle}\right). \tag{3.5}$$

This tensor represents third-order information about the graph. We form a transition tensor by

$$\underline{\boldsymbol{P}}(i, j, k) = \underline{\boldsymbol{T}}(i, j, k) / \sum_{i=1}^{n} \underline{\boldsymbol{T}}(i, j, k), \quad 1 \le i, j, k \le n.$$

In the case that $\sum_{i=1}^{n} \underline{\boldsymbol{T}}(i, j, k) = 0$, we fill in $\underline{\boldsymbol{P}}(:, j, k)$ with a stochastic vector $\boldsymbol{u}$, *i.e.*, $\underline{\boldsymbol{P}}(:, j, k) = \boldsymbol{u}$. We call the vector $\boldsymbol{u}$ the *dangling distribution vector*, borrowing the term from the PageRank community [3]. Next, we see how to interpret this transition tensor as a second-order Markov chain.

**3.2 Second-order Markov chains and the spacey random surfer** Next, we seek to generalize the Markov chain interpretation of spectral clustering to tensors. While spectral clustering on matrices is analogous to a first-order Markov chain, we will show that tensor spectral clustering is analogous to a second-order Markov chain on a matrix representation of the tensor.

Entries of the transition tensor $\underline{\boldsymbol{P}}$ from Sec. 3.1 can be interpreted as the transition probabilities of a second-order Markov chain. Specifically, given a second-order Markov chain with state space the set of vertices, $V$, we define the transition probabilities as

$$\underline{\boldsymbol{P}}(i,j,k) = \text{Prob}\,(S_{t+1} = i \mid S_t = j, S_{t-1} = k)\,.$$

In other words, the probability of moving to state $i$ depends on the current state $j$ and the last state $k$. For the triangle tensor in Eqn. (3.5),

$$\underline{\boldsymbol{P}}(i,j,k) = \frac{\mathbb{I}\,(i,j,k \text{ form triangle})}{\#(\text{triangles involving nodes } j \text{ and } k)}$$

If the previous state was node $k$ and the current state is node $j$, then, for the next state, the Markov chain chooses uniformly over all nodes $i$ that form a triangle with $j$ and $k$.

The stationary distribution $X_{ij}$ of the second-order Markov chain satisfies $\sum_k \underline{\boldsymbol{P}}(i,j,k)X_{jk} = X_{ij}$. We would like to model the full second-order dynamics of the Markov chain, but doing so is computationally infeasible because just storing the stationary distribution requires $O(n^2)$ memory. Instead, we will make the simplifying assumption that $X_{ij} = x_i x_j$ for some vector $\boldsymbol{x} \in \mathbb{R}^n_+$ with $\sum_i x_i = 1$. The stationary distribution then satisfies

$$\text{(3.6)} \qquad \sum_{1\le j,k\le n} \underline{\boldsymbol{P}}(i,j,k)x_j x_k = x_i.$$

With respect to Eqn. (3.6), $\boldsymbol{x}$ is called a $z$ eigenvector of the tensor $\underline{\boldsymbol{P}}$ with eigenvalue 1 [27]. To simplify notation, we will denote the one-mode unfolding of $\underline{\boldsymbol{P}}$ by $\boldsymbol{R} \in \mathbb{R}^{n\times n^2}$, namely $\boldsymbol{R} = \begin{bmatrix}\underline{\boldsymbol{P}}(:,:,1) & \underline{\boldsymbol{P}}(:,:,2) & \dots & \underline{\boldsymbol{P}}(:,:,n)\end{bmatrix}$. The matrix $\boldsymbol{R}$ is a column stochastic matrix. We use $\boldsymbol{R}_k = \underline{\boldsymbol{P}}(:,:,k)$ to denote the $k$th $n\times n$ block of $\boldsymbol{R}$. With this notation, Eqn. (3.6) reduces to $\boldsymbol{R}\cdot(\boldsymbol{x}\otimes\boldsymbol{x}) = \boldsymbol{x}$, where $\otimes$ denotes the Kronecker product.

The simplifying approximation $X_{ij} = x_i x_j$ is computationally and algebraically appealing, but we also want a random process to interpret the vector. Recent work [13] has considered the *multilinear pagerank* vector $\boldsymbol{x}$ that satisfies

$$\text{(3.7)} \qquad \alpha\boldsymbol{R}\,(\boldsymbol{x}\otimes\boldsymbol{x}) + (1-\alpha)\boldsymbol{v} = \boldsymbol{x},\;\; x_k \ge 0,\;\; \boldsymbol{e}^{\mathsf{T}}\boldsymbol{x} = 1,$$

for a constant $\alpha \in (0,1)$ and stochastic vector $\boldsymbol{v}$.

This vector is the stationary distribution of a stochastic process recently termed the *spacey random surfer* [12]. At any step of the process, a random surfer has just moved from node $k$ to node $j$. With probability $(1-\alpha)$, the surfer teleports to a random state via the stochastic vector $\boldsymbol{v}$. With probability $\alpha$, the surfer wants to transition to node $i$ with probability $\underline{\boldsymbol{P}}(i,j,k)$. However, the surfer *spaces out* and forgets that s/he came from node $k$. Instead, the surfer guesses the previous state, based on the historical distribution over the state space. Formally, the surfer guesses node $\ell$ with probability $\frac{1}{t+n}\left(1 + \sum_{r=1}^{t}\mathbb{I}\,[S_t = \ell]\right)$. It is important to note that although this process is an approximation to a second-order Markov chain, the process is no longer Markovian.

### 3.3 Second left eigenvector

Following the steps of spectral clustering, we now need to obtain an equivalent of the second left eigenvector (Sec. 2.3). In particular, we now show how to get a relevant eigenvector from the multilinear PageRank vector $\boldsymbol{x}$ and the transition tensor $\boldsymbol{P}$. The multilinear PageRank vector $\boldsymbol{x}$ satisfying $\alpha\boldsymbol{R}\cdot(\boldsymbol{x}\otimes\boldsymbol{x}) + (1-\alpha)\boldsymbol{v} = \boldsymbol{x}$ can also be re-interpreted as the stationary distribution of a particular Markov chain. Specifically, define the matrix

$$\text{(3.8)} \qquad \boldsymbol{P}[\boldsymbol{x}] := \sum_{k=1}^{n} x_k \boldsymbol{R}_k.$$

(Recall that $\boldsymbol{R}_k = \underline{\boldsymbol{P}}(:,:,k)$ is the $k$th $n\times n$ block of $\boldsymbol{R}$). The matrix $\boldsymbol{P}[\boldsymbol{x}]$ is column stochastic because each $\boldsymbol{R}_k$ is column stochastic and $\sum_{k=1}^n x_k = 1$. Note that

$$\boldsymbol{R}\cdot(\boldsymbol{x}\otimes\boldsymbol{x}) = \sum_{k=1}^{n}\boldsymbol{R}_k\,(x_k\boldsymbol{x}) = \left(\sum_{k=1}^{n} x_k\boldsymbol{R}_k\right)\boldsymbol{x} = \boldsymbol{P}[\boldsymbol{x}]\cdot\boldsymbol{x}.$$

Hence, $\boldsymbol{x}$ is the stationary distribution of the PageRank system $\alpha\boldsymbol{P}[\boldsymbol{x}]\cdot\boldsymbol{x} + (1-\alpha)\boldsymbol{v} = \boldsymbol{x}$. However, the transition matrix depends on $\boldsymbol{x}$ itself.

We use the second left eigenvector of $\boldsymbol{P}[\boldsymbol{x}]$ for our higher-order spectral clustering algorithm. Heuristically, $\boldsymbol{P}[\boldsymbol{x}]$ is a weighted sum of $n$ "views" of the graph (the matrices $\boldsymbol{R}_k$), from the perspective of each node ($k$, $1\le k\le n$), according to three-dimensional graph data (the tensor $\underline{\boldsymbol{T}}$). If node $k$ has a large influence on the three-dimensional data, then $x_k$ will be large and we will weight data associated with node $k$ more heavily. The ordering of the eigenvector will be used for a sweep cut on the vertices.

### 3.4 Sweep cuts

The last remaining step is to generalize the notion of the sweep cut (Sec. 2.4). Recall that the sweep cut takes some ordering on the nodes, $\sigma$, and computes some score $f(S_k)$ for each cut $S_k = \{\sigma_1,\dots,\sigma_k\}$. Finally, the sweep cut procedure returns $\arg\max_{S_k} f(S_k)$. The eigenvector from Sec. 3.3 provides us with an ordering for a sweep cut, just as in the two-dimensional case (Sec. 2.4). We generalize the cut and volume measures as follows:

$$\begin{aligned}\text{cut}_3(S) &:= \sum_{i,j,k\in V}\underline{\boldsymbol{T}}(i,j,k) - \sum_{i,j,k\in S}\underline{\boldsymbol{T}}(i,j,k) - \sum_{i,j,k\in\bar{S}}\underline{\boldsymbol{T}}(i,j,k)\\ \text{vol}_3(S) &:= \sum\underline{\boldsymbol{T}}(S,V,V).\end{aligned}$$

And we define "higher-order conductance" (denoted $\phi_3$) and "higher-order expansion" (denoted $\rho_3$) as

$$\text{(3.9)} \qquad \phi_3(S) := \frac{\text{cut}_3(S)}{\min\left(\text{vol}_3(S), \text{vol}_3(\bar{S})\right)}$$

$$\text{(3.10)} \qquad \rho_3(S) := \frac{\text{cut}_3(S)}{\min\left(|S|, |\bar{S}|\right)}.$$

This definition ensures that $\phi_3(S) \in [0,1]$, as in standard conductance.

**Algorithm 1:** Tensor Spectral Partitioning

```
Data: $G = (V, E)$, $|V| = n$, $\underline{\boldsymbol{T}} \in \mathbb{R}_+^{n\times n\times n}$, dangling
      distribution vector $\boldsymbol{u}$, $\alpha \in (0, 1)$
Result: Set of nodes $S \subset V$
for $1 \le i, j, k \le n$, $\underline{\boldsymbol{T}}(i, j, k) \ne 0$ do
    $\underline{\boldsymbol{P}}(i, j, k) \leftarrow \underline{\boldsymbol{T}}(i, j, k) / \sum_i \underline{\boldsymbol{T}}(i, j, k)$
for $j, k$ such that $\sum_i \underline{\boldsymbol{T}}(i, j, k) = 0$ do
    $\underline{\boldsymbol{P}}(:, j, k) \leftarrow u$
$\boldsymbol{x} \leftarrow MultilinearPageRank(\alpha, \underline{\boldsymbol{P}})$
$\boldsymbol{R}_k \leftarrow \underline{\boldsymbol{P}}(:, :, k)$
$\boldsymbol{P}[\boldsymbol{x}] \leftarrow \sum_k x_k \boldsymbol{R}_k$
Compute second left eigenvector $\boldsymbol{z}$ of $\boldsymbol{P}[\boldsymbol{x}]$
$\sigma \leftarrow$ sorted ordering of $z$
$S \leftarrow SweepCut(\sigma, G)$
```

**Algorithm 2:** Tensor Spectral Clustering (TSC)

```
Data: $G = (V, E)$, $|V| = n$, $\underline{\boldsymbol{T}} \in \mathbb{R}_+^{n\times n\times n}$, dangling
      distribution vector $\boldsymbol{u}$, $\alpha \in (0, 1)$, number of
      clusters $C$
Result: Partition $\mathcal{P}$ of $V$
if $|\mathcal{P}| < C$ then
    Partition $G$ into $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$
    via Algorithm 1.
    $\mathcal{P} = \mathcal{P} \cup \{V_1, V_2\}$.
    Recurse on largest component in $\mathcal{P}$.
```

**3.5 Tensor spectral clustering framework** We now have higher-order analogs of all the spectral clustering tools from Sec. 2. The central routine of our tensor spectral clustering framework is given in Algorithm 1, which is the tensor spectral partitioning algorithm. This subroutine takes a data tensor $\underline{\boldsymbol{T}}$ of third-order information about a graph $G$ and partitions the nodes into two sets. Algorithm 2 is the clustering algorithm that performs recursive bisection in order to decompose the graph into several components. This algorithm can also be used with other partitioning algorithms [11], and we will take that approach in Sec. 5.

**3.6 Complexity** The complexity of Algorithm 2 depends on the sparsity of the data tensor $\underline{\boldsymbol{T}}$, *i.e.*the number of higher-order structures in the network. The algorithm depends on the sparsity in three ways. First, all of the higher-order structures in the network must be enumerated as an upfront cost. Second, the sparsity affects the complexity of the multilinear PageRank subroutine in Algorithm 1. Third, the number of non-zeroes in $\boldsymbol{P}[\boldsymbol{x}]$ is equal to the number of the higher-order structures. When performing recursive bisection (Algorithm 2), there is no upfront cost to enumerate the structures—we only need to determine which structures are preserved under the partition.

We argue that the upfront cost is not cumbersome. Triangle enumeration for real-world undirected networks is a well-studied problem [7, 30]. For directed graphs, we can: (1) undirect the graph, (2) use high-performance code to enumerate the triangles, and (3) stream through the triangles and only keep those that are the directed structure of interest.

Now, we consider the second and third computations. Let $T$ be the number of non-zeroes in $\underline{\boldsymbol{T}}$. There are several methods for computing the multilinear PageRank vector in Algorithm 1 [13]. We use the shifted fixed point method (akin to the symmetric higher-order power method [20]). Each iteration takes $O(T)$ time, and we found that this method converges very quickly—usually within a handful of iterations. The computation of the second left eigenvector of $\boldsymbol{P}[\boldsymbol{x}]$ dominates the running time. We use the power method to compute this eigenvector. Since $\boldsymbol{P}[\boldsymbol{x}]$ has $T$ non-zero entries, each iteration takes $O(T)$ time.

Finally, we look at the relationship between $T$ and the size of the graph. In theory, $T$ can be $O(n^3)$, but this is far from what we see in practice. For the large networks considered in Sec. 6, $T \le 6m$ (see Table 1).

To summarize, the majority of our time is spent computing the eigenvector of $\boldsymbol{P}[\boldsymbol{x}]$. Each iteration takes $O(T)$ time, and $T \le 6m$ for the algorithms we consider. Standard spectral algorithms also compute an eigenvector with the power method, but each iteration is only $O(m)$ time. Thus, we can think of our algorithms as running within an order of magnitude of standard algorithms. However, when moving beyond third-order structures, we note that $T$ can be much larger.

## 4 Generalizations and directed 3-cycle cut

Before transitioning to applications, we mention two important generalities of our framework and discuss directed 3-cycle cuts. The directed 3-cycle will play an important role for our applications in Sections 5 and 6.

**4.1 Generalizations** Our first generalization deals with data beyond three dimensions. While we have presented the algorithm with three-dimensional data, the same ideas carry through for higher-order data. The multilinear PageRank vector can still be computed, although $\alpha$ must be smaller to guarantee convergence [13]. However, in practice, we do not observe large $\alpha$ impeding convergence.

Second, our TSC algorithm is a strict generalization of traditional spectral clustering in the following sense. There is a data tensor $\underline{\boldsymbol{T}}$ such that for any multilinear PageRank vector $\boldsymbol{x}$, we compute the same eigenvector that conductance cut computes. In particular, we can always define $\underline{\boldsymbol{T}}(i, j, k) = \boldsymbol{A}_{ij}$, where $\boldsymbol{A}$ is the adjacency matrix. Then $\boldsymbol{R}_k = \boldsymbol{P}$, $1 \le k \le n$, and $\boldsymbol{P}[\boldsymbol{x}] = \sum_k x_k P = \boldsymbol{P} \sum_k x_k = \boldsymbol{P}$.

**4.2 Directed 3-cycle cuts** We now turn our attention to a particular three-dimensional representation of directed graph data: directed 3-cycles (D3Cs), *i.e.*, sets of edges $(i, j)$, $(j, k)$, and $(k, i)$ for distinct nodes $i$, $j$, and $k$. Such structures are important for community detection [19] and are natural motifs for network feedback. We will use this structure for applications in Sections 5 and 6. The data tensor we use for directed 3-cycle cuts is

$$\underline{\boldsymbol{T}}(i, j, k) = \begin{cases} 2 & i, j, k \text{ form two D3Cs} \\ 1 & i, j, k \text{ form one D3C} \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

Nodes $i$, $j$, and $k$ form two D3Cs if and only if every possible directed edge between them is present. When $\underline{\boldsymbol{T}}(i, j, k) = 1$, we do not differentiate between 0, 1, or 2 reciprocated edges. For *directed 3-cycle cut*, we want to find partitions of the graph that do not cut many D3Cs.

**4.3 Strongly connected components** We now show that TSC correctly breaks up strongly connected components when using the data tensor in Eqn. (4.11). Suppose we have an undirected graph $G = (V, E)$ with two connected components $V_1$ and $V_2$. A standard result of the spectral method for conductance cut on undirected graphs (Sec. 2.3) is that there is a second left eigenvector $\boldsymbol{z}$ of $\boldsymbol{P}$ such that $\boldsymbol{z}^{\mathsf{T}}\boldsymbol{P} = \boldsymbol{z}$, and $\text{sign}(z_i) = -\text{sign}(z_j)$ for $i \in V_1$, $j \in V_2$ [6]. This means that the ordering induced by the eigenvector correctly separates the components. A similar result holds for strongly connected components in a directed graph $G$ using the directed Laplacian.

We now present a similar result for directed 3-cycle cut. First, we observe the following: *there is no directed 3-cycle that has nodes from different strongly connected components*. Now, Lemma 4.1 shows that if we have a graph with two strongly connected components, then, under some conditions, the second left eigenvector computed by Algorithm 1 correctly partitions the two strongly connected components.

Lemma 4.1. *Consider a directed graph $G = (V, E)$ with two components $V_1$ and $V_2$ such that there are no directed 3-cycles containing a node $i \in V_1$ and $j \in V_2$. Assume that the directed 3-cycle tensor $\underline{\boldsymbol{T}}$ is given by Eqn. (4.11). Augment the corresponding transition matrices $\boldsymbol{R}_k$ with a sink node $t$ so that transition involving $j \in V_1$, $k \in V_2$ (or vice versa) jump to the sink node,* i.e., *$\underline{\boldsymbol{P}}(i, j, k) = \mathbb{I}(i = t)$. Finally, instead of using the dangling distribution vector $\boldsymbol{u}$ to fill in $\underline{\boldsymbol{P}}$, assume that when $\sum_i \underline{\boldsymbol{T}}(i, j, k) = 0$ for $j, k \in V_1$, $\underline{\boldsymbol{P}}(i, j, k) = \mathbb{I}(i \in V_1)/|V_1|$. (And the same for transitions involving $j, k \in V_2$).*

*Then $\boldsymbol{P}[\boldsymbol{x}]$ has a second left eigenvector $\boldsymbol{z}$ with eigenvalue 1 such that $\boldsymbol{z}^{\mathsf{T}}\boldsymbol{e} = 0$ and sign$(z_i) = -$sign$(z_j)$ for any $i \in V_1$, $j \in V_2$.*

*Proof.* See the full version of the paper.[1]

[1] Available from https://github.com/arbenson/tensor-sc.

## 5 Applications on synthetic networks

We now explore applications of our TSC framework. The purpose of this section is to illustrate that explicitly partitioning higher-order network data can improve partitioning and clustering on directed networks. The examples that follow are small and synthetic but illustrative. In future work, we plan to use these ideas on real data sets.

For the applications in this section, we use the following parameters for the tensor spectral clustering algorithm: $\alpha = 0.99$ for the multilinear PageRank vector, $\gamma = 0.01$ for SS-HOPM, $\boldsymbol{u} = \boldsymbol{v} = \frac{1}{n}\boldsymbol{e}$, and the higher-order conductance score function (Eqn. (3.9)).

**5.1 Layered flow networks** Our first example is a network consisting of multiple layers, where feedback loops primarily occur within a layer. Information tends to flow "downwards" from one layer to the next. In other words, most edges between two layers point in the same direction. Figure 2 gives an example of such a network with three layers, each consisting of four nodes.

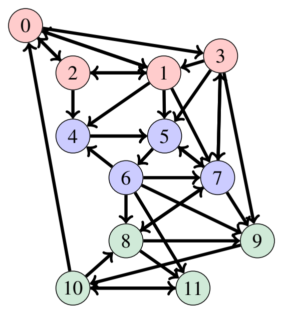
We are interested in separating the layers of the network via our TSC algorithm. Feedback in a directed network is synonymous with directed cycle. For this example, we count all directed 2-cycles (*i.e.*, reciprocated edges) and directed 3-cycles. In order to account for the directed 2-cycles, we will say that the data tensor $\underline{\boldsymbol{T}}$ is equal to one for any index of the form $(i, i, j)$, $(i, j, i)$, or $(j, i, i)$ when nodes $i$ and $j$ have reciprocated edges. Formally, the data tensor is:

$$\underline{\boldsymbol{T}}(i, j, k) = \begin{cases} 2 & i, j, k \text{ distinct and form two D3Cs} \\ 1 & i, j, k \text{ distinct and form one D3C} \\ 1 & (k = j \text{ or } k = i) \text{ and } (i, j), (j, i) \in E \\ 1 & j = i \text{ and } (i, k), (k, i) \in E \end{cases}$$

Figure 2 lists the three communities found by (1) TSC (Algorithm 2 with $C = 3$), (2) the directed Laplacian (DL), and (3) the directed Laplacian on the subgraph only including edges involved in at least one directed 2-cycle or directed 3-cycle (Sub-DL). TSC is the only method that correctly identifies the three communities. Sub-DL performs almost as well, but misclassifies node 1, placing it with the green nodes two layers beneath. In general, DL does not do well because there are a large number of edges between layers, and the algorithm does not want to cut these edges.

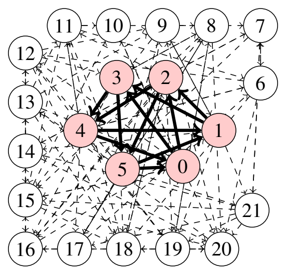
**5.2 Anomaly detection** Our second example is anomaly detection. In many real networks, most directed 3-cycles have at least one reciprocated edge [19]. Thus, a set of nodes with many directed 3-cycles and few reciprocated edges between them would be highly anamolous. The goal of this example is to show that our TSC framework can find such sets of nodes when they are planted in a network.

Figure 3 shows a network where the anomalous cluster we want to identify is nodes 0–5. All triangles between



| | |
|---|---|
| TSC | {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} |
| DL | {1, 2, 3, 6, 7, 10}, {0, 4, 5, 8, 9}, {11} |
| Sub-DL | {8, 10, 11}, {9}, {0, 1, 2, 3, 4, 5, 6, 7} |

Figure 2: (Left) Layered flow network, where almost all feedback occurs at three different layers (specified by the blue, red, and green nodes). There are many edges going from one layer to the layers below it. (Right) Three communities found when using TSC, the directed Laplacian (DL), and the directed Laplacian on the subgraph of edges participating in at least one directed 2- or 3-cycle (Sub-DL). Only TSC correctly identifies all three communities.



| | |
|---|---|
| TSC | {0, 1, 2, 3, 4, 5, 12, 13, 16} |
| DL | {1, 4, 5, 7, 8, 12, 13, 15, 18, 20} |
| Sub-DL | {0, 1, 4, 5, 9, 11 16, 17, 19, 20} |

Figure 3: (Left) Network with planted anomalous cluster (nodes 0–5). Between these nodes, there are many directed 3-cycles with no reciprocated edges (thick black lines). Nodes 6–22 follow an Erdős-Rényi graph pattern with edges indicated by dashed lines. (Right) Smaller of two communities found by TSC, the directed Laplacian (DL), and the directed Laplacian on the subgraph with only edges involved in a directed 3-cycle with no reciprocated edges (Sub-DL). Only TSC finds the entire anomalous cluster.

nodes 0–5 are directed 3-cycles with no reciprocated edges. Nodes 6–21 connect to each other according to a Erdős-Rényi model with edge probability 0.25. Finally, nodes 0–5 each have four outgoing and two incoming edges with nodes 6–21. In total, there are 18 directed 3-cycles with no reciprocated edges, and 8 of them occur between nodes 0–5.

To use the TSC framework, we form a data tensor that only counts directed 3-cycles with no reciprocated edges:

$$\begin{aligned}\underline{\boldsymbol{T}}(i,j,k) &= \mathbb{I}\left((i,j),(j,k),(k,i)\in E,(j,i),(k,j),(i,k)\notin E\right)\\ &+ \mathbb{I}\left((j,i),(k,j),(i,k)\in E,(i,j),(j,k),(k,i)\notin E\right)\end{aligned}$$

Figure 3 lists the smaller of the two communities found by (1) TSC (Algorithm 2 with $C = 2$), (2) the directed Laplacian (DL), and (3) the directed Laplacian on the subgraph only including edges involved in at least one directed 3-cycle with no reciprocated edges (Sub-DL). We see that only TSC correctly captures the planted anomalous community. DL does not capture any information about directed 3-cycles with no reciprocated edges, and hence the cut does not make sense in this context. Sub-DL correctly captures nodes 0, 1, 4, and 5, but misses nodes 2 and 3.

## 6 Directed 3-cycle cuts on large networks

We now transition to real data sets and show that our tensor spectral partitioning algorithm provides good cuts for the directed 3-cycle (D3C) data tensor given by Eqn. (4.11). We compare the following algorithms:

- **TSC**: This is our proposed method (Algorithm 2 with $C = 2$), where the data tensor is given by Eqn. (4.11). The sweep cut ordering is provided by the second left eigenvector of $\boldsymbol{P}[\boldsymbol{x}]$.
- **Undirected Laplacian (UL)**: The sweep cut ordering is provided by the second left eigenvector of the transition matrix of the undirected version of the graph.
- **Directed Laplacian (DL)** [5]: The sweep cut ordering is provided by the second left eigenvector of $\boldsymbol{P}_{sym}$ in Eqn. (2.4).
- **Asymmetric Laplacian (AL)** [4]: The sweep cut ordering is provided by the second left eigenvector of $\boldsymbol{P}$.
- **Co-clustering (Co)** [9, 28]: The sweep cut ordering is based on the second left and right singular vectors of a normalized adjacency matrix. Specifically, let $\boldsymbol{D}_{row} = \text{diag}(\boldsymbol{A}\boldsymbol{e})$ and $\boldsymbol{D}_{col} = \text{diag}(\boldsymbol{A}^{\mathsf{T}}\boldsymbol{e})$ and let $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}}$ be the singular value decomposition of $\boldsymbol{D}_{row}^{-1/2}\boldsymbol{A}\boldsymbol{D}_{col}^{-1/2}$. The the sweep cut ordering is provided by $\boldsymbol{D}_{row}^{-1/2}\boldsymbol{U}(:,2)$ or $\boldsymbol{D}_{col}^{-1/2}\boldsymbol{V}(:,2)$. We take the better of the two cuts.
- **Random**: The sweep cut ordering is random. This provides a simple baseline.

### 6.1 Data preprocessing

Before running partitioning algorithms, we first filter the networks as follows: (1) remove all edges that do not participate in any D3C, and (2) take the largest strongly connected component of the remaining network. We perform this filtering to make a fairer comparison between the different partitioning algorithms. Table 1 lists the relevant networks and statistics for the filtered networks that we use in our experiments. We limit ourselves to a few representative networks to illustrate the main patterns we observed. Data for more networks is available in the full version of this paper. Networks are available from SNAP [23].

### 6.2 Results

Figure 4 shows the sweep profiles on the networks in Table 1. The results are for a single cut of the network. The plots show the higher-order conductance (Eqn. (3.9)), higher-order expansion (Eqn. (3.10)), and density of the smaller of the partitioned vertex sets. For `email-EuAll` and `wiki-Talk`, higher-order conductance is the same for most algorithms, but TSC has much bet-

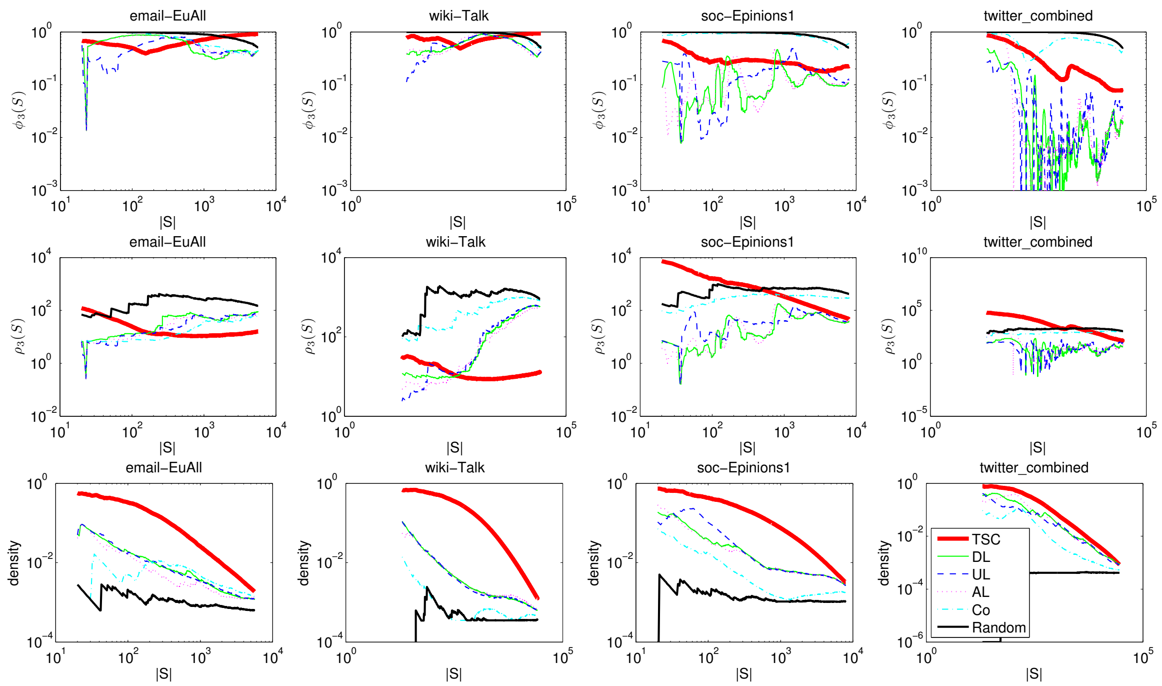


Figure 4: (Top row) Higher-order conductance, $\phi_3(S)$, as a function of the smaller vertex partition set ($|S|$). The size of the vertex set runs from twenty to half the nodes in the network. (Middle row) Higher-order expansion, $\rho_3(S)$. (Bottom row) Density of the cluster. For `email-EuAll` and `wiki-Talk` (left two coumns), higher-order conductance from TSC is on par with other spectral methods, and higher-order expansion is better for large enough clusters. For `soc-Epinions1` and `twitter_combined`, the higher-order conductance and expansion is better using standard clustering algorithms. In all cases, TSC finds much denser clusters.

Table 1: Statistics of networks used for computing directed 3-cycle cuts. The statistics are taken on the largest strongly connected component of the network after removing all edges that do not participate in any D3C.

| Network | $n = \|V\|$ | $m = \|E\|$ | # D3Cs |
|---|---|---|---|
| `email-EuAll` | 11,315 | 80,211 | 183,836 |
| `soc-Epinions1` | 15,963 | 262,779 | 738,231 |
| `wiki-Talk` | 52,411 | 957,753 | 5,138,613 |
| `twitter_combined` | 57,959 | 1,371,621 | 6,921,399 |

ter higher-order expansion when the vertex set gets large enough. On `soc-Epinions1` and `twitter_combined`, standard spectral methods have better higher-order conductance and higher-order expansion. Crucially, in all cases, TSC finds much denser subgraphs. In general, we expect communities with lots of directed 3-cycles to be dense sets. Thus, even though TSC sometimes does not always do well with respect to the score metrics discussed in Sec. 3.4, it is still finding relevant structure.

Since our goal is to explore structural properties of the cuts, we did not tune our TSC algorithms for high performance. Subsequently, we do not compare running times of the algorithms. However, we note that for each network, our straightforward implementation of TSC ran in under 10 minutes using a laptop with 4GB of memory.

## 7 Related work

While the bulk of community detection algorithms are for undirected networks, there is still an abundance of methods for directed networks [25]. There are several spectral algorithms related to partitioning directed networks. The ones we investigated in this paper were based on the undirected Laplacian (*i.e.*, standard spectral clustering but ignoring edge directions), the directed Laplacian [5], the asymmetric Laplacian [4], and co-clustering [9, 28]. Other spectral algorithms are based on dyadic methods [24] and optimizing directed modularity [22].

There is some work in community detection that explicitly targets higher-order structures. Klymko *et al.* weight directed edges in triangles and then revert to a clustering algorithm for undirected networks [19]. Clique percolation builds overlapping communities by examining small cliques [8]. Optimizing the LinkRank metric can identify communities based on information flow [18], which is similar to our use of directed 3-cycles in Sec. 5.1. Multi-way relationships between nodes are also explicitly handled by hypergraph partitioners [17] .

Finally, we mention that tensor factorizations have been used by Huang *et al.* to find overlapping communities [16]. This work uses new spectral techniques for learning latent variable models from higher-order moments [2].

## 8 Discussion

We have provided a framework for tensor spectral clustering that opens the door to further higher-order network analysis. The framework gives the user the flexibility to cluster structures based on his or her application. In Sec. 5 we provided two applications—layered flow networks and anomaly detection—that showed how this framework can lead to better clustering of nodes based on network motifs. For these applications, the networks were small and manually constructed. In future work, we plan to explore these applications on large networks.

In Sec. 6, we explored clustering based on directed 3-cycles. In some cases, TSC provided much better cuts in terms of higher-order expansion. Interestingly, for some networks, simply removing edges that do not participate in a directed 3-cycle and using a standard spectral clustering algorithm is sufficient for finding good cuts with respect to higher-order conductance and higher-order expansion. However, in these cases, we are comparing against baselines optimized for our specific problem. That being said, TSC does always identify much denser clusters. The networks we analyzed were social and internet-based, and it would be interesting to see if similar trends hold for networks derived from physical or biological systems.

For the large networks, we did not perform full directed clustering—we only investigated the sweep profiles. The higher-level goal of this paper is to explore the ideas in higher-order clustering, and we leave full-stack algorithms to future work. One interesting question for such algorithms is whether we should partition based on recursive bisection (Algorithm 2) or k-means. These algorithmic variations provide several opportunities for challenging future work.

## References

[1] N. Alon and V. D. Milman. $\lambda_1$, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.

[2] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *arXiv preprint arXiv:1210.7559*, 2012.

[3] P. Boldi, R. Posenato, M. Santini, and S. Vigna. Traps and pitfalls of topic-biased PageRank. In *Algorithms and Models for the Web-Graph*, pages 107–116. Springer, 2008.

[4] D. Boley, G. Ranjan, and Z.-L. Zhang. Commute times for a directed graph using an asymmetric Laplacian. *Linear Algebra and its Applications*, 435(2):224–242, 2011.

[5] F. Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.

[6] F. R. Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[7] J. Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29–41, 2009.

[8] I. Derényi, G. Palla, and T. Vicsek. Clique percolation in random networks. *PRL*, 94(16):160202, 2005.

[9] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD*, 2001.

[10] N. Durak, A. Pinar, T. G. Kolda, and C. Seshadhri. Degree relations of triangles in real-world networks and graph models. In *CIKM*, 2012.

[11] D. F. Gleich. Hierarchical directed spectral graph partitioning, 2006.

[12] D. F. Gleich, L.-H. Lim, and A. Benson. The Spacey Random Surfer: A stochastic process for multilinear PageRank.

[13] D. F. Gleich, L.-H. Lim, and Y. Yu. Multilinear PageRank. *arXiv*, cs.NA:1409.1465, 2014.

[14] M. S. Granovetter. The strength of weak ties. *American Journal of Sociology*, pages 1360–1380, 1973.

[15] C. J. Hillar and L.-H. Lim. Most tensor problems are NP-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013.

[16] F. Huang, U. Niranjan, M. Hakeem, and A. Anandkumar. Fast detection of overlapping communities via online tensor methods, 2013.

[17] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3):285–300, 2000.

[18] Y. Kim, S.-W. Son, and H. Jeong. Finding communities in directed networks. *Phys. Rev. E*, 81:016103, Jan 2010.

[19] C. Klymko, D. F. Gleich, and T. G. Kolda. Using triangles to improve community detection in directed networks. In *Proceedings of the ASE BigData Conference*, 2014.

[20] T. G. Kolda and J. R. Mayo. Shifted power method for computing tensor eigenpairs. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1095–1124, 2011.

[21] G. Kossinets and D. J. Watts. Empirical analysis of an evolving social network. *Science*, 311(5757):88–90, 2006.

[22] E. A. Leicht and M. E. Newman. Community structure in directed networks. *PRL*, 100(11):118703, 2008.

[23] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[24] Y. Li, Z.-L. Zhang, and J. Bao. Mutual or unrequited love: Identifying stable clusters in social networks with uni-and bi-directional links. In *Algorithms and Models for the Web Graph*, pages 113–125. Springer, 2012.

[25] F. D. Malliaros and M. Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95 – 142, 2013.

[26] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey. Shaping communities out of triangles. In *CIKM*, 2012.

[27] L. Qi. Eigenvalues of a real supersymmetric tensor. *Journal of Symbolic Computation*, 40(6):1302–1324, 2005.

[28] K. Rohe and B. Yu. Co-clustering for directed graphs; the stochastic co-blockmodel and a spectral algorithm. *arXiv preprint arXiv:1204.2296*, 2012.

[29] M. Rosvall, A. V. Esquivel, A. Lancichinetti, J. D. West, and R. Lambiotte. Memory in network flows and its effects on spreading dynamics and community detection. *Nature communications*, 5, 2014.

[30] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms*. 2005.