SIAM J. MATH. DATA SCI.
Vol. 6, No. 2, pp. 481–503

Scalable Tensor Methods for Nonuniform Hypergraphs*

Sinan G. Aksoy†, Ilya Amburg‡, and Stephen J. Young†

Abstract. While multilinear algebra appears natural for studying the multiway interactions modeled by hypergraphs, tensor methods for general hypergraphs have been stymied by theoretical and practical barriers. A recently proposed adjacency tensor is applicable to nonuniform hypergraphs but is prohibitively costly to form and analyze in practice. We develop tensor times same vector (TTSV) algorithms for this tensor, which improve complexity from $O(n^r)$ to a low-degree polynomial in r, where n is the number of vertices and r is the maximum hyperedge size. Our algorithms are implicit, avoiding formation of the order r adjacency tensor. We demonstrate the flexibility and utility of our approach in practice by developing tensor-based hypergraph centrality and clustering algorithms. We also show that these tensor measures offer complementary information to analogous graph-reduction approaches on data and are also able to detect higher-order structure that many existing matrix-based approaches provably cannot.

Key words. hypergraph, adjacency tensor, tensor times same vector, tensor-free methods, centrality, clustering

MSC codes. 05C65, 15A69, 05C50, 05C85

DOI. 10.1137/23M1584472

1. Introduction.

The study of hypergraphs is fraught with choices of representation. From Laplacians [13, 15, 46, 55], to probability transition matrices [16, 17, 25], to variants of incidence and adjacency matrices [2, 14, 37], there is no shortage of proposed hypergraph data structures. Despite these options, selecting among them can be challenging, as each comes with significant and sometimes nuanced limitations. For example, adjacency, random walk, and Laplacian matrices are typically lossy in that they only contain information about the hypergraph's clique expansion graph, thereby losing the information encoded in higher-order interactions [1, 25]. In contrast, rectangular incidence matrices faithfully model hypergraphs but have analytical limitations: for instance, their singular values reflect information about the weighted line graph and clique expansion reductions, which do not uniquely identify the hypergraph [30]. Arguably, these challenges stem from mismatching hypergraphs, models of higher-dimensional relationships, with two-dimensional arrays.

*Received by the editors July 6, 2023; accepted for publication (in revised form) February 15, 2024; published electronically June 11, 2024. Authors listed in alphabetical order.
https://doi.org/10.1137/23M1584472
Funding: The authors gratefully acknowledge the funding support from the Applied Mathematics Program within the U.S. Department of Energy's Office of Advanced Scientific Computing Research as part of Scalable Hypergraph Analytics via Random Walk Kernels (SHARWK). Pacific Northwest National Laboratory is operated by Battelle for the DOE under contract DE-AC05-76RL01830. **PNNL Information Release:** *PNNL-SA*-186918.

†Pacific Northwest National Laboratory, Richland, WA 99352 USA (sinan.aksoy@pnnl.gov, stephen.young@pnnl.gov).
‡Corresponding author. Pacific Northwest National Laboratory, Richland, WA 99352 USA (ilya.amburg@pnnl.gov).

Tensor arrays, therefore, appear a natural choice for hypergraph-native analyses. However, their application in the hypergraph setting poses immediate conceptual and computational challenges. A primary theoretical barrier is that nonuniform hypergraphs do not afford an obvious tensor representation. For this reason, despite real hypergraph-structured data nearly always exhibiting hyperedges of varying sizes, much of the existing tensor literature on hypergraphs is limited to the uniform case [10, 11, 47], relies on augmenting the hypergraph with auxiliary nodes [42, 54], or synthesizes a collection of differently sized tensors for each hyperedge size [29]. One notable exception of using tensors to *directly* study nonuniform hypergraphs, however, is the adjacency tensor recently proposed by Banerjee, Char, and Mondal [6]. Loosely speaking, this tensor encodes nonuniform hyperedges by "inflating" each to the maximum hyperedge size r. This yields an order r, n-dimensional tensor, where n is the number of vertices. Consequently, the nonuniform adjacency tensor solves a conceptual challenge but poses a computational one: its explicit formation and analysis are intractable for nearly any hypergraph data with nontrivially sized hyperedges, since fundamental tensor operations like tensor times same vector (TTSV) have cost $O(n^r)$.

In this work, we focus on ameliorating these computational challenges to enable use of the hypergraph adjacency tensor in practice. Our main focus is creating efficient algorithms for TTSV. In particular, we drastically speed up the tensor times same vector in all modes but one (TTSV1) operation from $O(n^r)$ to a low-degree polynomial in r. We perform an analogous speedup for tensor times same vector in all modes but two (TTSV2) using an approach that easily generalizes to tensor times same vector in all modes but k (TTSV$_K$). Moreover, our methods are implicit and tensor-free, avoiding formation of the costly order-r tensor. We achieve these improvements using combinatorial methods that exploit the nuanced symmetry of the hypergraph adjacency tensor. We derive best- and worst-case complexity bounds of our algorithms and supplement these analytical results with timing experiments on real data.

Since TTSV is a workhorse in many tensor algorithms such as canonical polyadic (CP) decomposition and tensor eigenvector computation [8, 9, 10, 11, 32, 33, 48], our algorithms enable a host of tensor-based hypergraph analytics. We illustrate this by proposing simple tensor-based centrality and clustering algorithms where TTSV is the primary subroutine. For centrality, we apply recent nonlinear Z- and H-eigenvector formulations [8] to nonuniform hypergraphs, whose existence is guaranteed by the Perron–Frobenius theorem for tensors [45]. For clustering, we outline an approach that uses fast CP decomposition to obtain an embedding for the hypergraph, which is then fed into k-means [24], or any metric space approach, for clustering. We then study these measures experimentally, showing each offers complementary node importance information on data. Moreover, we show that these tensor measures detect differences in structured hypergraphs with identical underlying graph information, as given by their weighted clique and line graphs. In contrast to many existing hypergraph methods, this means tensor approaches enabled by our algorithms analyze multiway interactions in hypergraph data directly—without reducing them to groups of pairwise interactions modeled by graphs.

The paper is structured as follows: Section 2 reviews the necessary preliminaries. Section 3 presents our algorithms for TTSV1. As the details of the algorithms and analysis for TTSV2 are similar to that of TTSV1, we defer their discussion to section SM1 of the supple-

mentary material. Section 4 applies these algorithms to develop a tensor-based approach for nonuniform hypergraph centrality and clustering. Section 5 concludes and highlights avenues for future work.

2. Preliminaries.

A *hypergraph* $H = (V, E)$ is a set V of n vertices and a set E of m hyperedges, each of which is a subset of V. The *degree* of a vertex v and hyperedge e is $d(v) = |\{e \in E : v \in e\}|$ and $d(e) = |e|$, respectively. The *rank* of a hypergraph is $\max_e d(e)$, and if $d(e) = k$ for all $e \in E$, we call the hypergraph *k-uniform*. The *volume* of a hypergraph is $\mathrm{Vol}(H) = \sum_{v \in V} d(v) = \sum_{e \in E} d(e)$. The set of hyperedges to which v belongs is denoted by $E(v)$, while $E(u, v)$ denotes those to which both u and v jointly belong. The *clique expansion* of a hypergraph (V, E) is the graph on V with edge set $\{\{u, v\} \in V \times V : u, v \in e \text{ for some } e \in E\}$. For more basic hypergraph terminology, we refer the reader to [2, 12].

A tensor of order r is an r-dimensional array. Lowercase bold letters denote vectors, e.g., $\mathbf{a}$, while uppercase bold letters in regular and Euler script denote matrices and tensors, e.g., $\mathbf{X}$ and $\boldsymbol{\mathcal{X}}$, respectively. For a tensor $\boldsymbol{\mathcal{X}}$, the value at index $i_1, \ldots, i_r$ is given by $\boldsymbol{\mathcal{X}}_{i_1,\ldots,i_r}$. In general, we assume the vertices of a hypergraph are indexed by $[n] = \{1, \ldots, n\}$ and that each index in a tensor $\boldsymbol{\mathcal{X}}$ has n components, $i_1, \ldots, i_r \in [n]^r$. The notation $\mathbf{a}^{[k]}$, $\mathbf{a} \odot \mathbf{b}$, and $\mathbf{a} \oslash \mathbf{b}$ denote the elementwise power, product, and division operations, respectively. Following standard notation from generating functions (see, for instance, [53]), we will denote the coefficient of t^r in the generating function $f(t)$ by $f[t^r]$. Lastly, we use Bachmann–Landau asymptotic notation. Our algorithm complexities are multivariate and depend on the rank r, number of hyperedges m, and volume $\mathrm{Vol}(H)$ of the hypergraph H.

Tensors have long been used to encode adjacency in uniform hypergraphs. In particular, a rank r uniform hypergraph's adjacency tensor is order r with

$$\boldsymbol{\mathcal{A}}_{v_1,\ldots,v_r} = \begin{cases} w & \text{if } \{v_1, \ldots, v_r\} \in E, \\ 0 & \text{otherwise,} \end{cases}$$

where w refers to a chosen weight, such as $w = 1$ or $1/(r-1)!$. Recently, [6] generalized this to nonuniform hypergraphs. Underlying this definition is a concept we call hyperedge *blowups*.

Definition 2.1. *Given a hyperedge e of a rank r hypergraph, we call the sets*

$$\beta(e) = \{i_1 \ldots i_r \in e^r : \textit{ for each } v \in e, \textit{ there is } j \textit{ such that } i_j = v\},$$
$$\kappa(e) = \{x : x \textit{ is a size } r \textit{ multiset with support } e\}$$

the "blowups" and "unordered blowups" of hyperedge e, respectively.

For example, for $e = \{1, 3\}$ in a rank 3 hypergraph, we have

$$\beta(e) = \{(1,1,3), (1,3,1), (1,3,3), (3,1,1), (3,1,3), (3,3,1)\},$$
$$\kappa(e) = \{\{1,1,3\}, \{1,3,3\}\}.$$

The nonuniform hypergraph adjacency tensor [6] places nonzeros in positions corresponding to blowups of hyperedges, with values weighted by the size of that hyperedge's blowup set.

Definition 2.2 (nonuniform hypergraph adjacency tensor [6]). *For a rank r hypergraph, its adjacency tensor $\boldsymbol{\mathcal{A}}$ is order r and is defined elementwise for each hyperedge e,*

$$\mathcal{A}_{p_1 \dots p_r} = \begin{cases} w_e & \text{if } p_1 \dots p_r \in \beta(e), \\ 0 & \text{otherwise,} \end{cases}$$

where w_e denotes a chosen hyperedge weighting function.

We note that while [6] takes $w_e = \frac{|e|}{|\beta(e)|}$ so that TTSV1 with the all-ones vector yields the degree sequence, our algorithms will not require this choice. Furthermore, in addition to adjacency tensors, our methods also easily adapt to the Laplacian tensors in [6], as we illustrate in section 4.3. Lastly, a different approach to building nonuniform hypergraph adjacency tensors relies on inserting copies of an auxiliary "dummy" vertex within each hyperedge [54], rather than using those already present in the hyperedge via blowups. While we focus on the blowup approach, our algorithms easily adapt to this case as well. The TTSV operations we tailor for $\mathcal{A}$ form the backbone of many tensor algorithms such as tensor decomposition [32, 33, 48] and eigenvector computation [8, 9, 10, 11] and are defined in general as follows: given an order r tensor $\mathcal{X}$ and a vector $\mathbf{b} \in \mathbb{R}^n$, $\mathcal{X}\mathbf{b}^{r-1} \in \mathbb{R}^n$ denotes the TTSV1 operation given by

$$\left[\mathcal{X}\mathbf{b}^{r-1}\right]_{i_1} = \sum_{i_2=1}^{n} \cdots \sum_{i_r=1}^{n} \mathcal{X}_{i_1,\dots,i_r} \prod_{k=2}^{r} \mathbf{b}_{i_k}, \tag{2.1}$$

and TTSV2, denoted $\mathcal{X}\mathbf{b}^{r-2} \in \mathbb{R}^{n \times n}$, is given by

$$\left[\mathcal{X}\mathbf{b}^{r-2}\right]_{i_1,i_2} = \sum_{i_3=1}^{n} \cdots \sum_{i_r=1}^{n} \mathcal{X}_{i_1,\dots,i_r} \prod_{k=3}^{r} \mathbf{b}_{i_k}. \tag{2.2}$$

Given the TTSV2 matrix, the TTSV1 vector is easily obtained through right-multiplication by $\mathbf{b}$. However, whenever the full TTSV2 is unnecessary, it suffices to compute TTSV1 directly. Lastly, we note that computing TTSV without explicitly forming the tensor is an example of an *implicit* tensor operation and has been recently studied in the context of moment tensors of Gaussian mixture models [43, 48].

3. Tensor times same vector for hypergraphs.

We now develop efficient, implicit TTSV methods for analyzing the hypergraph adjacency tensor $\mathcal{A}$. Rather than explicitly constructing, storing, or accessing elements of $\mathcal{A}$, our algorithms directly facilitate tensor operations on the input hypergraph. We divide our work into two approaches: first, we present simple algorithms which achieve speedup over the naive approach by leveraging several combinatorial observations on the unordered blowups discussed in Definition 2.1. Then, we further improve upon these algorithms by using generating function theory. Code for our TTSV algorithms is available at https://github.com/pnnl/GENTTSV, in Python for hypergraph libraries HyperNetX [44], HypergraphX [38], and XGI [36], and Julia for SimpleHypergraphs [4].

3.1. Unordered blowup approach.

This approach for computing TTSV1 for $\mathcal{A}$ will use the following basic combinatorial facts.

Lemma 3.1. *Let $e = \{v_1, \dots, v_k\}$ be a hyperedge of a rank r and $\beta(e)$ and $\kappa(e)$ as defined in Definition 2.1. Then the following hold:*

(a) $|\beta(e)| = |e|! \cdot \left\{ {r \atop |e|} \right\}$, *where* $\left\{ {r \atop |e|} \right\}$ *denotes the Stirling number of the second kind.*[1]
(b) $|\kappa(e)| = \binom{r-1}{r-|e|}$.
(c) *For given vertices* $u \in e$ *and* $x \in \kappa(e)$, *the number of blowups* $i_1, \ldots, i_r \in \beta(e)$ *that have support equal to that of* x *with* $i_j = u$ *is given by the multinomial coefficient*

$$\phi_1(x,u) := \binom{r-1}{m_x(v_1), \ldots, m_x(u)-1, \ldots, m_x(v_k)},$$

where $m_x(w)$ *denotes the multiplicity of node* w *in* x.

TTSV1-Unord presents the algorithm for the unordered blowup approach. The main idea is to exploit the nonzero pattern in $\mathcal{A}$. Recalling that i_1 corresponds to a node index when applying the TTSV1 vector given in (2.1) to $\mathcal{A}$, the only nonzeros that contribute to the sum for this component correspond to blowups of all hyperedges to which node i_1 belongs. This observation significantly reduces the cost of TTSV1 from the naive $O(n^r)$ approach in the absence of the special structure induced by the hypergraph and leads to a less naive algorithm: for every vertex, iterate through its hyperedges and update the sum over all elements of the corresponding blowups. However, this approach still requires explicitly enumerating blowups, which is extremely costly. Instead, TTSV1-Unord only considers *unordered* blowups. By Lemma 3.1(c), there are exactly $\phi_1(x,v)$ many blowups with v in a fixed position that correspond to a given unordered blowup $x \in \kappa(e)$. The correctness of TTSV1-Unord immediately follows. We note the cost savings here occurs *per hyperedge,* yielding speedups of several orders of magnitude for real datasets with larger r.

Proposition 3.2. *Let* $H = (V, E)$ *be a rank* r *hypergraph with* m *edges. Then* TTSV1-Unord *runs in time* $\Theta(\sum_{e \in E} r|e|\binom{r-1}{r-|e|})$. *Let* $\epsilon = \min\{\frac{\mathrm{Vol}(H)}{m}, 1 - \frac{\mathrm{Vol}(H)}{m}\}$; *then this running time is at least*

$$\Omega\left(r^2 m\right) \quad \text{and at most} \quad O\left(\epsilon m r^{3/2} 2^r\right).$$

TTSV1-Unord: TTSV1 via unordered blowups

Data: rank r hypergraph (V, E, w), vector $\mathbf{b}$
Result: $\mathcal{A}\mathbf{b}^{r-1} = \mathbf{s}$
for $v \in V$ **do**
  $c \leftarrow 0$
  for $e \in E(v)$ **do**
    for $x \in \kappa(e)$ **do**
      $c \mathrel{+}= w_e \dfrac{\phi_1(x,v)}{\mathbf{b}_v} \displaystyle\prod_{u \in x} \mathbf{b}_u$
    end
  end
  $\mathbf{s}_v \leftarrow c$
end
return s

[1] An explicit formula for Stirling numbers of the second kind is $\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$.

Proof. Calculate the innermost terms of the sum runs in $\Theta(r)$; thus the runtime is

$$\sum_{v\in V}\sum_{e\in E(v)}\sum_{x\in\kappa(e)}\Theta(r)=\sum_{e\in E}\sum_{v\in e}\sum_{x\in\kappa(e)}\Theta(r)=\sum_{e\in E}\Theta\left(r\,|e|\binom{r-1}{r-|e|}\right).$$

Thus, we have that the runtime is completely determined by the sizes of the edges of H. Now, noting that, by algebraic manipulation, we have that $r|e|\binom{r-1}{r-|e|}=|e|^2\binom{r}{|e|}$, we define $f:\{2,\dots,r\}\to\mathbb{N}$ by $f(k)=k^2\binom{r}{k}$. We observe that

$$\frac{f(k+1)}{f(k)}=\frac{(k+1)^2\binom{r}{k+1}}{k^2\binom{r}{k}}=\frac{(k+1)^2k!(r-k)!}{k^2(k+1)!(r-k-1)!}=\frac{k+1}{k}\frac{r-k}{k}.$$

Thus the maximum of f occurs at $k^*=\lfloor\frac{r}{2}\rfloor+1$ and f is monotonically increasing below k^* and monotonically decreasing above k^*. As $f(2),f(r)\in\Theta(r^2)$, we have that the runtime of TTSV1-Unord is bounded below by $\Theta(r^2m)$.

For the upper bound, let $e_1,\dots,e_m$ be a sequence of integers such that $\sum_i e_i=\mathrm{Vol}(H)$ and $\sum_i f(e_i)$ is maximized. By the monotonicity of f, either $e_i\le k^*$ for all i or $e_i\ge k^*$ for all i. Let $C=\lfloor 2\sqrt{r}\rfloor$ and suppose k and ℓ are such that $|k-k^*|,|\ell-k^*|\ge C$; then we have

$$\begin{aligned}\frac{f(k)+f(\ell)}{f(k^*)}=\frac{k^2\binom{r}{k}+\ell^2\binom{r}{\ell}}{(k^*)^2\binom{r}{k^*}}\le\frac{2r^2\binom{r}{k^*-C}}{(k^*)^2\binom{r}{k^*}}&\le 8\frac{(k^*)!(r-k^*)!}{(k^*-C)!(r-k^*+C)!}\\&=8\prod_{j=1}^{C}\frac{k^*-C+j}{r-k^*+j}\\&\le 8\left(\frac{k^*}{r-k^*+C}\right)^C\\&=8\left(1-\frac{r-2k^*+C}{r-k^*+C}\right)^C\\&\le 8\left(1-\frac{C-1}{\frac{r}{2}+C}\right)^C\\&\le 8\exp\left(-\frac{C(2C-2)}{r+2C}\right)\le 1.\end{aligned}$$

This implies, by the maximality of $e_1,\dots,e_m$, that $e_i\in\{2,r\}\cup[k^*-C,k^*+C]$ for all i. Further, since $f(k)=\Theta(f(k^*))=\Theta(r^{3/2}2^r)$ for all $k^*-C\le k\le k^*+C$, this gives the desired upper bound on the runtime of TTSV1-Unord. ∎

3.2. Generating function approach. The runtime of TTSV1-Unord is dominated by the innermost loop, iterating over unordered blowups $\kappa(e)$ for all edges e. Indeed, for most real datasets, where the size of a typical edge is much smaller than the largest edge, this is a significant bottleneck as the size of this set scales exponentially with the size of the edge, i.e., $|\kappa(e)|=O(r^{|e|})$. However, by taking a generating function approach the inner loop can be considerably simplified and accelerated. To illustrate this, consider the summation over the inner loop for the case when $e=\{u,v\}\in E(v)$:

$$\sum_{x\in\kappa(\{u,v\})} w_e \frac{\phi_1(x,v)}{\mathbf{b}_v}\prod_{i\in x}\mathbf{b}_i = \frac{w_e}{\mathbf{b}_v}\sum_{k_u=1}^{r-1}\phi_1(u^{k_u}v^{r-1-k_u},v)\mathbf{b}_u^{k_u}\mathbf{b}_v^{r-k_u}$$

$$= w_e\sum_{k_u=1}^{r-1}\frac{(r-1)!}{k_u!(r-1-k_u)!}\mathbf{b}_u^{k_u}\mathbf{b}_v^{r-1-k_u}$$

$$(*) \qquad = w_e(r-1)!\sum_{k_u=1}^{r-1}\frac{1}{k_u!}\mathbf{b}_u^{k_u}\frac{1}{(r-1-k_u)!}\mathbf{b}_v^{r-1-k_u}.$$

We note that the summation in (*) can be viewed as a convolution of two sequences

$$0, \frac{1}{1!}\mathbf{b}_u^1, \frac{1}{2!}\mathbf{b}_u^2, \ldots, \frac{1}{(r-1)!}\mathbf{b}_u^{r-1} \quad \text{and} \quad \frac{1}{0!}\mathbf{b}_v^0, \frac{1}{1!}\mathbf{b}_v^1, \frac{1}{2!}\mathbf{b}_v^2, \ldots, \frac{1}{(r-1)!}\mathbf{b}_v^{r-1}.$$

Alternatively, if we define $u_r(t) = \sum_{k=1}^{r-1}\frac{1}{k!}\mathbf{b}_u^k t^k$ and $v_r(t) = \sum_{k=0}^{r-1}\frac{1}{k!}\mathbf{b}_v^k t^k$, the summation (*) may be thought of as the coefficient of t^{r-1} in $u_r(t)v_r(t)$. Recall that the coefficient of t^r in the generating function $f(t)$ is denoted by $f[t^r]$. In fact, for any $d \geq 0$ the summation is equal to $(u_{r+d}v_{r+d})[t^{r-1}]$, and thus taking

$$u(t) = \sum_{k=1}^{\infty}\frac{1}{k!}\mathbf{b}_u^k t^k = \exp(\mathbf{b}_u t) - 1 \quad \text{and} \quad v(t) = \sum_{k=0}^{\infty}\frac{1}{k!}\mathbf{b}_v^k t^k = \exp(\mathbf{b}_v t),$$

we have that $(uv)[t^{r-1}]$ is the value of the inner sum for an arbitrary choice of largest edge size r. This argument may be easily extended to an arbitrary edge $\{v, u_1, \ldots, u_\ell\} = e \in E(v)$ by noting that the contribution to the inner sum by blowups of e with k_v copies of v and $k_1, \ldots, k_\ell$ copies of $u_1, \ldots, u_\ell$ is given by

$$w_e\frac{(r-1)!}{(k_v)!\prod_{j=1}^{\ell}(k_j)!}\mathbf{b}_v^{k_v}\prod_{j=1}^{\ell}\mathbf{b}_{u_j}^{k_j} = w_E(r-1)!\frac{\mathbf{b}_v^{k_v}}{(k_v)!}\prod_{j=1}^{\ell}\frac{\mathbf{b}_{u_j}^{k_j}}{(k_j)!}.$$

Now by summing over valid choices of k_v and $k_1, \ldots, k_\ell$, we get that the total contribution of the inner loop from the edge e is given by

$$w_e(r-1)!\sum_{k_v=0}^{r-\ell-1}\sum_{\substack{1\leq k_j\leq r-\ell\\ k_v+\sum_j k_j=r}}\frac{\mathbf{b}_v^{k_v}}{(k_v)!}\prod_{j=1}^{\ell}\frac{\mathbf{b}_{u_j}^{k_j}}{(k_j)!}.$$

As above, if we introduce the variable t, this can be viewed as the coefficient of t^{r-1} in a generating function, specifically

$$\left(w_e(r-1)!\sum_{k_v=0}^{\infty}\sum_{k_1,\ldots,k_\ell=1}^{\infty}\frac{\mathbf{b}_v^{k_v}t^{k_r}}{(k_v)!}\prod_{j=1}^{\ell}\frac{\mathbf{b}_{u_j}^{k_j}t^{k_j}}{(k_j)!}\right)[t^{r-1}].$$

Thus, by interchanging the products and summations we have that the inner sum in TTSV1-UNORD is given by

$$w_e(r-1)!\left(\exp(\mathbf{b}_v t)\prod_{u\in e-\{v\}}(\exp(\mathbf{b}_u t)-1)\right)[t^{r-1}].$$

Alternatively, the expression for the inner sum can be derived analytically from the generating function representation for TTSV. Specifically, using the same approach as above, we have that

$$\boldsymbol{\mathcal{X}}\mathbf{b}^r = \left(\sum_{e\in E} w_e r! \prod_{v\in e} (\exp(\mathbf{b}_v t) - 1)\right)[t^r].$$

Since, as shown in [34, Lemmas 3.1 and 3.3] for $k = 1$ and 2, any symmetric tensor $\boldsymbol{\mathcal{X}}$ and vector $\mathbf{b}$ satisfy $\nabla^k(\boldsymbol{\mathcal{X}}\mathbf{b}^r) = \frac{r!}{(r-k)!}\boldsymbol{\mathcal{X}}\mathbf{b}^{r-k}$, and we have that

$$\boldsymbol{\mathcal{X}}\mathbf{b}^{r-k} = \left(\sum_{e\in E} w_e (r-k)! \nabla^k \prod_{v\in e} (\exp(\mathbf{b}_v t) - 1)\right)[t^{r-k}].$$

For example, the (u, u) diagonal terms of TTSV2 are given by

$$\begin{aligned}&\left(\sum_{e\in E} w_e (r-2)! \frac{\partial^2}{\partial \mathbf{b}_u^2} \prod_{v\in e} (\exp(\mathbf{b}_v t) - 1)\right)[t^{r-2}] \\ &= \left(\sum_{e\in E(u)} w_e (r-2)! \exp(\mathbf{b}_u t) \prod_{v\in e-\{u\}} (\exp(\mathbf{b}_v t) - 1)\right)[t^{r-2}].\end{aligned}$$

While this generating function simplifies the exposition of these algorithms, in practice we require an efficient means of extracting the appropriate coefficient from the generating function. Accordingly, we present two different options for evaluating the generating function approach, a subset expansion approach, Subset-Gen, which runs in time $\Theta\big((|e| + \log_2(r))\, 2^{|e|}\big)$, and a modification of the fast Fourier transform (FFT) [20] approach to polynomial multiplication, FFT-Gen, which runs in time $\Theta(|e|\, r \log_2(r))$. We note that the runtimes of these two algorithms are asymptotically equivalent for edges of size $\log_2(r) + \log_2 \log_2(r) + \Theta(1)$, with Subset-Gen asymptotically faster for smaller edges and FFT-Gen asymptotically faster for larger edges. For convenience, we assume that all edges of size at most $\log_2(r) + \log_2 \log_2(r)$ are evaluated using Subset-Gen and larger edges are evaluated via FFT-Gen. Figure 1

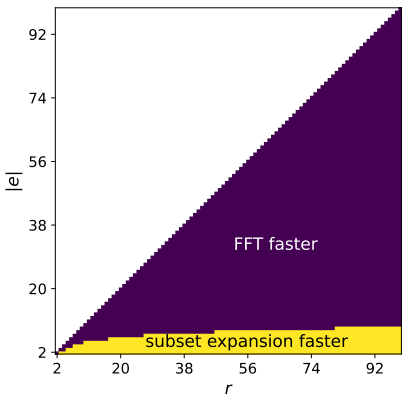


Figure 1. *Efficiency of FFT versus subset expansion for generating function evaluation.*

plots the edge size region where subset expansion outperforms FFT (and vice-versa). It is interesting to note that difference in runtimes between the two approaches can be understood by which computationally efficient subroutines are being used. For instance, FFT-Gen relies on the FFT as a computationally efficient subroutine and as a consequence must take the time to form all the degree r polynomials associated with the edge e and operate on these polynomials (which are larger than the original edge). In contrast to this approach, Subset-Gen relies on the fact that arbitrary coefficients of exponential generating functions can be efficiently evaluated. In particular, Subset-Gen relies on the fact that $e^{at}\prod_{i=1}^{k}\left(e^{b_i t}-1\right)$ can be expanded to

$$e^{at}\sum_{S\subseteq[k]}(-1)^{|\overline{S}|}\prod_{s\in S}e^{b_s t}=\sum_{S\subseteq[k]}(-1)^{k-|S|}e^{(a+\sum_{s\in S}b_s)t}.$$

The t^r coefficient of each of these summands can then be trivially evaluated. Thus, there is an exponential dependence on the edge size and minimal dependence on the maximum edge size r.

While not explored in this work, we note that further asymptotic speedups are possible by hybridizing these two approaches. Specifically, by partitioning the edge e into small sets $e_1,\ldots,e_k$ of size approximately $\log\log(r)$, the full generating function for these can be calculated for the sets individually using a variant of the Subset-Gen approach, and then this set of generating functions could be combined using the FFT.

Proposition 3.3. *Let $H=(V,E)$ be a rank r hypergraph with m edges, and let $k^*=\log_2(r)+\log_2\log_2(r)$. The asymptotic runtime of* TTSV1-Gen *is given by*

$$\sum_{\substack{e\in E\\ |e|\le k^*}}|e|\,2^{|e|}\log_2(r)+\sum_{\substack{e\in E\\ |e|>k^*}}|e|^2\,r\log_2(r).$$

FFT-Gen: FFT evaluation of generating function

Data: $a,b_1,\ldots,b_k\in\mathbb{R}$ and $r\in\mathbb{N}$.
Result: $\left(\exp(at)\prod_i\left(\exp(b_i t)-1\right)\right)[t^r]$
$f\leftarrow\sum_{j=0}^{r}\frac{a^j}{j!}t^j$
for $i=1,\ldots,k$ **do**
  $g\leftarrow\sum_{j=1}^{r}\frac{b_i^j}{j!}t^j$
  $f\leftarrow$ multFFT(f,g)
  $f\leftarrow\sum_{j=0}^{r}f[t^j]t^j$
end
return $f[t^r]$

multFFT: Multiplication using the FFT

Data: polynomials $f(t)$ and $g(t)$
Result: $f(t)g(t)$
$c(t)\leftarrow FFT^{-1}(FFT(g(t))\odot FFT(f(t)))$
return $c(t)$

Subset-Gen: Subset enumeration evaluation of generating function

Data: $a,b_1,\ldots,b_k\in\mathbb{R}$ and $r\in\mathbb{N}$
Result: $\left(\exp(at)\prod_i\left(\exp(b_i t)-1\right)\right)[t^r]$
$c\leftarrow 0$
for $S\subseteq[k]$ **do**
  $c\mathrel{+}=(-1)^{k-|S|}\left(a+\sum_{s\in S}b_s\right)^r$
end
return $c/r!$

If, in addition, the average hyperedge size $\delta = \frac{\mathrm{Vol}\, H}{m}$ *is such that there is some constant* $\epsilon > 0$ *such that* $2 + \epsilon < \delta < (1-\epsilon)r$, TTSV1-Gen *runs in time at least*

$$\begin{cases} \Omega\left(\mathrm{Vol}(H) 2^{\delta} \log_2(r)\right), & \delta < \frac{1}{2}k^*, \\ \Omega\left(\mathrm{Vol}(H) 2^{(1-o(1))\delta} \log_2(r)\right), & \frac{1}{2}k^* \le \delta \le k^*, \\ \Omega\left(\mathrm{Vol}(H)\delta r \log_2(r)\right), & k^* < \delta, \end{cases}$$

and at most

$$O\left(r^2 \log_2(r)\, \mathrm{Vol}(H)\right).$$

Proof. As in the proof of Proposition 3.2, the runtime depends on the hyperedge size sequence, yielding $\Theta(\sum_{e\in E} |e| h(|e|, r))$, where $h(k, r)$ is the runtime of extracting the coefficient of t^{r-1} in the generating function associated with an edge of size k. If, for every edge, we use the faster of FFT-Gen and Subset-Gen, this gives that

$$h(k,r) = \begin{cases} \log_2(r) 2^k, & k \le k^*, \\ kr \log_2(r), & k^* < k. \end{cases}$$

As the shape of the edge size distribution has different effects for "small" edges (those of size at most k^*) and "large" edges (those with size at least k^*), we consider each of these distributions separately. To that end, let m_S be the number of small edges, and let m_F be the number of large edges, with δ_S and δ_F denoting the respective average edge sizes. We note that $m = m_S + m_F$ and $\delta_S m_S + \delta_F m_F = \mathrm{Vol}(H) = \delta m$. We first note that by standard results, the runtime for large edges is minimized when all edges have the same size, δ_F, and maximized when all edges have size either k^* or r.

We now consider the extremal runtimes for the small edges and show that local modifications of the edge sizes will lead to both extremal runtimes. In particular, let e and f be small edges such that $|e| + 1 < |f| \le k^*$; then

$$\begin{aligned} \frac{|e|\, h(|e|, r) + |f|\, h(|f|, r)}{\log_2(r)} &= |e|\, 2^{|e|} + |f|\, 2^{|f|} \\ &= (|e|+1)\, 2^{|e|+1} + (|f|-1)\, 2^{|f|-1} - (|e|+2)\, 2^{|e|} + (|f|+1)\, 2^{|f|-1} \\ &\ge (|e|+1)\, 2^{|e|+1} + (|f|-1)\, 2^{|f|-1} \\ &= \frac{(|e|+1)\, h(|e|+1, r) + (|f|-1)\, h(|f|-1, r)}{\log_2(r)}. \end{aligned}$$

As a consequence, the runtime of the small edges is minimized when all m_S of the edges have the same size, δ_S, and it is maximized when every small edge has size either 2 of k^*.

Thus, the maximum total runtime is at most

$$8 \log_2(r) \frac{(r-\delta)m - (r-k^*)m_k}{r-2} + (k^*)^2 r \log_2(r) m_k + r^3 \log_2(r) \frac{(\delta-2)m - (k^*-2)m_k}{r-2},$$

where m_k is the number of edges of size k^*. As the coefficient on m_k in this expression is asymptotically negative, this is maximized when m_k is 0, that is, when the only edge sizes are either 2 and r, and thus the asymptotic runtime is $O(r^2 \log_2(r)\, \mathrm{Vol}(H))$.

To address the minimum runtime, we recall from above that the runtime is minimized if all the small edges and all the large edge have the same size. For notational convenience, we define $V_S = \delta_S m_S$ and $V_F = \delta_F m_F$ as the volume of edges which are processed by subset expansion and FFT, respectively. Then, the asymptotic runtime is given by

$$V_S 2^{\frac{V_S}{m_S}} \log_2(r) + m_F \left(\frac{V_F}{m_F}\right)^2 r \log_2(r) = V_S 2^{\delta_S} \log_2(r) + V_F \delta_F r \log_2(r).$$

Now suppose that $V_F/\operatorname{Vol} H \in \Omega(1)$. Then, as $\delta_F \geq \delta$, we see that

$$\frac{V_F \delta_F r \log_2(r)}{\operatorname{Vol}(H)\delta r \log_2(r)} \in \Omega(1).$$

In particular, this implies that the only case where the runtime is $o(\operatorname{Vol}(H)\delta r \log_2(r))$ is when $V_S = (1 - o(1))\operatorname{Vol}(H)$ and $\delta_S = (1 - o(1))\delta$. As $\delta_S < k^*$, this implies that if $\delta \geq k^*$, then the minimal runtime is $\Omega(\operatorname{Vol}(H)\delta)$.

We note that we may view V_S and V_K as functions of δ_S and δ_K, yielding

$$f(\delta_S, \delta_F) = \frac{m\delta_K - \operatorname{Vol}(H)}{\delta_K - \delta_S} \delta_S 2^{\delta_s} \log_2(r) + \frac{\operatorname{Vol}(H) - m\delta_S}{\delta_F - \delta_S} \delta_F^2 r \log_2(r).$$

It is a straightforward, albeit tedious, exercise to show that, taken as a function of δ_F, f is minimized when δ_F takes the value

$$\delta_F^* = \delta_S + \delta_S \sqrt{1 - \frac{2^{\delta_S}}{\delta_S r}} \leq 2\delta_S.$$

In particular, this implies that if $\delta_S < \frac{1}{2}k^*$ (equivalently, if $\delta < k^*$), then the minimum runtime is

$$\Theta\left(\operatorname{Vol}(H) 2^{\delta} \log_2(r)\right). \qquad \blacksquare$$

We note that the gap between upper and lower runtime bounds for TTSV1-Gen and TTSV2-Gen is on the order of r/δ and $(r/\delta)^2$, respectively. This is, in a sense, the best possible gap, as it represents the ratio between the maximum and minimum second and third moments, respectively, of edge sizes for a fixed average size. In practice, these moments govern the runtime for these algorithms up to a multiplicative function based on the max edge size r.

4. The hypergraph adjacency tensor in practice. We explain how to use our TTSV algorithms to perform fundamental hypergraph analyses, which we then apply to the datasets listed and summarized in Table 1. In particular, we focus on algorithms for computing tensor eigenvector centralities, as well as a CP-decomposition approach for performing tensor-based hypergraph clustering. Our study here is meant to be illustrative rather than exhaustive: TTSV algorithms find far-ranging application in tensor computation beyond these two tasks. While our goal is not to argue that the specific centrality and clustering algorithms we propose are the best of their type, we show they provide complementary information to graph reduction approaches performed on the clique expansion, and provide a concrete example to illustrate that they detect subtle, higher-order structure in hypergraphs that a wide variety of popular,

Table 1
Summary statistics of datasets.

Dataset	Ref.	$\lvert V\rvert$	$\lvert E\rvert$	δ	Vol(H)	r
`mathoverflow`	[51]	73.8K	5.4K	24.2	132K	1.8K
filtered		39.8K	5.2K	11.3	58.8K	100
`MAG-10`	[3, 49]	80.2K	51.9K	3.5	181K	25
`DAWN`	[3]	2.1K	87.1K	3.9	343K	22
`cooking`	[3]	6.7K	39.8K	10.8	429K	65
`walmart-trips`	[3]	88.9K	69.9K	6.6	461K	25
`trivago-clicks`	[18]	173K	233K	3.1	726K	86
`stackoverflow`	[51]	15.2M	1.1M	23.7	26.1M	61.3K
filtered		7.8M	1.0M	10.1	10M	76
`amazon-reviews`	[41]	2.3M	4.3M	17.1	73.1M	9.4K
filtered		2.2M	3.7M	9.6	35.6M	27

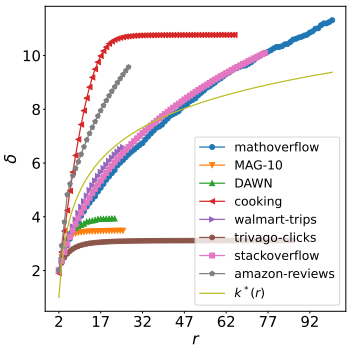


Figure 2. *δ versus r.*

existing matrix-based methods provably cannot. Taken together, this suggests that the tensor approaches enabled by our algorithms are flexible, tractable, and worthwhile avenues for nonuniform hypergraph data analysis. Before proceeding, we compare the empirical runtimes of the aforementioned TTSV algorithm variants. All experiments were run on a single core of a MacBook Pro with an M1 Max processor and 32 GB of RAM.

4.1. Timing experiments. We compare the empirical runtimes of TTSV1-Gen and TTSV1-Unord against two baselines: *explicit,* in which $\mathcal{A}$ is constructed explicitly and TTSV1 is computed from definition, and *ordered,* which is identical to TTSV1-Unord except that it iterates over full blowup sets $\beta(e)$ rather than $\kappa(e)$. To study scaling in hypergraph rank, we increase r from 2 up to a maximum of 100 or the largest hyperedge size,[2] or until the execution timeout of one hour. This procedure corresponds to the "less than or equal to" (LEQ) filtering discussed in [35]. With this scheme, we are able to process `MAG-10`, `DAWN`, `walmart-trips`, `cooking`, and `trivago-clicks` in their unfiltered entirety, `mathoverflow` until $r = 100$, to guard against floating-point errors, and `stackoverflow` and `amazon-reviews` until timeout at $r = 76$ and $r = 27$, respectively.

Figure 3 presents the TTSV1 results for all datasets in Table 1. Similar results for TTSV2 are reported in Figure SM1 of the supplementary material. Missing times indicate the algorithm either timed out after an hour or forced an out-of-memory error. For all datasets, the explicit algorithm becomes intractable after $r > 3$. The separation between the green and blue lines attests to the significant speedup from considering only unordered blowups. Furthermore, TTSV1-Gen provides additional speedup many orders of magnitude over TTSV1-Unord, even for modest values of r. For example, in `mathoverflow`, the speedup at $r = 9$ is about an order of magnitude and increases to about three orders of magnitude by $r = 18$.[3]

[2]See section SM2 of the supplementary material for further discussion of numerical considerations.

[3]Our implementation utilizes two heuristic approaches, which, while not affecting the asymptotic analysis, result in significant performance gains: for sufficiently small r we use a direct convolution approach instead of the FFT [27, 52], and for edges of size r we use a direct approach identical to the uniform hypergraph case [7].

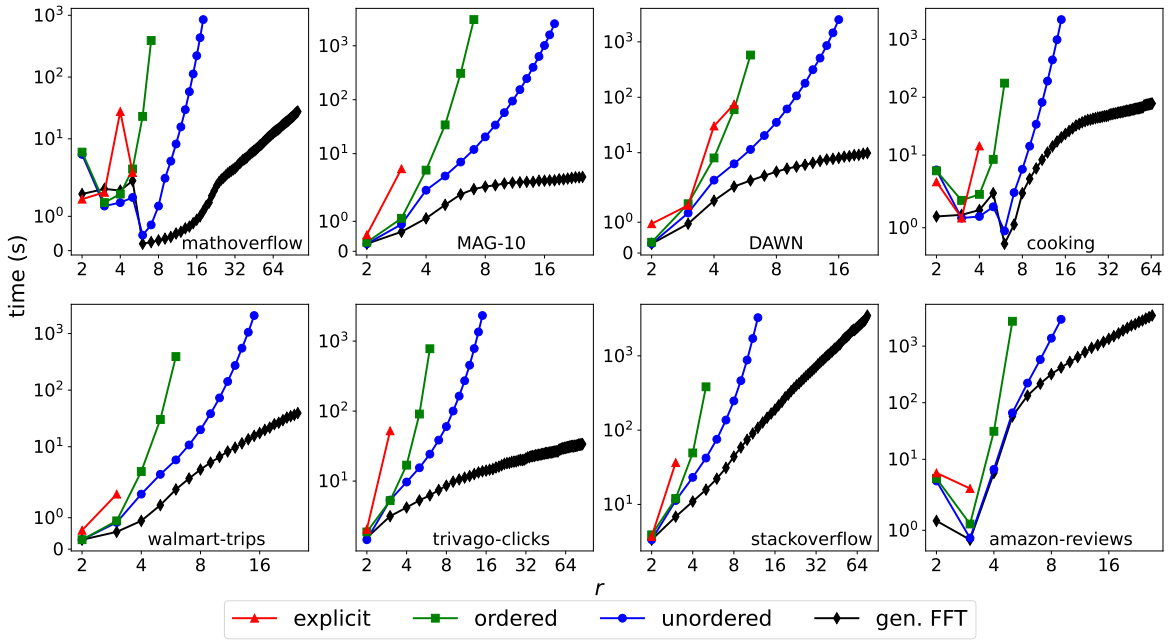


Figure 3. *Runtimes of TTSV1 algorithms.*

Having established the superiority of TTSV1-UNORD, we study its performance on the hyperedge level as a function of hyperedge size and r. We select `DAWN` and `cooking`, as their hyperedge distributions differ substantially. Figure 4 presents timing results for TTSV1-GEN, as well as the hyperedge size distributions. For `DAWN`, the bulk of the compute time for most values of r is spent on edges with sizes $3 \leq |e| \leq 6$, yet the per-hyperedge runtime for edges in this range is relatively fast. These observations are reconciled by the fact that the edge degree distribution is right-skewed. For `cooking`, the edge size distribution is even more right-skewed, but also has a mode at $|e| = 9$, exceeding that of `DAWN`, which is 3. As a result, most of the runtime is concentrated around edges of size $8 \leq |e| \leq 17$. However, the per-edge runtime is still dominated by the larger edges for higher values of r. Note that many edge sizes beyond 36 are not present in the dataset, yielding the white striations in the runtime plots.

4.2. Centrality. Our TTSV algorithms may be utilized to compute nonlinear hypergraph centralities. Here, we discuss several such centrality measures in the nonuniform setting, apply the relevant Perron–Frobenius theory guaranteeing their existence, and then compute them[4] on data and illustrative toy examples to show they yield meaningful, complementary information to existing centrality measures. We focus on the tensor-based Z-eigenvector centrality (ZEC) and H-eigenvector centrality (HEC) introduced by Benson [8] for *uniform* hypergraphs. Here, the centrality c_u of node u depends on the sum of products of its neighbors' centralities. Using the formalism of hyperedge blowups, we note that this same concept may be applied to nonuniform hypergraphs where ZEC and HEC satisfy

[4]Our centrality code is available at https://github.com/pnnl/GENTTSV.

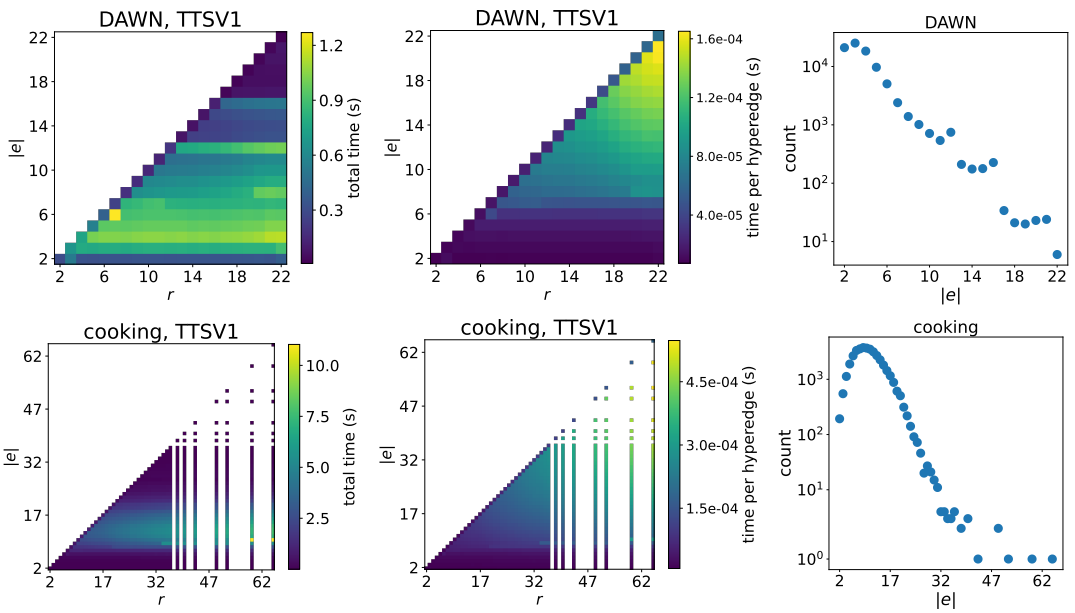


Figure 4. *Total (left) and per-edge (middle) runtimes for hyperedge size $|e|$ as a function of rank r, and hyperedge size distributions (right) for* `DAWN` *and* `cooking`.

$$\sum_{e\in E(u)}\sum_{x\in\beta(e)}\left[\frac{1}{c_u}\prod_{v\in x}c_v\right]=\lambda c_u,$$
$$\sum_{e\in E(u)}\sum_{x\in\beta(e)}\left[\frac{1}{c_u}\prod_{v\in x}c_v\right]=\lambda c_u^{r-1},$$

respectively. Note that these differ in that the second preserves dimensionality on both sides of the equation. Both of these problems can be cast as eigenvector problems whose existence, as we soon show, is guaranteed if the hypergraph is connected.

Definition 4.1 (Z- and H-eigenvector centrality (ZEC and HEC)). *Let H be a connected, rank r hypergraph with adjacency tensor $\mathcal{A}$. The Z- and H-eigenvector centrality of H is given by a vector $\mathbf{c}$ with $||\mathbf{c}||_1=1$ satisfying*

$$\mathcal{A}\mathbf{c}^{r-1}=\lambda\mathbf{c},$$
$$\mathcal{A}\mathbf{c}^{r-1}=\lambda\mathbf{c}^{[r-1]},$$

respectively, for some eigenvalue $\lambda>0$.

While the basic pattern of nonlinearity afforded by these methods is the same in the uniform and nonuniform hypergraph settings, the intuition behind these measures differs slightly. For ZEC in the uniform setting, the centrality of a node is proportional to the sum of products of the centralities of all nodes in edges that contain it. In the nonuniform setting, however, the importance of a node derives from multiplying the centralities of other nodes within all possible *blowups* of the containing hyperedges. Despite this apparent "unevenness" in how node importance is computed in the product, considering all possible blowups ensures there is no

artificial bias to any node within the hyperedge. HEC proceeds similarly, except the polynomials on both sides of the equation have the same degree for every term—a desirable property whenever dimensionality preservation is important, such as in physics-based applications.

The Perron–Frobenius theory guarantees the existence of the Z- and H-eigenvectors for the nonuniform hypergraph adjacency tensor $\mathcal{A}$ as follows: recall, from [45], that an order k, n-dimensional tensor $\mathcal{X}$ is *irreducible* if the associated directed graph (V, E) with $V = \{1, \ldots, n\}$ and

$$E = \{(i, j) : \text{ there exists } I = \{i_2, \ldots, i_n\} \text{ with } j \in I \text{ and } \mathcal{X}_{ii_2 \ldots i_k} \neq 0\},$$

is strongly connected. Via the symmetry of $\mathcal{A}$, it easily follows that its associated digraph is precisely the hypergraph's clique expansion, in which every directed edge (i, j) is reciprocated by an edge (j, i). Consequently, defining a connected hypergraph as one whose clique expansion is connected, the Perron–Frobenius theorem for nonnegative tensors [45, Theorem 3.11, p. 50] is applicable to connected, nonuniform hypergraphs as follows.

Theorem 4.2 (Perron–Frobenius theorem for the hypergraph adjacency tensor [45]). *If H is a connected, rank r hypergraph with adjacency tensor $\mathcal{A}$, then there exist*

- *Z-eigenpair $\lambda > 0, \mathbf{x} > 0$ satisfying $\mathcal{A}\mathbf{x}^{r-1} = \lambda \mathbf{x}$, and*
- *H-eigenpair $\lambda > 0, \mathbf{x} > 0$ satisfying $\mathcal{A}\mathbf{x}^{r-1} = \lambda \mathbf{x}^{[r-1]}$, where λ is the largest H-eigenvalue of $\mathcal{A}$ and $\mathbf{x}$ is unique up to scaling.*

Observe here that uniqueness up to scaling is only guaranteed for H-eigenpairs, the theory of which is generally stronger than that for the Z-counterparts. Having established the necessary theory, we now provide implementation details. ZEC is computed using a dynamical system approach proposed by Benson and Gleich in [9]. Specifically, we have

$$[\mathcal{A}\mathbf{x}^{r-2}]\mathbf{x} = \lambda \mathbf{x} \iff \mathcal{A}\mathbf{x}^{r-1} = \mathbf{x},$$

which is the same as requiring that the system

$$\frac{dx}{dt} = \Lambda(\mathcal{A}\mathbf{x}^{r-2}) - \mathbf{x}$$

BG: Z-eigenvector centrality

Data: n-vertex, rank r hypergraph H, tolerance τ, step size h
Result: Z-eigenvector centrality, $\mathbf{x}$
$\mathbf{y} = \frac{1}{n} \cdot \mathbf{1}$
repeat
  $\mathbf{Y} = \text{TTSV2}(H, \mathbf{y})$
  $\mathbf{d}$ = dominant eigenvector of $\mathbf{Y}$
  $\mathbf{x} = \mathbf{y} + h \cdot (\mathbf{d} - \mathbf{y})$
  $\mathbf{g} = \mathbf{x} \oslash \mathbf{y}$
  $\mathbf{y} = \mathbf{x}$
until $\frac{\max \mathbf{g} - \min \mathbf{g}}{\min \mathbf{g}} < \tau$;
return $\mathbf{x}$

NQZ: H-eigenvector centrality

Data: n-vertex, rank r hypergraph H, tolerance τ
Result: H-eigenvector centrality, $\mathbf{x}$
$\mathbf{y} = \frac{1}{n} \cdot \mathbf{1}$
$\mathbf{z} = \text{TTSV1}(H, \mathbf{y})$
repeat
  $\mathbf{x} = \mathbf{z}^{[\frac{1}{r-1}]} / \left\| \mathbf{z}^{[\frac{1}{r-1}]} \right\|_1$
  $\mathbf{z} = \text{TTSV1}(H, \mathbf{x})$
  $\lambda_{\min} = \min (\mathbf{z} \oslash \mathbf{x}^{[r-1]})$
  $\lambda_{\max} = \max (\mathbf{z} \oslash \mathbf{x}^{[r-1]})$
until $(\lambda_{\max} - \lambda_{\min})/\lambda_{\min} < \tau$;
return $\mathbf{x}$

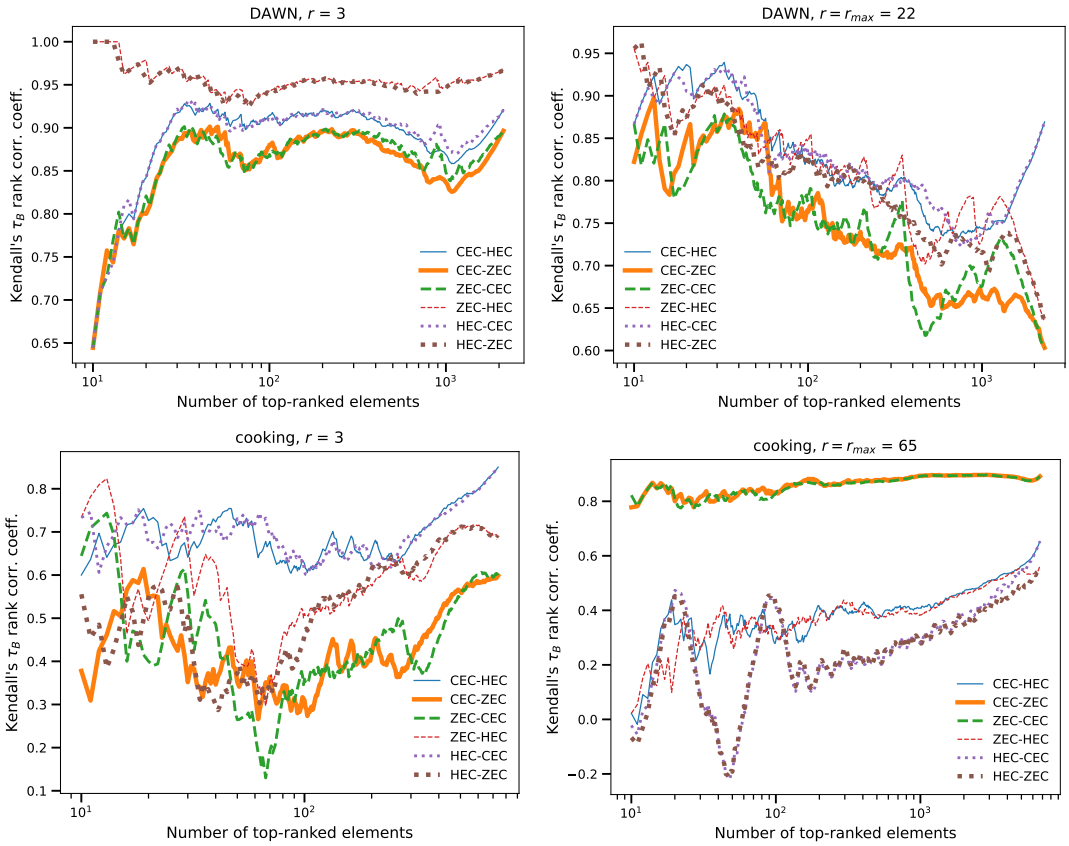


Figure 5. *Kendall τ_B rank correlation coefficient of ZEC, HEC, and CEC for the top k nodes.*

have a steady state solution, where Λ maps a matrix to its dominant eigenvector. Consequently, any forward integration scheme can be applied together with an eigenvector solver to get a tensor eigenvector through a nonlinear matrix eigenvector problem. BG presents a concrete instantiation of the Benson–Gleich approach, which utilizes TTSV2 as a subroutine. For HEC, we use a power-iteration like method along the lines of Ng, Qi, and Zhou [40], presented in NQZ, which relies on the calculation of TTSV1 as a subroutine.

Applying these algorithms to `DAWN` and `cooking`, we now address three questions: (1) do our nonuniform hypergraph centrality measures provide different centrality scores in practice from existing methods?, (2) how do these rankings change as we vary the maximum hyperedge size r?, and (3) can these measures detect higher-order structure in hypergraphs that is inexpressible in graphs and therefore undetectable by clique expansion and associated approaches? Section SM3 of the supplementary material presents additional case-study for `cooking` and `DAWN`.

4.2.1. Tensor centralities provide complementary information. We compare ZEC and HEC scores against each other, and against clique expansion centrality (CEC): the dominant eigenvector of the weighted clique expansion adjacency matrix [8]. We compare the ordinal rankings induced by ZEC, HEC, and CEC by computing Kendall's τ_B rank correlation coefficient among the top k ranked vertices. Figure 5 presents the results for `DAWN` and `cooking` for

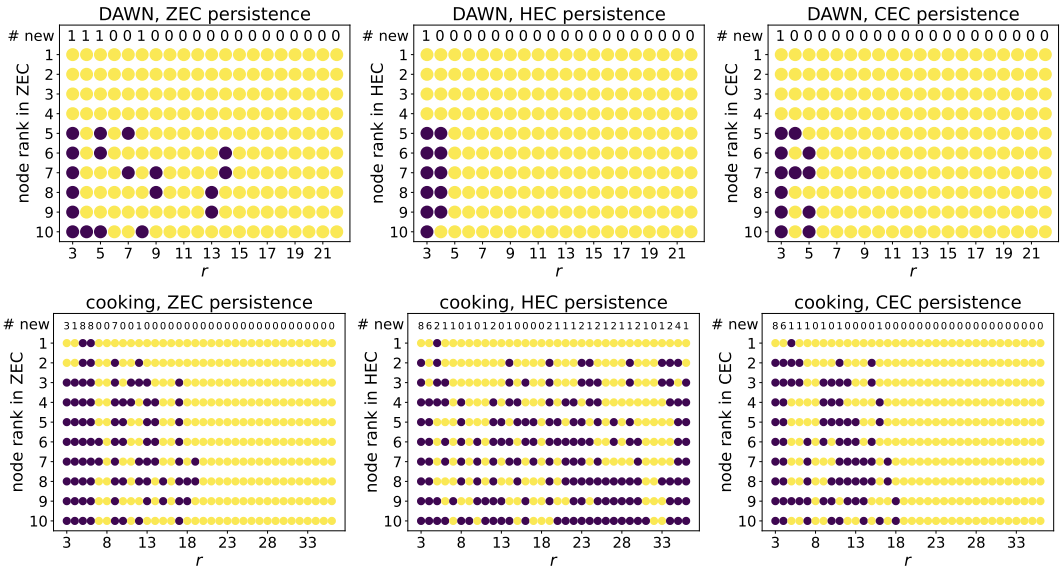
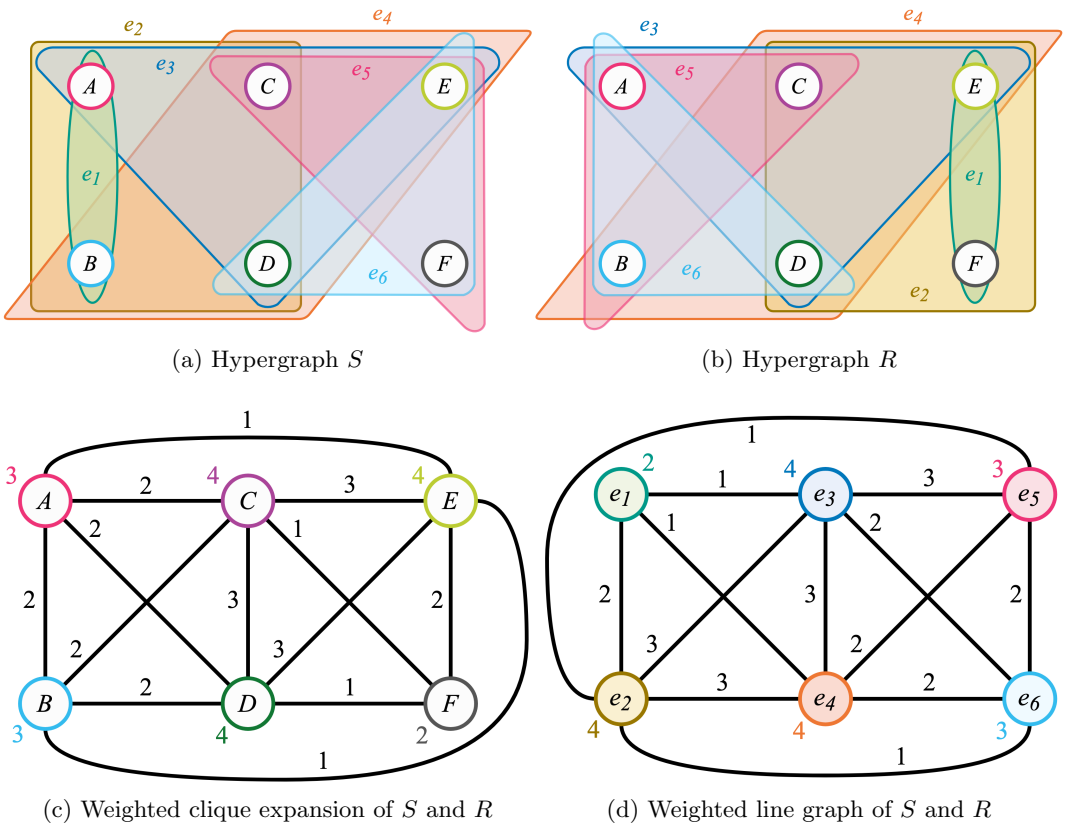


Figure 6. *Node rank persistence of the top* 10 *nodes under ZEC, HEC, and CEC for* `DAWN` *and* `cooking`*. Purple indicates a rank change from* $r-1$*, and yellow indicates no change.*

both the full, unfiltered data ($r = 22$ and 65, respectively), as well as for the $r = 3$ filtering. We observe that CEC rankings are relatively uncorrelated with those of HEC and ZEC, and that no pair of measures exhibits a consistent level of correlation among the top k ranked vertices. When compared against each other, ZEC and HEC are either weakly correlated or uncorrelated, suggesting they provide different information in practice. Lastly, the differences between $r = 3$ and the unfiltered data suggest these correlations are sensitive to filtering.

4.2.2. Persistence in tensor centrality. Next, we study the persistence of rankings induced by ZEC, HEC, and CEC. In particular, we perform an LEQ filtering sweep for r ranging from 2 to the maximum hyperedge size, and record whether the rank of each of the top 10 nodes changes as we increase r. Figure 6 presents the results for `DAWN` and `cooking`, where purple indicates rank change from $r-1$, yellow indicates no change, and the top row lists the number of new nodes in the top 10. For `DAWN`, CEC and HEC rankings quickly stabilize, showing no changes after $r = 5$ and 4, respectively. In contrast, ZEC rankings stabilize more slowly at $r = 14$. For all three centrality measures, however, stabilization occurs before the maximum hyperedge size at $r = 22$, echoing the claim in [8] that higher-order information is sometimes well captured by hyperedges that are "medium" to "small" relative to the largest hyperedge. Consequently, for larger data (such as `mathoverflow`) where the maximum hyperedge size is prohibitively large for our algorithms, analyses may still be satisfactorily performed on a filtering to smaller hyperedges. This is, of course, dataset and question-dependent, as evidenced by the persistence results for `cooking`: here, HEC rankings continue to show instability across larger r, whereas ZEC and CEC rankings both stabilize around $r = 18$. This highlights how these tensor centralities can differ from matrix analogues as well as from each other.

4.2.3. Tensor centrality distinguishes Gram mates. Having shown that the tensor-based ZEC and HEC provide different information than the matrix-based CEC, we now investigate whether ZEC and HEC capture higher-order structure that is *inexpressible* by the hypergraph's



(a) Hypergraph S

(b) Hypergraph R

(c) Weighted clique expansion of S and R

(d) Weighted line graph of S and R

Figure 7. *Two nonisomorphic hypergraphs with identical weighted clique expansions and identical line graphs.*

clique expansion graph. To address this more nuanced question, we analyze highly structured families of hypergraphs called *Gram mates* [30, 31]. Gram mates are pairs of hypergraphs having incidence matrices S and R satisfying

$$SS^T = RR^T,$$
$$S^T S = R^T R.$$

Interpreted combinatorially, $SS^T = RR^T$ means the codegree of any pair of vertices in S is the same as that in R, thereby yielding identical weighted clique expansions. Similarly, $S^T S = R^T R$ guarantees each pair of hyperedges has the same intersection cardinality in one hypergraph as in the other, meaning their weighted line graphs are identical. Figure 7 presents a small example derived from [30, 39] of nonisomorphic Gram mate hypergraphs alongside their weighted clique expansion and line graphs. We emphasize that many existing hypergraph measures and matrices cannot distinguish between these two hypergraphs. For example:

- The singular values of the incidence matrices S and R.
- Bolla [13] and Rodriguez's [46] hypergraph Laplacian, Cardoso's signless Laplacian [15], the hypergraph adjacency matrix [14, 37], and s-line graphs [2].
- Gibson's dynamical system for categorical data and hypergraph clustering [23].
- The hypergraph core/periphery, structural equivalence, and centrality methods derived from the "dual-projection" approach advocated for in [22].

- Bipartite projection based analyses, such as bipartite modularity [5].

In contrast to the above, ZEC and HEC do distinguish between the two hypergraphs in Figure 7. In particular, letting $\mathbf{x}_S$ and $\mathbf{x}_R$ denote the centrality vectors for either ZEC or HEC applied to hypergraphs S and R, we have that

$$\begin{aligned}\mathbf{x}_S(u) &= \mathbf{x}_R(u) \text{ for } u = C, D,\\ \mathbf{x}_S(u) &> \mathbf{x}_R(u) \text{ for } u = E, F,\\ \mathbf{x}_S(u) &< \mathbf{x}_R(u) \text{ for } u = A, B.\end{aligned}$$

4.3. Clustering. Our TTSV algorithms also enable computation of hypergraph tensor embeddings, which may then serve as features for many clustering algorithms, such as k-means. Following this approach, we aim to embed the hypergraph adjacency tensor $\boldsymbol{\mathcal{A}}$ in $\mathbb{R}^{n\times q}$, where q is the target embedding dimension, so that each node is represented by a q-dimensional vector. We perform the embedding by finding a symmetric CP-decomposition [26, 32] of $\boldsymbol{\mathcal{A}}$, meaning we seek an $n \times q$ matrix $\mathbf{E}$ and a vector $\boldsymbol{\lambda} \in \mathbb{R}^q$ such that the tensor norm given by

$$f(\boldsymbol{\lambda}, \mathbf{E}) = ||\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{X}}|| \text{ with } \boldsymbol{\mathcal{X}} = \sum_{j=1}^{q} \boldsymbol{\lambda}_j \mathbf{E}_j^{\otimes r} \tag{4.1}$$

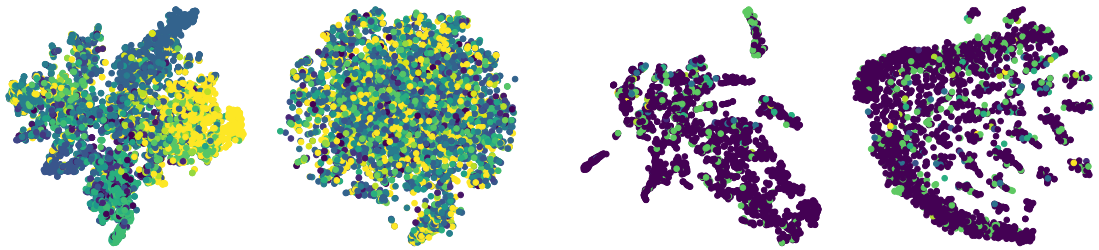
is minimized, where $\mathbf{E} = [\mathbf{E}_1 \mathbf{E}_2 \cdots \mathbf{E}_q]$ and $\mathbf{E}_j^{\otimes r}$ is the r-way tensor outer product of $\mathbf{E}_j$ with itself. To optimize (4.1), we employ a standard first-order optimization scheme and utilize the closed-form expressions [32] for the gradients

$$\begin{aligned}\frac{\partial f}{\partial \boldsymbol{\lambda}_j} &= -2\left[\boldsymbol{\mathcal{A}}\mathbf{E}_j^r - \sum_{k=1}^{q} \boldsymbol{\lambda}_k \langle \mathbf{E}_j, \mathbf{E}_k\rangle^r\right],\\ \frac{\partial f}{\partial \mathbf{E}_j} &= -2d\boldsymbol{\lambda}_j\left[\boldsymbol{\mathcal{A}}\mathbf{E}_j^{r-1} - \sum_{k=1}^{q} \boldsymbol{\lambda}_k \langle \mathbf{E}_j, \mathbf{E}_k\rangle^{r-1}\mathbf{E}_k\right],\end{aligned}$$

where $\langle \mathbf{E}_j, \mathbf{E}_k\rangle = \mathbf{E}_j^T \mathbf{E}_k$, $\boldsymbol{\mathcal{A}}\mathbf{E}_j^r = \sum_{i_1=1}^{n} \cdots \sum_{i_r=1}^{n} \boldsymbol{\mathcal{A}}_{i_1,\ldots,i_r} \prod_{k=1}^{r} \mathbf{E}_{j_{i_k}}$ is the TTSV operation which results in a scalar. Note that the TTSV value is obtained from the TTSV1 vector simply by taking an inner product with $\mathbf{E}_j$. In the first order scheme, computing f and its derivatives explicitly requires $O(qn^r)$ time, but we use TTSV1-Gen, together with the gradient computation approach outlined in [48], to cut this time down to $O(\text{TTSV1} + nq^2)$, where $O(\text{TTSV1})$ is the worst-case runtime of TTSV1-Gen. After obtaining this CP decomposition for $\boldsymbol{\mathcal{A}}$, the resulting embedding $E \in \mathbb{R}^{n\times q}$ may be used as features for a standard k-means clustering algorithm [24] or more generally within any metric-space clustering framework. Instead of clustering $\boldsymbol{\mathcal{A}}$ directly, we cluster the corresponding normalized Laplacian tensor $\boldsymbol{\mathcal{L}}$ from [6], given by

$$\boldsymbol{\mathcal{L}}_{p_1\ldots p_r} = \begin{cases} -\left[\prod_{j=1}^{r} d(v_{p_j})^{-1/r}\right] \frac{|e|}{|\beta(e)|} & \text{if } p_1 \ldots p_r \in \beta(e),\\ 1 & \text{if } p_1 = p_2 = \cdots = p_r,\\ 0 & \text{otherwise.}\end{cases}$$

It is worth noting that $\boldsymbol{\mathcal{L}}$ does not equally weight all blowups of an edge, and so TTSV1-Gen cannot be directly applied. However, if $\mathbf{d}$ is the vector of all degrees, we have that $\boldsymbol{\mathcal{L}}\mathbf{x}^r = \boldsymbol{\mathcal{I}}\mathbf{x}^r - \boldsymbol{\mathcal{A}}(\mathbf{d}^{[-1/r]} \odot \mathbf{x})^r$ and, more generally, $\boldsymbol{\mathcal{L}}\mathbf{x}^{r-k} = \boldsymbol{\mathcal{I}}\mathbf{x}^{r-1} - (\mathbf{d}^{[-1/r]})^{\otimes k} \odot \boldsymbol{\mathcal{A}}(\mathbf{d}^{[-1/r]} \odot \mathbf{x})^{r-k}$, where $\odot$ is the appropriate elementwise (Hadamard) tensor product.



(a) t-SNE of embedding for the normalized Laplacian of the clique expansion (left) vs. normalized Laplacian tensor of the hypergraph (right) of `cooking`

(b) t-SNE of embedding for the normalized Laplacian of the clique expansion (left) vs. normalized Laplacian tensor of the hypergraph (right) for `DAWN`

Figure 8. *Comparison of matrix* [55] *to tensor embedding of* `cooking` *(left) and* `DAWN` *(right).*

We now apply the aforementioned clustering approach to `cooking` and `DAWN`. To better reveal clusters, we filter out high-degree nodes that appear in more than 20% of the hyperedges. We color nodes based on hyperedge type metadata, assigning each node to the majority color of hyperedge it appears in. To better reveal node colors and speed up the computations, we also filter out hyperedges above size $r = 8$. Figures 8a and 8b present t-SNE [50] plots visualizing embeddings of these datasets. For each dataset, the left visualization presents the matrix-based embedding using the normalized Laplacian [19] of the hypergraph's clique expansion graph,[5] while the right shows the normalized Laplacian tensor embedding of the hypergraph. For both datasets we observe starkly different geometry between the t-SNE representations of the matrix and tensor embeddings, providing qualitative evidence that these two approaches are capturing different features of the hypergraph cluster in practice.[6]

5. Conclusion and future work. We developed a suite of algorithms for performing fundamental tensor operations on the nonuniform hypergraph adjacency tensor. Improving upon approaches that are intractable in time and space complexity, we developed efficient, implicit methods tailored to exploit the nuanced symmetry of the adjacency tensor. We then demonstrated how these algorithms give rise to fundamental tensor-based hypergraph analyses, such as centrality and clustering, which hold promise in capturing hypergraph-native structure over existing matrix-based approaches. Our exploration here is not comprehensive, and many avenues remain for future work. First, we note that the hypergraph adjacency tensor we utilized is defined for simple, unweighted, nonuniform hypergraphs. Real data may present multiple hyperedges, weights, vertex multiplicities within a hyperedge, or directionality. Extending our methods to accommodate such cases in a principled manner would be advantageous. Second, our application of TTSV algorithms to perform hypergraph analyses is cursory and leaves a number of exciting possibilities to future work: how might one develop multilinear tensor PageRank for nonuniform hypergraphs, or use our algorithms in supervised and semisupervised machine learning problems, such as node classification and link predic-

[5] We also used this embedding to initialize the iterative scheme to obtain the tensor embedding.

[6] No effort was made to tune the performance of either clustering algorithm or evaluate which clustering is "better." Indeed, understanding which hypergraph structural features are highlighted by the CP decomposition of $\boldsymbol{\mathcal{L}}$ is a compelling open question for future work. In particular, the effectiveness of spectral clustering is explainable in part by the tight connection between Laplacian spectra and the combinatorial properties of the graph captured by the Cheeger inequality. The authors are unaware of any similar known results for the nonuniform adjacency tensor (Laplacian or otherwise) associated with a hypergraph.

tion? Furthermore, it seems plausible that our generating function approach to exploiting symmetry in the hypergraph adjacency tensor could be extended to other tensor operations: for instance, the Tucker decomposition involves repeatedly performing a tensor times same matrix operation [21, 28]. Lastly, despite the current approach being tailored to hypergraphs, we believe that generating-function-based tensor algorithms similar to the ones we presented may have utility in general symmetric tensor problems beyond the context of hypergraphs.

Acknowledgments. We thank Tammy Kolda, Jiajia Li, and Shruti Shivakumar for helpful discussions on tensor decompositions, and Yosuke Mizutani for his assistance in drawing the hypergraphs visualized in Figure 7. We would also like to thank the anonymous referee who pointed out the alternative derivation of the generating function expression for TTSVk based on the relationship $\nabla^k \boldsymbol{\mathcal{X}}\mathbf{b}^r = \frac{(r-k)!}{r!}\boldsymbol{\mathcal{X}}\mathbf{b}^{r-k}$.

REFERENCES

[1] S. Agarwal, K. Branson, and S. Belongie, *Higher order learning with graphs*, in Proceedings of the 23rd International Conference on Machine Learning, 2006, pp. 17–24.
[2] S. G. Aksoy, C. Joslyn, C. O. Marrero, B. Praggastis, and E. Purvine, *Hypernetwork science via high-order hypergraph walks*, EPJ Data Sci., 9 (2020), 16.
[3] I. Amburg, N. Veldt, and A. Benson, *Clustering in graphs and hypergraphs with categorical edge labels*, in Proceedings of the Web Conference 2020, 2020, pp. 706–717.
[4] A. Antelmi, G. Cordasco, B. Kamiński, P. Prałat, V. Scarano, C. Spagnuolo, and P. Szufel, *Simplehypergraphs.jl—novel software framework for modelling and analysis of hypergraphs*, in Algorithms and Models for the Web Graph: 16th International Workshop, WAW 2019, Brisbane, QLD, Australia, 2019, Proceedings 16, Springer, 2019, pp. 115–129.
[5] R. Arthur, *Modularity and projection of bipartite networks*, Phys. A, 549 (2020), 124341.
[6] A. Banerjee, A. Char, and B. Mondal, *Spectra of general hypergraphs*, Linear Algebra Appl., 518 (2017), pp. 14–30, https://doi.org/10.1016/j.laa.2016.12.022.
[7] A. Benson and J. Kleinberg, *Link prediction in networks with core-fringe data*, in WWW'19, The World Wide Web Conference, ACM, 2019, pp. 94–104.
[8] A. R. Benson, *Three hypergraph eigenvector centralities*, SIAM J. Math. Data Sci., 1 (2019), pp. 293–312, https://doi.org/10.1137/18M1203031.
[9] A. R. Benson and D. F. Gleich, *Computing tensor Z-eigenvectors with dynamical systems*, SIAM J. Matrix Anal. Appl., 40 (2019), pp. 1311–1324, https://doi.org/10.1137/18M1229584.
[10] A. R. Benson, D. F. Gleich, and J. Leskovec, *Tensor spectral clustering for partitioning higher-order network structures*, in Proceedings of the 2015 SIAM International Conference on Data Mining, SIAM, 2015, pp. 118–126, https://doi.org/10.1137/1.9781611974010.14.
[11] A. R. Benson, D. F. Gleich, and L.-H. Lim, *The spacey random walk: A stochastic process for higher-order data*, SIAM Rev., 59 (2017), pp. 321–345, https://doi.org/10.1137/16M1074023.
[12] C. Berge, *Hypergraphs: Combinatorics of Finite Sets*, North-Holland Math. Library 45, Elsevier, 1984.
[13] M. Bolla, *Spectra, Euclidean representations and clusterings of hypergraphs*, Discrete Math., 117 (1993), pp. 19–39.
[14] K. Cardoso, R. Del-Vecchio, L. Portugal, and V. Trevisan, *Adjacency energy of hypergraphs*, Linear Algebra Appl., 648 (2022), pp. 181–204.
[15] K. Cardoso and V. Trevisan, *The signless Laplacian matrix of hypergraphs*, Spec. Matrices, 10 (2022), pp. 327–342.
[16] T. Carletti, F. Battiston, G. Cencetti, and D. Fanelli, *Random walks on hypergraphs*, Phys. Rev. E, 101 (2020), 022308.
[17] U. Chitra and B. Raphael, *Random walks on hypergraphs with edge-dependent vertex weights*, in International Conference on Machine Learning, PMLR, 2019, pp. 1172–1181.
[18] P. S. Chodrow, N. Veldt, and A. R. Benson, *Generative hypergraph clustering: From blockmodels to modularity*, Sci. Adv., 7 (2021), eabh1303, https://doi.org/10.1126/sciadv.abh1303.

[19] F. R. Chung, *Spectral Graph Theory*, CBMS Regional Conf. Ser. in Math. 92, AMS, 1997.
[20] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.
[21] L. De Lathauwer, B. De Moor, and J. Vandewalle, *On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342, https://doi.org/10.1137/S0895479898346995.
[22] M. G. Everett and S. P. Borgatti, *The dual-projection approach for two-mode networks*, Soc. Netw., 35 (2013), pp. 204–210.
[23] D. Gibson, J. Kleinberg, and P. Raghavan, *Clustering categorical data: An approach based on dynamical systems*, VLDB J., 8 (2000), pp. 222–236.
[24] J. A. Hartigan and M. A. Wong, *Algorithm AS 136: A k-means clustering algorithm*, J. R. Stat. Soc. Ser. C Appl. Statist., 28 (1979), pp. 100–108.
[25] K. Hayashi, S. G. Aksoy, C. H. Park, and H. Park, *Hypergraph random walks, Laplacians, and clustering*, in Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020, pp. 495–504.
[26] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Phys., 6 (1927), pp. 164–189.
[27] L. Ismail, *Convolution on top of the IBM cell BE processor*, in PDPTA, 2009, pp. 507–611.
[28] R. Jin, *Scalable symmetric Tucker tensor decomposition*, in Fourteenth International Conference on Sampling Theory and Applications, 2023, https://openreview.net/forum?id=17iCMIE0V9.
[29] Z. T. Ke, F. Shi, and D. Xia, *Community Detection for Hypergraph Networks via Regularized Tensor Power Iteration*, https://arxiv.org/abs/1909.06503v2, 2020.
[30] S. Kim and S. Kirkland, *Gram mates, sign changes in singular values, and isomorphism*, Linear Algebra Appl., 644 (2022), pp. 108–148.
[31] S. Kirkland, *Two-mode networks exhibiting data loss*, J. Complex Netw., 6 (2018), pp. 297–316.
[32] T. G. Kolda, *Numerical optimization for symmetric tensor decomposition*, Math. Program., 151 (2015), pp. 225–248.
[33] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500, https://doi.org/10.1137/07070111X.
[34] T. G. Kolda and J. R. Mayo, *Shifted power method for computing tensor eigenpairs*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1095–1124, https://doi.org/10.1137/100801482.
[35] N. W. Landry, I. Amburg, M. Shi, and S. G. Aksoy, *Filtering higher-order datasets*, J. Phys. Complexity, 5 (2024), 015006.
[36] N. W. Landry, M. Lucas, I. Iacopini, G. Petri, A. Schwarze, A. Patania, and L. Torres, *XGI: A Python package for higher-order interaction networks*, J. Open Source Softw., 8 (2023), 5162.
[37] W.-C. W. Li and P. Solé, *Spectra of regular graphs and hypergraphs and orthogonal polynomials*, European J. Combin., 17 (1996), pp. 461–477.
[38] Q. F. Lotito, M. Contisciani, C. De Bacco, L. Di Gaetano, L. Gallo, A. Montresor, F. Musciotto, N. Ruggeri, and F. Battiston, *Hypergraphx: A library for higher-order network analysis*, J. Complex Netw., 11 (2023), cnad019.
[39] Y. Mizutani, S. Aksoy, E. Alhajjar, J. Aslam, B. S. Chrisman, S. Days-Merrill, H. Jenne, T. G. Oellerich, and E. Semrad, *Information loss in weighted hypergraph line graphs and clique expansions*, in 2023 Joint Mathematics Meetings (JMM 2023), AMS, 2023.
[40] M. Ng, L. Qi, and G. Zhou, *Finding the largest eigenvalue of a nonnegative tensor*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1090–1099, https://doi.org/10.1137/09074838X.
[41] J. Ni, J. Li, and J. McAuley, *Justifying recommendations using distantly-labeled reviews and fine-grained aspects*, in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 188–197.
[42] X. Ouvrard, J.-M. L. Goff, and S. Marchand-Maillet, *Adjacency and Tensor Representation in General Hypergraphs Part 1: e-Adjacency Tensor Uniformisation Using Homogeneous Polynomials*, https://arxiv.org/abs/1712.08189v5, 2018.
[43] J. M. Pereira, J. Kileel, and T. G. Kolda, *Tensor Moments of Gaussian Mixture Models: Theory and Applications*, https://arxiv.org/abs/2202.06930, 2022.

[44] B. Praggastis, S. Aksoy, D. Arendt, M. Bonicillo, C. Joslyn, E. Purvine, M. Shapiro, and J. Y. Yun, *HyperNetX: A Python package for modeling complex network data as hypergraphs*, J. Open Source Softw., 9 (2024), 6016, https://doi.org/10.21105/joss.06016.

[45] L. Qi and Z. Luo, *Tensor Analysis: Spectral Theory and Special Tensors*, SIAM, 2017, https://doi.org/10.1137/1.9781611974751.

[46] J. A. Rodríguez, *On the Laplacian eigenvalues and metric parameters of hypergraphs*, Linear Multilinear Algebra, 50 (2002), pp. 1–14.

[47] A. Sharma and J. Srivastava, *Joint Symmetric Tensor Decomposition for Hypergraph Embeddings*.

[48] S. Sherman and T. G. Kolda, *Estimating higher-order moments using symmetric tensor decomposition*, SIAM J. Matrix Anal. Appl., 41 (2020), pp. 1369–1387, https://doi.org/10.1137/19M1299633.

[49] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. P. Hsu, and K. Wang, *An overview of Microsoft Academic Service (MAS) and applications*, in Proceedings of the 24th International Conference on World Wide Web, ACM Press, 2015, pp. 243–246, https://doi.org/10.1145/2740908.2742839.

[50] L. Van der Maaten and G. Hinton, *Visualizing data using t-SNE*, J. Mach. Learn. Res., 9 (2008), pp. 2579–2605.

[51] N. Veldt, A. R. Benson, and J. Kleinberg, *Minimizing localized ratio cut objectives in hypergraphs*, in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press, 2020, pp. 1708–1718.

[52] P. Virtanen, et al., *SciPy* 1.0*: Fundamental algorithms for scientific computing in Python*, Nat. Methods, 17 (2020), pp. 261–272, https://doi.org/10.1038/s41592-019-0686-2.

[53] H. S. Wilf, *Generatingfunctionology*, CRC Press, 2005.

[54] Y. Zhen and J. Wang, *Community detection in general hypergraph via graph embedding*, J. Amer. Statist. Assoc., 118 (2023), pp. 1620–1629.

[55] D. Zhou, J. Huang, and B. Schölkopf, *Learning with hypergraphs: Clustering, classification, and embedding*, in Advances in Neural Information Processing Systems 19, NIPS 2006, MIT Press, 2007, pp. 1601–1608.