Journal of Global Optimization (2022) 84:229–253
https://doi.org/10.1007/s10898-022-01140-4

Several approximation algorithms for sparse best rank-1 approximation to higher-order tensors

Xianpeng Mao[1] · **Yuning Yang**[2]

Received: 14 June 2021 / Accepted: 21 January 2022 / Published online: 28 February 2022

Abstract
Sparse tensor best rank-1 approximation (BR1Approx), which is a sparsity generalization of the dense tensor BR1Approx, and is a higher-order extension of the sparse matrix BR1Approx, is one of the most important problems in sparse tensor decomposition and related problems arising from statistics and machine learning. By exploiting the multilinearity as well as the sparsity structure of the problem, four polynomial-time approximation algorithms are proposed, which are easily implemented, of low computational complexity, and can serve as initial procedures for iterative algorithms. In addition, theoretically guaranteed approximation lower bounds are derived for all the algorithms. We provide numerical experiments on synthetic and real data to illustrate the efficiency and effectiveness of the proposed algorithms; in particular, serving as initialization procedures, the approximation algorithms can help in improving the solution quality of iterative algorithms while reducing the computational time.

Keywords Tensor · Sparse · Rank-1 approximation · Approximation algorithm · Approximation bound

1 Introduction

In the big data era, people often face intrinsically multi-dimensional, multi-modal and multi-view data-sets that are too complex to be processed and analyzed by traditional data mining tools based on vectors or matrices. Higher-order tensors (hypermatrices) naturally can represent such complex data sets. Tensor decomposition tools, developed for understanding tensor-based data sets, have shown the power in various fields such as signal processing, image processing, statistics, and machine learning; see the surveys [4–6, 8, 16, 29].

In high dimensional data-sets, another structure that cannot be ignored is the sparsity. Sparsity tensor examples come from clustering problems, online advertising, web link analysis, ranking of authors based on citations [17, 24, 26, 31, 32], and so on. Therefore, recent

✉ Yuning Yang
yyang@gxu.edu.cn

1 School of Physical Science and Technology, Guangxi University, Nanning 530004, China

2 College of Mathematics and Information Science, Guangxi University, Nanning 530004, China

advances incorporate sparsity into tensor decomposition models and tensor-based statistics and machine learning problems [1, 3, 21, 26, 30, 32, 35, 38]; just to name a few. As pointed out by Allen [1], introducing sparsity into tensor problems is desirable for feature selection, for compression, for statistical consistency, and for better visualization and interpretation of the data analysis results.

In (dense) tensor decomposition and related tensor models and problems, the best rank-1 approximation (BR1Approx) is one of the most important and fundamental problems [10, 28]. Just as the dense counterpart, the sparse tensor BR1Approx serves as a keystone in the computation of sparse tensor decomposition and related sparse tensor models [1, 26, 30, 32, 35]. Roughly speaking, the sparse tensor BR1Approx is to find a projection of a given data tensor onto the set of sparse rank-1 tensors in the sense of Euclidean distance. This is equivalent to maximizing a multilinear function over both unit sphere and ℓ_0 constraints; mathematical models will be detailed in Section 2. Such a problem also closely connects to the sparse tensor spectral norm defined in [30, 32].

For (dense) tensor BR1Approx, several methods have been proposed; e.g., power methods [10, 18], approximation algorithms [12, 13, 39], and convex relaxations [15, 25, 37]. For sparse matrix BR1Approx, solution methods have been studied extensively in the context of sparse PCA and sparse SVD [36, 40]; see, e.g., iterative methods [20, 36], approximation algorithms [2, 11], and semidefinite relaxation [9]; just to name a few. For sparse tensor BR1Approx, in the context of sparse tensor decomposition, Allen [1] first studied models and iterative algorithms based on ℓ_1 regularization in the literature, whereas Sun et al. [32] developed ℓ_0-based models and algorithms, and analyzed their statistical performance. Wang et al. [35] considered another ℓ_1 regularized model and algorithm, which is different from [1]. In the study of co-clustering, Papalexakis et al. [26] proposed alternating minimization methods for nonnegative sparse tensor BR1Approx.

For nonconvex and NP-hard problems, approximation algorithms are nevertheless encouraged. However, in the context of sparse tensor BR1Approx problems, little attention was paid to this type of algorithms. To fill this gap, by fully exploiting the multilinearity and sparsity of the model, we develop four polynomial-time approximation algorithms, some extending their matrix or dense tensor counterparts; in particular, the last algorithm, which is the most efficient one, is even new when reducing to the matrix or dense tensor cases. The proposed algorithms are easily implemented, and the computational complexity is not high: the most expensive execution is, if necessary, to only compute the largest singular vector pairs of certain matrices. Therefore, the algorithms are able to serve as initialization procedures for iterative algorithms. Moreover, for each algorithm, we derive theoretically guaranteed approximation lower bounds. Experiments on synthetic as well as real data show the usefulness of the introduced algorithms.

The rest of this work is organized as follows. Sect. 2 introduces the sparse tensor BR1Approx model. Approximation algorithms are presented in Sect. 3. Numerical results are illustrated in Sect. 4. Section 5 draws some conclusions.

2 Sparse tensor best rank-1 approximation

Throughout this work, vectors are written as $(\mathbf{x}, \mathbf{y}, \ldots)$, matrices correspond to $(A, B, \ldots)$, and tensors are written as $(\mathcal{A}, \mathcal{B}, \cdots)$. $\mathbb{R}^{n_1 \times \cdots \times n_d}$ denotes the space of $n_1 \times \cdots \times n_d$ real tensors. For two tensors $\mathcal{A}, \mathcal{B}$ of the same size, their inner product $\langle \mathcal{A}, \mathcal{B} \rangle$ is given by the sum of entry-wise product. The Frobenius norm of $\mathcal{A}$ is defined by $\|\mathcal{A}\|_F = \langle \mathcal{A}, \mathcal{A} \rangle^{1/2}$. $\circ$ denotes

the outer product; in particular, for $\mathbf{x}_j \in \mathbb{R}^{n_j}$, $j = 1, \ldots, d$, $\mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d$ denotes a rank-1 tensor in $\mathbb{R}^{n_1 \times \cdots \times n_d}$. $\|\mathbf{x}\|_0$ represents the number of nonzero entries of a vector $\mathbf{x}$.

Given $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ with $d \geq 3$, the tensor BR1Approx consists of finding a set of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_d$, such that

$$\min_{\lambda \in \mathbb{R}, \mathbf{x}_j \in \mathbb{R}^{n_j}, 1 \leq j \leq d} \|\lambda \cdot \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d - \mathcal{A}\|_F^2 \text{ s.t. } \|\mathbf{x}_j\| = 1,\ 1 \leq j \leq d, \tag{2.1}$$

When $\mathcal{A}$ is sparse, it may be necessary to also investigate the sparsity of the latent factors $\mathbf{x}_j$, $1 \leq j \leq d$. Assume that the true sparsity level of each latent factors is known a prior, or can be estimated; then the sparse tensor BR1Approx can be modeled as follows [32]:

$$\min_{\lambda \in \mathbb{R}, \mathbf{x}_j \in \mathbb{R}^{n_j}, 1 \leq j \leq d} \|\lambda \cdot \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d - \mathcal{A}\|_F^2 \text{ s.t. } \|\mathbf{x}_j\| = 1,\ \|\mathbf{x}_j\|_0 \leq r_j, 1 \leq j \leq d, \tag{2.2}$$

where $1 \leq r_j \leq n_j$ are positive integers standing for the sparsity level. Allen [1] and Wang et al. [35] proposed ℓ_1 regularized models for sparse tensor BR1Approx problems.

Since $\mathbf{x}_j$'s are normalized in (2.2), we have

$$\begin{aligned} \|\lambda \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d - \mathcal{A}\|_F^2 &= \|\mathcal{A}\|_F^2 - 2\lambda \langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle + \lambda^2 \prod_{j=1}^d \|\mathbf{x}_j\|^2 \\ &= \|\mathcal{A}\|_F^2 - 2\lambda \langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle + \lambda^2, \end{aligned}$$

minimizing which with respect to λ gives $\lambda = \langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle$, and so

$$\|\lambda \mathbf{x}_1 \circ \cdots \mathbf{x}_d - \mathcal{A}\|_F^2 = \|\mathcal{A}\|_F^2 - \langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle^2 .$$

Due to the multilinearity of $\langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle$, $\langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle^2$ is maximized if and only if $\langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle$ is maximized. Thus (2.2) can be equivalently recast as

$$\boxed{\max_{\mathbf{x}_j, 1 \leq j \leq d} \ \langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle \text{ s.t. } \|\mathbf{x}_j\| = 1, \|\mathbf{x}_j\|_0 \leq r_j, 1 \leq j \leq d.} \tag{2.3}$$

This is the main model to be focused on. When $r_j = n_j$, (2.3) boils down exactly to the tensor singular value problem [19]. Thus (2.2) can be regarded as a sparse tensor singular value problem.

When $r_j = n_j$, (2.3) is already NP-hard in general [14]; on the other hand, when $d = 2$ and $\mathbf{x}_1 = \mathbf{x}_2$, it is also NP-hard [22]. Therefore, we may deduce that (2.3) is also NP-hard, whose NP-hardness comes from two folds: the multilinearity of the objective function, and the sparsity constraints. In view of this, approximation algorithms for solving (2.3) are necessary.

In the rest of this work, to simplify notations, we denote

$$\mathcal{A}\mathbf{x}_1 \cdots \mathbf{x}_d := \langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle .$$

In addition, we also use the following notation to denote the partial gradients of $\mathcal{A}\mathbf{x}_1 \cdots \mathbf{x}_d$ with respect to $\mathbf{x}_j$:

$$\mathcal{A}\mathbf{x}_1 \cdots \mathbf{x}_{j-1}\mathbf{x}_{j+1} \cdots \mathbf{x}_d \ = \ \nabla_{\mathbf{x}_j} \mathcal{A}\mathbf{x}_1 \cdots \mathbf{x}_d \in \mathbb{R}^{n_j},\ 1 \leq j \leq d.$$

For example, $(\mathcal{A}\mathbf{x}_2 \cdots \mathbf{x}_d)_{i_1} = \sum_{i_2=1,\ldots,i_d=1}^{n_2,\ldots,n_d} \mathcal{A}_{i_1 i_2 \cdots i_d} \mathbf{x}_{2,i_2} \cdots \mathbf{x}_{d,i_d}$, where we write $\mathbf{x}_j := [x_{j,1}, \ldots, x_{j,n_j}]^\top$. The partial Hessian of $\mathcal{A}\mathbf{x}_1 \cdots \mathbf{x}_d$ with respect to $\mathbf{x}_{d-1}$ and $\mathbf{x}_d$ is denoted as:

$$\mathcal{A}\mathbf{x}_1 \cdots \mathbf{x}_{d-2} = \nabla_{\mathbf{x}_{d-1}, \mathbf{x}_d} \mathcal{A}\mathbf{x}_1 \cdots \mathbf{x}_d \in \mathbb{R}^{n_{d-1} \times n_d},$$

with $(\mathcal{A}\mathbf{x}_1 \cdots \mathbf{x}_{d-2})_{i_{d-1}, i_d} = \sum_{i_1=1,\ldots,i_{d-2}=1}^{n_1,\ldots,n_{d-2}} \mathcal{A}_{i_1 \cdots i_{d-1} i_d} \mathbf{x}_{1,i_1} \cdots \mathbf{x}_{d-2,i_{d-2}}$.

3 Approximation algorithms and approximation bounds

Four approximation algorithms are proposed in this section, all of which admit theoretical lower bounds. We first present some preparations used in this section. For any nonzero $\mathbf{a} \in \mathbb{R}^n$, $1 \le r \le n$, denote $[\mathbf{a}]^{\downarrow,r} \in \mathbb{R}^n$ as a truncation of $\mathbf{a}$ as

$$[\mathbf{a}]_i^{\downarrow,r} = \begin{cases} a_i, & \text{if } |a_i| \text{ is one of the } r \text{ largest (in magnitude) entries of } \mathbf{a}, \\ 0, & \text{otherwise.} \end{cases}$$

In particular, if $a_{i_1}, a_{i_2}, a_{i_3}, \ldots$ are respectively the r-, $(r+1)$-, $(r+2)$-, ... largest entries (in magnitude) with $i_1 < i_2 < i_3 < \cdots$, and $|a_{i_1}| = |a_{i_2}| = |a_{i_3}| = \cdots$, then we set $[\mathbf{a}]_{i_1}^{\downarrow,r} = a_{i_1}$ and $[\mathbf{a}]_{i_2}^{\downarrow,r} = [\mathbf{a}]_{i_3}^{\downarrow,r} = \cdots = 0$. Thus $[\mathbf{a}]^{\downarrow,r}$ is uniquely defined. We can see that $[\mathbf{a}]^{\downarrow,r}$ is a best r-approximation to $\mathbf{a}$ [20, Proposition 4.3], i.e.,

$$[\mathbf{a}]^{\downarrow,r} \in \arg\min_{\|\mathbf{x}\|_0 \le r} \|\mathbf{x} - \mathbf{a}\| \quad \Leftrightarrow \quad \frac{[\mathbf{a}]^{\downarrow,r}}{\| [\mathbf{a}]^{\downarrow,r} \|} \in \arg\max_{\|\mathbf{x}\|=1, \|\mathbf{x}\|_0 \le r} \mathbf{a}^\top \mathbf{x}. \tag{3.4}$$

It is not hard to see that the following proposition holds.

Proposition 3.1 *Let* $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{a} \neq 0$ *and let* $\mathbf{a}^0 = [\mathbf{a}]^{\downarrow,r} / \| [\mathbf{a}]^{\downarrow,r} \|$ *with* $1 \le r \le n$. *Then*

$$\langle \mathbf{a}, \mathbf{a}^0 \rangle \ge \sqrt{\frac{r}{n}} \|\mathbf{a}\|.$$

Let $\lambda_{\max}(\cdot)$ denote the largest singular value of a given matrix. The following lemma is important.

Lemma 3.1 *Given a nonzero* $A \in \mathbb{R}^{m \times n}$, *with* $(\mathbf{y}, \mathbf{z})$ *being the normalized singular vector pair corresponding to* $\lambda_{\max}(A)$. *Let* $\mathbf{z}^0 = [\mathbf{z}]^{\downarrow,r} / \| [\mathbf{z}]^{\downarrow,r} \|$ *with* $1 \le r \le n$. *Then there holds*

$$\left\| A\mathbf{z}^0 \right\| \ge \sqrt{\frac{r}{n}} \lambda_{\max}(A).$$

Proof From the definition of $\mathbf{z}$, we see that $\|A\mathbf{z}\| = \lambda_{\max}(A)$ and $A^\top A\mathbf{z} = \lambda_{\max}^2(A)\mathbf{z}$. Therefore,

$$\begin{aligned} \lambda_{\max}(A) \left\| A\mathbf{z}^0 \right\| &= \left\| A\mathbf{z}^0 \right\| \cdot \|A\mathbf{z}\| \\ &\ge \left\langle A\mathbf{z}^0, A\mathbf{z} \right\rangle \\ &= \lambda_{\max}(A) \cdot \left\langle A\mathbf{z}^0, \mathbf{y} \right\rangle = \lambda_{\max}^2(A) \cdot \left\langle \mathbf{z}^0, \mathbf{z} \right\rangle \\ &\ge \sqrt{\frac{r}{n}} \lambda_{\max}^2(A), \end{aligned}$$

where the last inequality follows from Proposition 3.1, the definition of $\mathbf{z}^0$, and that $\|\mathbf{z}\| = 1$. This completes the proof. □

3.1 The first algorithm

To illustrate the first algorithm, we denote $\mathbf{e}_j^{i_j} \in \mathbb{R}^{n_j}$, $1 \le i_j \le n_j$, $j = 1, \ldots, d$, as standard basis vectors in $\mathbb{R}^{n_j}$. For example, $\mathbf{e}_2^1$ is a vector in $\mathbb{R}^{n_2}$ with the first entry being one and the remaining ones being zero. Denote $\boldsymbol{r} := (r_1, \ldots, r_d)$; without loss of generality, we assume that $r_1 \le \cdots \le r_d$.

Algorithm $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0) = \text{approx\_alg}(\mathcal{A}, \boldsymbol{r})$ (A)

1. For each $i_j = 1, \ldots, n_j$, $j = 1, \ldots, d-1$, compute $\left[\mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-1}^{i_{d-1}}\right]^{\downarrow, r_d} \in \mathbb{R}^{n_d}$.
2. Let $(\bar{i}_1, \ldots, \bar{i}_{d-1})$ be a tuple of indices such that

$$\left\|\left[\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-1}^{\bar{i}_{d-1}}\right]^{\downarrow, r_d}\right\| = \max_{1 \le i_j \le n_j, 1 \le j \le d-1} \left\|\left[\mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-1}^{i_{d-1}}\right]^{\downarrow, r_d}\right\|;$$

denote $\bar{\mathbf{x}}_d^0 := \left[\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-1}^{\bar{i}_{d-1}}\right]^{\downarrow, r_d}$ and $\mathbf{x}_d^0 := \bar{\mathbf{x}}_d^0 / \|\bar{\mathbf{x}}_d^0\|$.
3. Sequentially update

$$\begin{aligned}
\bar{\mathbf{x}}_{d-1}^0 &= \left[\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}} \mathbf{x}_d^0\right]^{\downarrow, r_{d-1}}, \mathbf{x}_{d-1}^0 = \bar{\mathbf{x}}_{d-1}^0 / \left\|\bar{\mathbf{x}}_{d-1}^0\right\|, \\
\bar{\mathbf{x}}_{d-2}^0 &= \left[\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-3}^{\bar{i}_{d-3}} \mathbf{x}_{d-1}^0 \mathbf{x}_d^0\right]^{\downarrow, r_{d-2}}, \mathbf{x}_{d-2}^0 = \bar{\mathbf{x}}_{d-2}^0 / \left\|\bar{\mathbf{x}}_{d-2}^0\right\|, \\
&\vdots \\
\bar{\mathbf{x}}_1^0 &= \left[\mathcal{A}\mathbf{x}_2^0 \cdots \mathbf{x}_d^0\right]^{\downarrow, r_1}, \mathbf{x}_1^0 = \bar{\mathbf{x}}_1^0 / \left\|\bar{\mathbf{x}}_1^0\right\|.
\end{aligned}$$

4. Return $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$.

It is clear that $\mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-1}^{i_{d-1}}$'s are mode-$d$ fibers of $\mathcal{A}$. For the definition of fibers, one can refer to [16]; following the Matlab notation we have $\mathcal{A}(i_1, \ldots, i_{d-1}, :) = \mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-1}^{i_{d-1}}$. Algorithm A is a straightforward extension of [2, Algorithm 1] for sparse symmetric matrix PCA to higher-order tensor cases. Intuitively, the first two steps of Algorithm A enumerate all the mode-d fibers of $\mathcal{A}$, such that the select one admits the largest length with respect to its largest r_d entries (in magnitude). $\mathbf{x}_d^0$ is then given by the normalization of this fiber. Then, according to (3.4), the remaining $\mathbf{x}_j^0$ are in fact obtained by sequentially updated as

$$\mathbf{x}_j^0 \in \arg \max_{\|\mathbf{y}\|=1, \|\mathbf{y}\|_0 \le r_j} \mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{j-1}^{\bar{i}_{j-1}} \mathbf{y} \mathbf{x}_{j+1}^0 \cdots \mathbf{x}_d^0, \ j = d-1, \ldots, 1. \tag{3.5}$$

We consider the computational complexity of Algorithm A. Computing $\mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-1}^{i_{d-1}}$ takes $O(n_d)$ flops, while it takes $O(n_d \log(n_d))$ flops to compute $\left[\mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-1}^{i_{d-1}}\right]^{\downarrow, r}$ by using quick-sort. Thus the first two steps take $O(n_1 \cdots n_d \log(n_d) + n_d \log(n_d))$ flops. Computing $\mathbf{x}_j^0$, $j = 1, \ldots, d-1$ respectively takes $O(\prod_{k=1}^{j} n_k \cdot n_d + n_j \log(n_j))$ flops. Thus the total complexity can be roughly estimated as $O(\prod_{j=1}^{d} n_j \cdot \log(n_d) + \sum_{j=1}^{d} n_j \log(n_j))$.

We first show that Algorithm A is well-defined.

Proposition 3.2 *If $\mathcal{A} \neq 0$ and $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$ is generated by Algorithm* A*, then $\mathbf{x}_j^0 \neq 0$, $1 \le j \le d$.*

Proof Since $\mathcal{A}$ is a nonzeros tensor, there exists at least one fiber that is not identically zero. Therefore, the definition of $\bar{\mathbf{x}}_d^0$ shows that $\bar{\mathbf{x}}_d^0$ is not identically zero, and hence $\mathbf{x}_d^0$. We also observe that $\left\langle \mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}} \mathbf{x}_d^0, \mathbf{e}_{d-1}^{\bar{i}_{d-1}} \right\rangle = \mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-1}^{\bar{i}_{d-1}} \mathbf{x}_d^0 = \|\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-1}^{\bar{i}_{d-1}}\| > 0$, which

implies that $\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-2}^{\bar{i}_{d-2}}\mathbf{x}_d^0 \neq 0$, and so $\mathbf{x}_{d-1}^0 \neq 0$. (3.5) shows that $\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-2}^{\bar{i}_{d-2}}\mathbf{x}_{d-1}^0\mathbf{x}_d^0 \geq \mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-1}^{\bar{i}_{d-1}}\mathbf{x}_d^0 > 0$. Similarly, we can show that $\mathbf{x}_{d-2}^0 \neq 0, \ldots, \mathbf{x}_1^0 \neq 0$. □

We next first present the approximation bound when $d = 3$. The bound extends that of [2] to higher-order cases.

Theorem 3.1 *Denote v^{opt} as the optimal value of (2.3) when $d = 3$.*
Let $(\mathbf{x}_1^0, \mathbf{x}_2^0, \mathbf{x}_3^0)$ be generated by Algorithm A. Then it holds that

$$\mathcal{A}\mathbf{x}_1^0\mathbf{x}_2^0\mathbf{x}_3^0 \geq \frac{v^{\mathrm{opt}}}{\sqrt{r_1 r_2}}.$$

Proof Denote $(\mathbf{x}_1^*, \mathbf{x}_2^*, \mathbf{x}_3^*)$ as a maximizer of (2.3). By noticing that $\|\mathbf{x}_1^*\|_0 \leq r_1$ and $\mathcal{A}\mathbf{x}_1^*\mathbf{x}_2^*\mathbf{x}_3^* = \sum_{\{i_1:x_{1,i_1}^*\neq 0\}}^{n_1} x_{1,i_1}^* \cdot \left(\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^*\mathbf{x}_3^*\right)$; recalling that we write $\mathbf{x}_j := [x_{j,1}, \ldots, x_{j,n_j}]^\top$, we have

$$\begin{aligned} v^{\mathrm{opt}} &= \mathcal{A}\mathbf{x}_1^*\mathbf{x}_2^*\mathbf{x}_3^* = \sum\nolimits_{\{i_1:x_{1,i_1}^*\neq 0\}}^{n_1} x_{1,i_1}^* \left(\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^*\mathbf{x}_3^*\right) \\ &\leq \sqrt{\sum\nolimits_{\{i_1:x_{1,i_1}^*\neq 0\}}^{n_1} \left(x_{1,i_1}^*\right)^2} \sqrt{\sum\nolimits_{\{i_1:x_{1,i_1}^*\neq 0\}}^{n_1} \left(\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^*\mathbf{x}_3^*\right)^2} \\ &\leq \|\mathbf{x}_1^*\| \cdot \sqrt{r_1} \max_{i_1} \left|\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^*\mathbf{x}_3^*\right| = \sqrt{r_1} \max_{i_1} \left|\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^*\mathbf{x}_3^*\right|, \end{aligned} \tag{3.6}$$

where the first inequality uses the Cauchy-Schwartz inequality, and the last equality follows from $\|\mathbf{x}_1^*\| = 1$. In the same vein, we have

$$v^{\mathrm{opt}} \leq \sqrt{r_1} \max_{i_1} \left|\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^*\mathbf{x}_3^*\right| \leq \sqrt{r_1 r_2} \max_{i_1,i_2} \left|\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{e}_2^{i_2}\mathbf{x}_3^*\right|.$$

Assume that $\left|\mathcal{A}\mathbf{e}_1^{\hat{i}_1}\mathbf{e}_2^{\hat{i}_2}\mathbf{x}_3^*\right| = \max_{i_1,i_2} \left|\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{e}_2^{i_2}\mathbf{x}_3^*\right|$; denote

$$\hat{\mathbf{x}}_3 := \left[\mathcal{A}\mathbf{e}_1^{\hat{i}_1}\mathbf{e}_2^{\hat{i}_2}\right]^{\downarrow,r_3} \Big/ \left\|\left[\mathcal{A}\mathbf{e}_1^{\hat{i}_1}\mathbf{e}_2^{\hat{i}_2}\right]^{\downarrow,r_3}\right\|.$$

(3.4) shows that $\left|\mathcal{A}\mathbf{e}_1^{\hat{i}_1}\mathbf{e}_2^{\hat{i}_2}\mathbf{x}_3^*\right| \leq \left|\mathcal{A}\mathbf{e}_1^{\hat{i}_1}\mathbf{e}_2^{\hat{i}_2}\hat{\mathbf{x}}_3\right|$. By noticing the definition of $\mathbf{x}_3^0$, we then have

$$\begin{aligned} \left|\mathcal{A}\mathbf{e}_1^{\hat{i}_1}\mathbf{e}_2^{\hat{i}_2}\mathbf{x}_3^*\right| \leq \left|\mathcal{A}\mathbf{e}_1^{\hat{i}_1}\mathbf{e}_2^{\hat{i}_2}\hat{\mathbf{x}}_3\right| &= \left\|\left[\mathcal{A}\mathbf{e}_1^{\hat{i}_1}\mathbf{e}_2^{\hat{i}_2}\right]^{\downarrow,r_3}\right\| \leq \max_{i_1,i_2} \left\|\left[\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{e}_2^{i_2}\right]^{\downarrow,r_3}\right\| \\ &= \mathcal{A}\mathbf{e}_1^{\bar{i}_1}\mathbf{e}_2^{\bar{i}_2}\mathbf{x}_3^0. \end{aligned}$$

Finally, recalling the definitions of $\mathbf{x}_1^0$ and $\mathbf{x}_2^0$ and combining the above pieces, we arrive at

$$v^{\mathrm{opt}} \leq \sqrt{r_1 r_2}\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\mathbf{e}_2^{\bar{i}_2}\mathbf{x}_3^0 \leq \sqrt{r_1 r_2}\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\mathbf{x}_2^0\mathbf{x}_3^0 \leq \sqrt{r_1 r_2}\mathcal{A}\mathbf{x}_1^0\mathbf{x}_2^0\mathbf{x}_3^0,$$

as desired. □

In the same spirit of the proof of Theorem 3.1, for general $d \geq 3$, one can show that

$$\begin{aligned} v^{\mathrm{opt}} &\leq \sqrt{r_1} \max_{i_1} \left|\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^*\cdots\mathbf{x}_d^*\right| \leq \cdots \leq \sqrt{\prod\nolimits_{j=1}^{d-1} r_j} \max_{i_1,\ldots,i_{d-1}} \left|\mathcal{A}\mathbf{e}_1^{i_1}\cdots\mathbf{e}_{d-1}^{i_{d-1}}\mathbf{x}_d^*\right| \\ &\leq \sqrt{\prod\nolimits_{j=1}^{d-1} r_j} \max_{i_1,\ldots,i_{d-1}} \mathcal{A}\mathbf{x}_1^0\cdots\mathbf{x}_d^0. \end{aligned}$$

Theorem 3.2 *Denote $v^{\rm opt}$ as the optimal value of the problem (2.3).*

Let $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$ be generated by Algorithm A for a general d-th order tensor $\mathcal{A}$. Then it holds that

$$\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0 \geq \frac{v^{\rm opt}}{\sqrt{\prod_{j=1}^{d-1} r_j}}.$$

3.2 The second algorithm

The second algorithm is presented as follows.

Algorithm $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0) = \text{approx\_alg}(\mathcal{A}, \boldsymbol{r})$ (B)

1. For each tuple $(i_1, \ldots, i_{d-2})$, $i_j = 1, \ldots, n_j$, $1 \leq j \leq d-2$, solve the matrix singular value problem $\max_{\|\mathbf{x}_{d-1}\|=\|\mathbf{x}_d\|=1} \mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-2}^{i_{d-2}} \mathbf{x}_{d-1}\mathbf{x}_d$.
2. Let $(\bar{i}_1, \ldots, \bar{i}_{d-2})$ be the optimal tuple of indices with $(\bar{\mathbf{x}}_{d-1}, \bar{\mathbf{x}}_d)$ being the optimal solution pair, i.e.,

$$\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}} \bar{\mathbf{x}}_{d-1}\bar{\mathbf{x}}_d = \max_{1\leq i_j \leq n_j, 1\leq j\leq d-2, \|\mathbf{x}_{d-1}\|=\|\mathbf{x}_d\|=1} \mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-2}^{i_{d-2}} \mathbf{x}_{d-1}\mathbf{x}_d;$$

denote $\mathbf{x}_d^0 := \left[\bar{\mathbf{x}}_d\right]^{\downarrow, r_d} / \| \left[\bar{\mathbf{x}}_d\right]^{\downarrow, r_d} \|$.
3. Sequentially update $\mathbf{x}_{d-1}^0, \ldots, \mathbf{x}_1^0$ as Step 3 of Algorithm A.
4. Return $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$.

The main difference from Algorithm A mainly lies in the first step, where Algorithm A requires to find the fiber with the largest length with respect to the largest r_3 entries (in magnitude), while the first step of Algorithm B looks for the matrix $\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}} \in \mathbb{R}^{n_{d-1}\times n_d}$ with the largest spectral radius among all $\mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-2}^{i_{d-2}}$. Algorithm B combines the ideas of both Algorithms 1 and 2 of [2] and extends them to higher-order tensors. When reducing to the matrix case, our algorithm here is still different from [2], as we find sparse singular vector pairs, while [2] pursues sparse eigenvectors of a symmetric matrix.

The computational complexity of Algorithm B is as follows. Computing the largest singular value of $\mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-2}^{i_{d-2}}$ takes $O(\min\{n_{d-1}^2 n_d, n_{d-1} n_d^2\})$ flops in theory. Computing $\mathbf{x}_d^0$ takes $O(n_d \log(n_d))$ flops. Thus the first two steps take $O(n_1 \cdots n_{d-2} \min\{n_{d-1}^2 n_d, n_{d-1} n_d^2\} + n_d \log(n_d))$ flops. The flops of the third step are the same as those of Algorithm A. As a result, the total complexity is dominated by $O(n_1 \cdots n_{d-2} \min\{n_{d-1}^2 n_d, n_{d-1} n_d^2\} + \sum_{j=1}^d n_j \log(n_j))$.

Algorithm B is also well-defined as follow.

Proposition 3.3 *If $\mathcal{A} \neq 0$ and $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$ is generated by Algorithm* B*, then $\mathbf{x}_j^0 \neq 0$, $1 \leq j \leq d$.*

Proof The definition of $(\bar{i}_1, \ldots, \bar{i}_{d-2})$ shows that the matrix $\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}} \neq 0$, and hence $\bar{\mathbf{x}}_d \neq 0$ and $\mathbf{x}_d^0 \neq 0$. We also observe from step 2 that $\bar{\mathbf{x}}_d = \mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}} \bar{\mathbf{x}}_{d-1} / \|\mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}} \bar{\mathbf{x}}_{d-1}\|$, and so

$$\left\langle \mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-2}^{\bar{i}_{d-2}}\mathbf{x}_d^0, \bar{\mathbf{x}}_{d-1}\right\rangle = \left\langle \mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-2}^{\bar{i}_{d-2}}\bar{\mathbf{x}}_{d-1}, \mathbf{x}_d^0\right\rangle$$
$$= \left\langle \bar{\mathbf{x}}_d, \mathbf{x}_d^0\right\rangle \left\|\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-2}^{\bar{i}_{d-2}}\bar{\mathbf{x}}_{d-1}\right\| > 0,$$

implying that $\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-2}^{\bar{i}_{d-2}}\mathbf{x}_d^0 \neq 0$, and so $\mathbf{x}_{d-1}^0 = \left[\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-2}^{\bar{i}_{d-2}}\mathbf{x}_d^0\right]^{\downarrow,r_{d-1}} \neq 0$. Similar arguments apply to show that $\mathbf{x}_{d-2}^0 \neq 0, \ldots, \mathbf{x}_1^0 \neq 0$ then. □

To analyze the approximation bound, we need the following proposition.

Proposition 3.4 *Let $\bar{i}_1, \ldots, \bar{i}_{d-2}$ and $(\bar{\mathbf{x}}_{d-1}, \bar{\mathbf{x}}_d)$ be defined in Algorithm* B. *Then it holds that*

$$\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\cdots\mathbf{e}_{d-2}^{\bar{i}_{d-2}}\bar{\mathbf{x}}_{d-1}\bar{\mathbf{x}}_d \geq \max_{i_1,\ldots,i_{d-2},\|\mathbf{y}\|=\|\mathbf{z}\|=1,\|\mathbf{y}\|_0\leq r_{d-1},\|\mathbf{z}\|_0\leq r_d} \left|\mathcal{A}\mathbf{e}_1^{i_1}\cdots\mathbf{e}_{d-2}^{i_{d-2}}\mathbf{y}\mathbf{z}\right|.$$

Proof The result holds by noticing the definition of $(\mathbf{e}_1^{\bar{i}_1}, \ldots, \mathbf{e}_{d-2}^{\bar{i}_{d-2}}, \bar{\mathbf{x}}_{d-1}, \bar{\mathbf{x}}_d)$, and the additional sparsity constraints in the right-hand side of the inequality. □

We first derive the approximation bound with $d = 3$ as an illustration.

Theorem 3.3 *Let v^{opt} be defined as that in Theorem 3.1 when $d = 3$, and let $(\mathbf{x}_1^0, \mathbf{x}_2^0, \mathbf{x}_3^0)$ be generated by Algorithm B. Then it holds that*

$$\mathcal{A}\mathbf{x}_1^0\mathbf{x}_2^0\mathbf{x}_3^0 \geq \sqrt{\frac{r_2r_3}{n_2n_3r_1}}v^{\mathrm{opt}}.$$

Proof Denote $(\mathbf{x}_1^*, \mathbf{x}_2^*, \mathbf{x}_3^*)$ as a maximizer to (2.3). We have

$$\begin{aligned} v^{\mathrm{opt}} &\leq \sqrt{r_1}\max_{i_1}\left|\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^*\mathbf{x}_3^*\right| \\ &\leq \sqrt{r_1}\max_{i_1,\|\mathbf{y}\|=\|\mathbf{z}\|=1,\|\mathbf{y}\|_0\leq r_2,\|\mathbf{z}\|_0\leq r_3}\left|\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{y}\mathbf{z}\right| \\ &\leq \sqrt{r_1}\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\bar{\mathbf{x}}_2\bar{\mathbf{x}}_3, \end{aligned} \tag{3.7}$$

where the first inequality is the same as (3.6), while Proposition 3.4 gives the last one. Now denote $A := \mathcal{A}\mathbf{e}_1^{\bar{i}_1} \in \mathbb{R}^{n_2\times n_3}$. Our remaining task is to show that

$$(\mathbf{x}_2^0)^\top A\mathbf{x}_3^0 \geq \sqrt{\frac{r_2r_3}{n_2n_3}}\bar{\mathbf{x}}_2^\top A\bar{\mathbf{x}}_3. \tag{3.8}$$

Recalling the definition of $(\bar{\mathbf{x}}_2, \bar{\mathbf{x}}_3)$, we have $\lambda_{\max}(A) = \bar{\mathbf{x}}_2^\top A\bar{\mathbf{x}}_3$. Lemma 3.1 tells us that

$$\left\|A\mathbf{x}_3^0\right\| \geq \sqrt{\frac{r_3}{n_3}}\lambda_{\max}(A) = \sqrt{\frac{r_3}{n_3}}\bar{\mathbf{x}}_2^\top A\bar{\mathbf{x}}_3; \tag{3.9}$$

on the other hand, since

$$\bar{\mathbf{x}}_2^0 = \left[\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\mathbf{x}_3^0\right]^{\downarrow,r_2} = \left[A\mathbf{x}_3^0\right]^{\downarrow,r_2},\ \mathbf{x}_2^0 = \bar{\mathbf{x}}_2^0/\left\|\bar{\mathbf{x}}_2^0\right\|,$$

it follows from Proposition 3.1 that

$$\left\langle A\mathbf{x}_3^0, \mathbf{x}_2^0 \right\rangle \geq \sqrt{\frac{r_2}{n_2}} \left\| A\mathbf{x}_3^0 \right\|,$$

combining which with (3.9) gives (3.8). Finally, (3.8) together with (3.7) and the definition of $\mathbf{x}_1^0$ yields

$$\begin{aligned} \mathcal{A}\mathbf{x}_1^0\mathbf{x}_2^0\mathbf{x}_3^0 &\geq \mathcal{A}\mathbf{e}_1^{\bar{i}_1}\mathbf{x}_2^0\mathbf{x}_3^0 = \left\langle A\mathbf{x}_3^0, \mathbf{x}_2^0 \right\rangle \geq \sqrt{\frac{r_2 r_3}{n_2 n_3}}\bar{\mathbf{x}}_2^\top A\bar{\mathbf{x}}_3 = \sqrt{\frac{r_2 r_3}{n_2 n_3}}\mathcal{A}\mathbf{e}_1^{\bar{i}_1}\bar{\mathbf{x}}_2\bar{\mathbf{x}}_3 \\ &\geq \sqrt{\frac{r_2 r_3}{n_2 n_3 r_1}} v^{\text{opt}}, \end{aligned}$$

as desired. □

The following approximation bound is presented for general order $d \geq 3$.

Theorem 3.4 *Let* $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$ *be generated by Algorithm* B. *Then it holds that*

$$\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0 \geq \sqrt{\frac{r_{d-1} r_d}{n_{d-1} n_d \prod_{j=1}^{d-2} r_j}} v^{\text{opt}}.$$

Proof Similar to (3.7), we have

$$\begin{aligned} v^{\text{opt}} &\leq \sqrt{r_1} \max_{i_1} |\mathcal{A}\mathbf{e}_1^{i_1}\mathbf{x}_2^* \cdots \mathbf{x}_d^*| \leq \cdots \\ &\leq \sqrt{\prod_{j=1}^{d-2} r_j} \max_{i_1,\ldots,i_{d-2}} |\mathcal{A}\mathbf{e}_1^{i_1} \cdots \mathbf{e}_{d-2}^{i_{d-2}}\mathbf{x}_{d-1}^*\mathbf{x}_d^*| \\ &\leq \sqrt{\prod_{j=1}^{d-2} r_j}\, \mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}}\bar{\mathbf{x}}_{d-1}\bar{\mathbf{x}}_d. \end{aligned}$$

Similar to the proof of (3.8), we can obtain

$$\begin{aligned} v^{\text{opt}} &\leq \sqrt{\prod_{j=1}^{d-2} r_j}\, \mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}}\bar{\mathbf{x}}_{d-1}\bar{\mathbf{x}}_d \\ &\leq \sqrt{\frac{n_{d-1} n_d \prod_{j=1}^{d-2} r_j}{r_{d-1} r_d}}\, \mathcal{A}\mathbf{e}_1^{\bar{i}_1} \cdots \mathbf{e}_{d-2}^{\bar{i}_{d-2}}\mathbf{x}_{d-1}^0\mathbf{x}_d^0 \\ &\leq \sqrt{\frac{n_{d-1} n_d \prod_{j=1}^{d-2} r_j}{r_{d-1} r_d}}\, \mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0. \end{aligned}$$

□

3.3 The third algorithm

We begin with the illustration from third-order tensors. We will employ the Matlab function `reshape` to denote tensor folding/unfolding operations. For instance, given $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, $\mathbf{a} = \texttt{reshape}(\mathcal{A}, \prod_{j=1}^{d} n_j, 1)$ means the unfolding of $\mathcal{A}$ to a vector $\mathbf{a}$ in $\mathbb{R}^{\prod_{j=1}^{d} n_j}$, while $\mathcal{A} = \texttt{reshape}(\mathbf{a}, n_1, \ldots, n_d)$ means the folding of $\mathbf{a}$ back to $\mathcal{A}$.

Algorithm $(\mathbf{x}_1^0, \mathbf{x}_2^0, \mathbf{x}_3^0) = \text{approx\_alg}(\mathcal{A}, r_1, r_2, r_3)$ (C0)

1. Unfold $\mathcal{A}$ to $A_1 := \texttt{reshape}(\mathcal{A}, n_1, n_2 n_3) \in \mathbb{R}^{n_1 \times n_2 n_3}$; solve the matrix singular value problem

$$(\bar{\mathbf{x}}_1, \bar{\mathbf{w}}_1) \in \arg\max_{\|\mathbf{x}\|=\|\mathbf{w}\|=1} \mathbf{x}^\top A_1 \mathbf{w};$$

denote $\mathbf{x}_1^0 := [\bar{\mathbf{x}}_1]^{\downarrow, r_1} / \left\| [\bar{\mathbf{x}}_1]^{\downarrow, r_1} \right\|$.

2. Let $A_2 := \texttt{reshape}(A_1^\top \mathbf{x}_1^0, n_2, n_3) \in \mathbb{R}^{n_2 \times n_3}$; solve the matrix singular value problem

$$(\bar{\mathbf{x}}_2, \bar{\mathbf{w}}_2) \in \arg\max_{\|\mathbf{y}\|=\|\mathbf{w}\|=1} \mathbf{y}^\top A_2 \mathbf{w};$$

denote $\mathbf{x}_2^0 := [\bar{\mathbf{x}}_2]^{\downarrow, r_2} / \left\| [\bar{\mathbf{x}}_2]^{\downarrow, r_2} \right\|$.

3. Compute $\bar{\mathbf{x}}_3^0 := \left[A_2^\top \mathbf{x}_2^0 \right]^{\downarrow, r_3}$, $\mathbf{x}_3^0 := \bar{\mathbf{x}}_3^0 / \left\| \bar{\mathbf{x}}_3^0 \right\|$.

4. Return $(\mathbf{x}_1^0, \mathbf{x}_2^0, \mathbf{x}_3^0)$.

Different from Algorithm B, Algorithm C0 is mainly based on a series of computing leading singular vector pairs of certain matrices. In fact, Algorithm C0 generalizes the approximation algorithm for dense tensor BR1Approx [13, Algorithm 1 with DR 2] to our sparse setting; the main difference lies in the truncation of $\bar{\mathbf{x}}_j$ to obtain the sparse solution $\mathbf{x}_j^0$. In particular, if no sparsity is required, i.e., $r_j = n_j$, $j = 1, 2, 3$, then Algorithm C0 boils down essentially to [13, Algorithm 1 with DR 2]. The next proposition shows that Algorithm C0 is well-defined.

Proposition 3.5 *If $\mathcal{A} \neq 0$ and $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$ is generated by Algorithm* C0*, then $\mathbf{x}_j^0 \neq 0$, $1 \leq j \leq 3$.*

Proof It is clear that $\bar{\mathbf{x}}_1 \neq 0$, $\mathbf{x}_1^0 \neq 0$, and $\bar{\mathbf{w}}_1 \neq 0$ due to that $A_1 \neq 0$. We have $\langle \mathbf{x}_1^0, A_1 \bar{\mathbf{w}}_1 \rangle = \langle \mathbf{x}_1^0, \bar{\mathbf{x}}_1 \rangle \|A_1 \bar{\mathbf{w}}_1\| > 0$, and so $A_1^\top \mathbf{x}_1^0 \neq 0$, and $A_2 \neq 0$. Similar argument shows that $\mathbf{x}_2^0 \neq 0$ and $\mathbf{x}_3^0 \neq 0$. □

Due to the presence of the truncation, deriving the approximation bound is different from that of [13]. In particular, we need Lemma 3.1 to build bridges in the analysis. The following relation

$$\mathcal{A}\mathbf{x}_1^0 \mathbf{x}_2^0 \mathbf{x}_3^0 = \left\langle A_1^\top \mathbf{x}_1^0, \mathbf{x}_3^0 \otimes \mathbf{x}_2^0 \right\rangle = \left\langle A_2, \mathbf{x}_2^0 (\mathbf{x}_3^0)^\top \right\rangle = \left\langle A_2^\top \mathbf{x}_2^0, \mathbf{x}_3^0 \right\rangle \tag{3.10}$$

is also helpful in the analysis, where $\otimes$ denotes the Kronecker product [16].

Theorem 3.5 *Let v^{opt} be defined as that in Theorem 3.1 when $d = 3$, and let $(\mathbf{x}_1^0, \mathbf{x}_2^0, \mathbf{x}_3^0)$ be generated by Algorithm* C0*. Then it holds that*

$$\mathcal{A}\mathbf{x}_1^0 \mathbf{x}_2^0 \mathbf{x}_3^0 \geq \sqrt{\frac{r_1 r_2 r_3}{n_1 n_2 n_3 n_2}} \lambda_{\max}(A_1) \geq \sqrt{\frac{r_1 r_2 r_3}{n_1 n_2 n_3 n_2}} v^{\text{opt}}.$$

Proof From the definition of A_1, $\bar{\mathbf{x}}_1$ and $\bar{\mathbf{w}}_1$, we see that

$$\lambda_{\max}(A_1) = \bar{\mathbf{x}}_1^\top A_1 \bar{\mathbf{w}}_1 \geq \max_{\|\mathbf{x}\|=\|\mathbf{y}\|=\|\mathbf{z}\|=1} \mathcal{A}\mathbf{x}\mathbf{y}\mathbf{z} \geq v^{\text{opt}}.$$

Therefore, to prove the approximation bound, it suffices to show that

$$\mathcal{A}\mathbf{x}_1^0\mathbf{x}_2^0\mathbf{x}_3^0 \geq \sqrt{\frac{r_1r_2r_3}{n_1n_2n_3n_2}}\lambda_{\max}(A_1). \tag{3.11}$$

To this end, recalling Lemma 3.1 and the definition of $\mathbf{x}_1^0$ (which is a truncation of the leading left singular vector of A_1), we obtain

$$\left\|A_1^\top \mathbf{x}_1^0\right\| \geq \sqrt{\frac{r_1}{n_1}}\lambda_{\max}(A_1). \tag{3.12}$$

Since $A_2 = \texttt{reshape}(A_1^\top \mathbf{x}_1^0, n_2, n_3)$, it holds that $\|A_2\|_F = \left\|A_1^\top \mathbf{x}_1^0\right\|$. Using again Lemma 3.1 and recalling the definition of $\mathbf{x}_2^0$ (which is a truncation of the leading left singular vector of A_2), we get

$$\left\|A_2^\top \mathbf{x}_2^0\right\| \geq \sqrt{\frac{r_2}{n_2}}\lambda_{\max}(A_2) \geq \sqrt{\frac{r_2}{n_2^2}}\|A_2\|_F = \sqrt{\frac{r_2}{n_2^2}}\left\|A_1^\top \mathbf{x}_1^0\right\|, \tag{3.13}$$

where the second inequality follows from the relation between the spectral norm and the Frobenius norm of a matrix. Finally, Proposition 3.1 and the definition of $\mathbf{x}_3^0$ gives that

$$\mathcal{A}\mathbf{x}_1^0\mathbf{x}_2^0\mathbf{x}_3^0 = \left\langle A_2^\top \mathbf{x}_2^0, \mathbf{x}_3^0\right\rangle \geq \sqrt{\frac{r_3}{n_3}}\left\|A_2^\top \mathbf{x}_2^0\right\|, \tag{3.14}$$

where the equality follows from (3.10). Combining the above relation with (3.12) and (3.13) gives (3.11). This completes the proof. □

When extending to d-th order tensors, the algorithm is presented as follows.

Algorithm $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0) = \text{approx\_alg}(\mathcal{A}, \boldsymbol{r})$ (C)

1. Unfold $\mathcal{A}$ to $A_1 = \texttt{reshape}(\mathcal{A}, n_1, \prod_{j=2}^d n_j) \in \mathbb{R}^{n_1 \times \prod_{j=2}^d n_j}$; solve the matrix singular value problem

$$(\bar{\mathbf{x}}_1, \bar{\mathbf{w}}_1) \in \arg\max_{\|\mathbf{x}_1\|=\|\mathbf{w}_1\|=1} \mathbf{x}_1^\top A_1 \mathbf{w}_1;$$

denote $\mathbf{x}_1^0 := [\bar{\mathbf{x}}_1]^{\downarrow, r_1} / \left\|[\bar{\mathbf{x}}_1]^{\downarrow, r_1}\right\|$.

2. For $j = 2, \ldots, d-1$, denote $A_j := \texttt{reshape}(A_{j-1}^\top \mathbf{x}_{j-1}^0, n_j, \prod_{k=j+1}^d n_k) \in \mathbb{R}^{n_j \times \prod_{k=j+1}^d n_k}$; solve the matrix singular value problem

$$(\bar{\mathbf{x}}_j, \bar{\mathbf{w}}_j) \in \arg\max_{\|\mathbf{x}_j\|=\|\mathbf{w}_j\|=1} \mathbf{x}_j^\top A_j \mathbf{w}_j;$$

denote $\mathbf{x}_j^0 := \left[\bar{\mathbf{x}}_j\right]^{\downarrow, r_j} / \left\|\left[\bar{\mathbf{x}}_j\right]^{\downarrow, r_j}\right\|$.

3. Compute $\bar{\mathbf{x}}_d^0 := \left[A_{d-1}^\top \mathbf{x}_{d-1}^0\right]^{\downarrow, r_d}$, $\mathbf{x}_d^0 := \bar{\mathbf{x}}_d^0 / \left\|\bar{\mathbf{x}}_d^0\right\|$.

4. Return $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$.

Remark 3.1 In step 2, by noticing the recursive definition of A_j, one can check that A_j is in fact the same as $\texttt{reshape}(\mathcal{A}\mathbf{x}_1^0\cdots\mathbf{x}_{j-1}^0, n_j, \prod_{k=j+1}^d n_k)$, where $\mathcal{A}\mathbf{x}_1^0\cdots\mathbf{x}_{j-1}^0$ is regarded as a tensor of size $n_j\times\cdots\times n_d$ with $\left(\mathcal{A}\mathbf{x}_1^0\cdots\mathbf{x}_{j-1}^0\right)_{i_j\cdots i_d} = \sum_{i_1=1,\ldots,i_{j-1}=1}^{n_1,\ldots,n_{j-1}} \mathcal{A}_{i_1\cdots i_{j-1}i_j\cdots i_d}(\mathbf{x}_1^0)_{i_1}\cdots(\mathbf{x}_{j-1}^0)_{i_{j-1}}$.

The computational complexity of the first step is $O(n_1^2 n_2\cdots n_d + n_1\log(n_1))$, where $O(n_1^2 n_2\cdots n_d)$ comes from solving the singular value problem of an $n_1\times\prod_{j=2}^d n_j$ matrix. In the second step, for each j, computing $A_{j-1}^\top\mathbf{x}_{j-1}^0$ requires $O(n_{j-1}\cdots n_d)$ flops; computing the singular value problem requires $O(n_j^2 n_{j+1}\cdots n_d)$ flops; thus the complexity is $O(n_{j-1}\cdots n_d + n_j^2 n_{j+1}\cdots n_d + n_j\log(n_j))$. The third step is $O(n_{d-1}n_d + n_d\log(n_d))$. Thus the total complexity is dominated by $O(n_1\prod_{j=1}^d n_j + \sum_{j=1}^d n_j\log(n_j))$, which is the same as Algorithm B in theory.

Similar to Proposition 3.5, we can show that Algorithm C is well-defined, i.e., if $\mathcal{A}\neq 0$, then $\mathbf{x}_j^0\neq 0$, $1\le j\le d$. The proof is omitted.

Concerning the approximation bound, similar to (3.14), one has

$$\mathcal{A}\mathbf{x}_1^0\cdots\mathbf{x}_d^0 = \langle\mathcal{A}\mathbf{x}_1^0\cdots\mathbf{x}_{d-1}^0, \mathbf{x}_d^0\rangle = \langle A_{d-1}^\top\mathbf{x}_{d-1}^0, \mathbf{x}_d^0\rangle \ge \sqrt{\frac{r_d}{n_d}}\|A_{d-1}^\top\mathbf{x}_{d-1}^0\|,$$

where the second equality comes from Remark 3.1 that $\mathcal{A}\mathbf{x}_1^0\cdots\mathbf{x}_{d-2}^0 = A_{d-1}$ (up to a reshaping) and the inequality is due to Proposition 3.1. Analogous to (3.13), one can prove the following relation:

$$\left\|A_j^\top\mathbf{x}_j^0\right\| \ge \sqrt{\frac{r_j}{n_j^2}}\left\|A_{j-1}^\top\mathbf{x}_{j-1}^0\right\|,\ 2\le j\le d-1.$$

Based on the above relations and (3.12), for order $d\ge 3$, we present the approximation bound without proof.

Theorem 3.6 *Let* $(\mathbf{x}_1^0,\ldots,\mathbf{x}_d^0)$ *be generated by Algorithm* C. *Then it holds that*

$$\mathcal{A}\mathbf{x}_1^0\cdots\mathbf{x}_d^0 \ge \sqrt{\frac{\prod_{j=1}^d r_j}{\prod_{j=1}^d n_j}}\frac{\lambda_{\max}(A_1)}{\sqrt{\prod_{j=2}^{d-1} n_j}} \ge \sqrt{\frac{\prod_{j=1}^d r_j}{\prod_{j=1}^d n_j}}\frac{v^{\text{opt}}}{\sqrt{\prod_{j=2}^{d-1} n_j}}.$$

3.4 The fourth algorithm

Algorithm C computes $\bar{\mathbf{x}}_j$ from A_j via solving singular value problems. When the size of the tensor is huge, this might be time-consuming. To further accelerate the algorithm, we propose the following algorithm, which is similar to Algorithm C, while it obtains $\bar{\mathbf{x}}_j$ without solving singular value problems. Denote A^k as the k-th row of a matrix A. The algorithm is presented as follows:

Algorithm $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0) = \text{approx\_alg}(\mathcal{A}, \boldsymbol{r})$ (D)

1. Unfold $\mathcal{A}$ to $A_1 = \texttt{reshape}(\mathcal{A}, n_1, \prod_{j=2}^d n_j)$; let $A_1^{\tilde{k}}$ be the row of A_1 with the largest magnitude, i.e., $\|A_1^{\tilde{k}}\| = \max_{1\le k\le n_1} \left\|A_1^k\right\|_F$. Denote $\mathbf{w}_1 := (A_1^{\tilde{k}})^\top / \|(A_1^{\tilde{k}})^\top\|$ and let

$$\bar{\mathbf{x}}_1 = A_1 \mathbf{w}_1;$$

denote $\mathbf{x}_1^0 := [\bar{\mathbf{x}}_1]^{\downarrow, r_1} / \left\|[\bar{\mathbf{x}}_1]^{\downarrow, r_1}\right\|$.

2. For $j = 2, \ldots, d-1$, denote $A_j := \texttt{reshape}(A_{j-1}^\top \mathbf{x}_{j-1}^0, n_j, \prod_{k=j+1}^d n_k)$; let $A_j^{\tilde{k}}$ be the row of A_j with the largest magnitude. Denote $\mathbf{w}_j := (A_j^{\tilde{k}})^\top / \|(A_j^{\tilde{k}})^\top\|$ and let

$$\bar{\mathbf{x}}_j = A_j \mathbf{w}_j;$$

denote $\mathbf{x}_j^0 := \left[\bar{\mathbf{x}}_j\right]^{\downarrow, r_j} / \left\|\left[\bar{\mathbf{x}}_j\right]^{\downarrow, r_j}\right\|$.

3. Compute $\bar{\mathbf{x}}_d^0 := \left[A_{d-1}^\top \mathbf{x}_{d-1}^0\right]^{\downarrow, r_d}$, $\mathbf{x}_d^0 := \bar{\mathbf{x}}_d^0 / \left\|\bar{\mathbf{x}}_d^0\right\|$.

4. Return $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$.

It is clear from the above algorithm that computing $\bar{\mathbf{x}}_j$ only requires some matrix-vector productions, which has lower computational complexity than computing singular vectors. Numerical results presented in the next section will show the efficiency and effectiveness of this simple modification.

In Algorithm D, the first step needs $O(n_1 \cdots n_d + n_1 \log(n_1))$ flops; in the second step, for each j, the complexity is $O(n_{j-1} \cdots n_d + n_j \log(n_j))$; the third step is $O(n_{d-1} n_d + n_d \log(n_d))$. Thus the total complexity is dominated by $O(n_1 \cdots n_d + \sum_{j=1}^d n_j \log(n_j))$, which is lower than that of Algorithm C, due to the SVD-free computation of $\bar{\mathbf{x}}_j$'s.

Reducing to the dense tensor setting, i.e., $r_j = n_j$ for each j, Algorithm D is even new for dense tensor BR1Approx problems; when $d = 2$, similar ideas have not been applied to approximation algorithms for sparse matrix PCA/SVD yet.

The next proposition shows that Algorithm D is well-defined, whose proof is quite similar to that of Proposition 3.5 and is omitted.

Proposition 3.6 *If $\mathcal{A} \neq 0$ and $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$ is generated by Algorithm* D*, then $\mathbf{x}_j^0 \neq 0$, $1 \le j \le d$.*

The approximation bound analysis essentially relies on the following lemma.

Lemma 3.2 *Given $A \in \mathbb{R}^{m\times n}$, with $A^{\tilde{k}}$ being the row of A having the largest magnitude. Let $\mathbf{w} = (A^{\tilde{k}})^\top / \|(A^{\tilde{k}})^\top\|$, $\mathbf{x} = A\mathbf{w}$, and $\mathbf{x}^0 = [\mathbf{x}]^{\downarrow, r} / \|[\mathbf{x}]^{\downarrow, r}\|$ with $1 \le r \le m$. Then there holds*

$$\left\|A^\top \mathbf{x}^0\right\| \ge \sqrt{\frac{r}{m^2}} \|A\|_F .$$

Proof We have

$$\begin{aligned} \left\|A^\top \mathbf{x}^0\right\| &= \max_{\|\mathbf{z}\|=1} \left\langle A^\top \mathbf{x}^0, \mathbf{z} \right\rangle \\ &\ge \left\langle A^\top \mathbf{x}^0, \mathbf{w} \right\rangle = \left\langle \mathbf{x}, \mathbf{x}^0 \right\rangle \\ &\ge \sqrt{\frac{r}{m}} \|\mathbf{x}\|, \end{aligned}$$

Table 1 Comparisons of the proposed approximation algorithms on the approximation bound and computational complexity

Algorithm	Approximation bound	Computational complexity
Algorithm A	$\frac{v^{\text{opt}}}{\sqrt{r^{d-1}}}$	$O(n^d \log(n) + dn \log(n))$
Algorithm B	$\sqrt{\frac{r^2}{n^2}} \frac{v^{\text{opt}}}{\sqrt{r^{d-2}}}$	$O(n^{d+1} + dn \log(n))$
Algorithm C	$\sqrt{\frac{r^d}{n^d}} \frac{\lambda_{\max}(A_1)}{\sqrt{n^{d-2}}}$	$O(n^{d+1} + dn \log(n))$
Algorithm D	$\sqrt{\frac{r^d}{n^d}} \frac{\|\mathcal{A}\|_F}{\sqrt{n^{d-1}}}$	$O(n^d + dn \log(n))$

where the second inequality follows from that $\mathbf{w}$ is normalized, and the last one comes from Proposition 3.1. We also have from the definition of $\mathbf{x}$ that

$$\begin{aligned} \|\mathbf{x}\|^2 &= \sum_{k=1}^{m} (A^k \mathbf{w})^2 \geq (A^{\tilde{k}} \mathbf{w})^2 \\ &= \left\langle A^{\tilde{k}}, \frac{A^{\tilde{k}}}{\|A^{\tilde{k}}\|} \right\rangle^2 = \left\| A^{\tilde{k}} \right\|^2 \\ &\geq \frac{1}{m} \|A\|_F^2 , \end{aligned}$$

where the last inequality comes from the definition of $A^{\tilde{k}}$. Combining the above analysis gives the desired result. □

Theorem 3.7 *Let* $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$ *be generated by Algorithm* D. *Then it holds that*

$$\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0 \geq \sqrt{\frac{\prod_{j=1}^d r_j}{\prod_{j=1}^d n_j}} \frac{\|\mathcal{A}\|_F}{\sqrt{\prod_{j=1}^{d-1} n_j}} \geq \sqrt{\frac{\prod_{j=1}^d r_j}{\prod_{j=1}^d n_j}} \frac{v^{\text{opt}}}{\sqrt{\prod_{j=1}^{d-1} n_j}}.$$

Proof As the discussions above Theorem 3.6 we get $\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0 \geq \sqrt{\frac{r_d}{n_d}} \left\| A_{d-1}^\top \mathbf{x}_{d-1}^0 \right\|$. Using Lemma 3.2, for $2 \leq j \leq d-1$, we have

$$\left\| A_j^\top \mathbf{x}_j^0 \right\| \geq \sqrt{\frac{r_j}{n_j^2}} \left\| A_j \right\|_F = \sqrt{\frac{r_j}{n_j^2}} \left\| A_{j-1}^\top \mathbf{x}_{j-1}^0 \right\|.$$

In particular, from step 1 of the algorithm, $\left\| A_1^\top \mathbf{x}_1^0 \right\| \geq \sqrt{\frac{r_1}{n_1^2}} \|A_1\|_F = \sqrt{\frac{r_1}{n_1^2}} \|\mathcal{A}\|_F$. It is clear that $\|\mathcal{A}\|_F \geq \lambda_{\max}(A_1) \geq v^{\text{opt}}$. Combining the analysis gives the desired results. □

Before ending this section, we summarize the approximation ratio and computational complexity of the proposed algorithms in Table 1. For convenience we set $r_1 = \cdots = r_d$ and $n_1 = \cdots = n_d$.

Concerning the approximation ratio $\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0 / v^{\text{opt}}$, we see that if r is a constant, then the ratio of Algorithm A is also a constant, while those of other algorithms rely on n. This is the advantage of Algorithm A, compared with other algorithms. When $d = 2$, the ratios of Algorithms A and B are respectively $1/\sqrt{r}$ and r/n, which coincide with their matrix

counterparts; see, [2, Algorithms 1 and 2]. We also observe that Algorithm B generalizes the approximation algorithm and bound for dense tensor BR1Approx in [39, Theorem 5.1]: If sparsity is not required, i.e., $r = n$, the bound of Algorithm B boils down to those of [39, Theorem 5.1]. For Algorithm C, when $r = n$, or r is proportional to n up to a constant, the ratio reduces to $O(1/\sqrt{n^{d-2}})$, which recovers that of [13, Algorithm 1]. Although the ratio of Algorithm D is worse than that of Algorithm C with an additional factor $1/\sqrt{n}$, we shall also observe that the numerator of the bound of Algorithm D is $\|\mathcal{A}\|_F$, while that of Algorithm C is $\lambda_{\max}(A_1)$ (see Algorithm C for the definition of A_1), where the latter is usually much smaller than the former. Note that two randomized approximation algorithms as initializations were proposed in [32, Algorithms 3 and 4] (see its arXiv version), where the performance analysis was considered on structured tensors. It would be also interesting to study their approximation bound for general tensors. Concerning the computational complexity, we see that Algorithm D admits the lowest one in theory. This is also confirmed by our numerical observations that will be presented in the next section. Note that if the involved singular value problems in Algorithm C are solved by the Lanczos algorithm at k steps [7] where k is a user-defined parameter, then the computational complexity is $O(kn^d + dn\log(n))$.

Finally, We discuss that whether the approximation bounds are achieved. We only consider the cases that $r_j < n_j$. We first consider Algorithm C and take $d = 3$ as an example. In (3.13), achieving $\lambda_{\max}(A_2) = \frac{\|A_2\|_F}{\sqrt{n_2}}$ requires that $\operatorname{rank}(A_2) = n_2$ (assuming that $n_2 \le n_3$) and all the singular values are the same. If these are true, then $\|A_2^\top \mathbf{x}_2^0\| > \sqrt{\frac{r_2}{n_2}}\lambda_{\max}(A_2)$.[1] Thus the bound of Algorithm C may not be tight. The approximation ratio of Algorithm D relies on Lemma 3.2. However, there do not exist matrices achieving the approximation ratio $\sqrt{\frac{r}{m^2}}$ in Lemma 3.2, implying that the bound of Algorithm D may not be tight. For Algorithm A, if step 3 is not executed (use $\mathbf{e}_1^{\bar{i}_1}, \ldots, \mathbf{e}_{d-1}^{\bar{i}_{d-1}}, \mathbf{x}_d^0$ obtained in step 2 as the output), then the bound is tight: Consider $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times n_3}$ with $n_1 = n_2 = n_3 = n$ as an all-one tensor and let $1 < r_1 = r_2 = r_3 = r < n$; then $v^{\text{opt}} = r^{\frac{3}{2}}$. The generated feasible point is $\mathbf{x}_j^0 = \mathbf{e}_j^1$, $j = 1, 2$, and x_3^0 is the vector whose first r entries are $\frac{1}{\sqrt{r}}$ and the remaining ones are zero. Then $\mathcal{A}\mathbf{x}_1^0\mathbf{x}_2^0\mathbf{x}_3^0/v^{\text{opt}} = \frac{1}{r}$, achieving the bound. For Algorithm B, in case that $d = 3$ and if $\mathbf{x}_1^0$ is not updated by step 3 (use $\mathbf{e}_1^{\bar{i}_1}$ obtained in step 2 as the output), then the bound can be tight: Consider $\mathcal{A} \in \mathbb{R}^{4\times4\times4}$, with $\mathcal{A}(i,:,:) = A := \begin{bmatrix} 0&1&0&1\\ 0&1&0&1\\ 1&0&1&0\\ 1&0&1&0 \end{bmatrix}$, $i = 1, \ldots, 4$, and let $r_1 = r_2 = r_3 = 2$. Then $v^{\text{opt}} = 2\sqrt{2}$.[2] Applying Algorithm B to $\mathcal{A}$ yields $\bar{i}_1 = 1$, with $\bar{\mathbf{x}}_3 = [1\ 1\ 1\ 1]^\top/2$; then $\mathbf{x}_3^0 = [1\ 1\ 0\ 0]^\top/\sqrt{2}$, and $\mathbf{x}_2^0 = \mathbf{x}_3^0$. Finally, $\mathcal{A}\mathbf{e}_1^1\mathbf{x}_2^0\mathbf{x}_3^0/v^{\text{opt}} = \frac{1}{2\sqrt{2}} = \sqrt{\frac{r^2}{n^2 r}}$. In summary, the reason that why the bounds cannot be achieved is that although we derive the worst-case inequalities in every step of the analysis, after putting them together, the final approximation bounds might not be tight. How to improve them still need further research.

[1] Otherwise, from the analysis of Lemma 3.1, $\|A_2^\top \mathbf{x}_2^0\| = \sqrt{\frac{r_2}{n_2}}\lambda_{\max}(A_2)$ if and only if 1) $A_2^\top \bar{\mathbf{x}}_2 = \alpha A_2^\top \mathbf{x}_2^0$ for some $\alpha \in \mathbb{R}$, and 2) $\left\langle \bar{\mathbf{x}}_2, \mathbf{x}_2^0 \right\rangle = \sqrt{\frac{r_2}{n_2}}$. 2) holds if and only if every entry of $\bar{\mathbf{x}}_2$ takes the same value, which together with $r_2 < n_2$ implies that $\bar{\mathbf{x}}_2 - \alpha\mathbf{x}_2^0 \neq 0$; however, this and 1) lead to $\operatorname{rank}(A_2) < n_2$, deducing a contradiction.

[2] Due to the structure of $\mathcal{A}$, $\mathcal{A}\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3 = \langle \mathbf{e}, \mathbf{x}_1\rangle \cdot \mathbf{x}_2^\top A\mathbf{x}_3$ with $\mathbf{e} = [1\ 1\ 1\ 1]^\top$. It is clear that $\mathbf{x}_1^* = \frac{1}{\sqrt{2}}[1\ 1\ 0\ 0]^\top$. Denote $\mathbf{x}_2^* := \mathbf{x}_1^*$, and $\mathbf{x}_3^* := \frac{1}{\sqrt{2}}[0\ 1\ 0\ 1]^\top$. Since $(\mathbf{x}_2^*)^\top A\mathbf{x}_3^* = \lambda_{\max}(A) = 2$, $(\mathbf{x}_2^*, \mathbf{x}_3^*)$ is a maximizer to $\max_{\|\mathbf{x}_i\|=1, \|\mathbf{x}_i\|_0\le 2, i=2,3} \mathbf{x}_2^\top A\mathbf{x}_3$, and so $(\mathbf{x}_1^*, \mathbf{x}_2^*, \mathbf{x}_3^*)$ is a maximizer to (2.3), with $\mathcal{A}\mathbf{x}_1^*\mathbf{x}_2^*\mathbf{x}_3^* = 2\sqrt{2}$.

4 Numerical experiments

We evaluate the proposed algorithms in this section on synthetic and real data. All the computations are conducted on an Intel i7 CPU desktop computer with 16 GB of RAM. The supporting software is Matlab R2019a. Tensorlab [33] is employed for basic tensor operations.

Performance of approximation algorithms We first compare Algorithms A, B, C, and D on solving (2.3). The tensor is given by

$$\mathcal{A} = \sum_{i=1}^{R} \mathbf{x}_{1,i} \circ \cdots \circ \mathbf{x}_{d,i} \in \mathbb{R}^{n_1 \times \cdots \times n_d}, \tag{4.15}$$

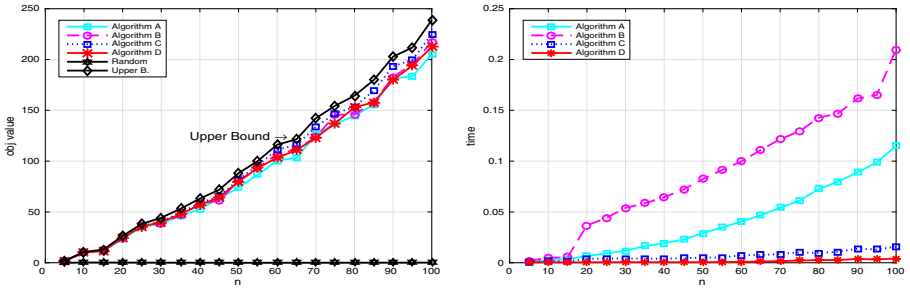
where $\mathbf{x}_{j,i} \in \mathbb{R}^{n_j}$, $i = 1, \ldots, R$, $j = 1, \ldots, d$, and we let $R = 10$. Here the vectors are first randomly drawn from the normal distribution, and then $sr = 70\%$ of the entries are randomly set to be zero. We set $d = 3, 4$, and let $n_1 = \cdots = n_d = n$ with n varying from 5 to 100. For each case, we randomly generated 50 instances, and the averaged results are presented. $r_j = \lfloor (1 - sr)n_j \rfloor$ for each j in (2.3). As a baseline, we also randomly generate feasible points $(\mathbf{x}_1^{\text{random}}, \ldots, \mathbf{x}_d^{\text{random}})$ and evaluate its performance. On the other hand, we easily see that

$$v^{\text{ub}} := \min\{\lambda_{\max}(A_{(1)}), \ldots, \lambda_{\max}(A_{(d)})\} \tag{4.16}$$

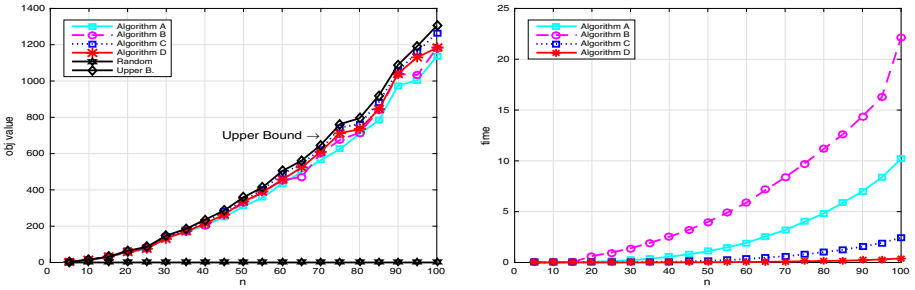
is an upper bound for problem (2.3), where $A_{(j)} = \texttt{reshape}(\mathcal{A}, n_j, \prod_{k \neq j}^{d} n_k)$ denotes the j-mode unfolding of $\mathcal{A}$. Thus we also evaluate v^{ub}. The results are depicted in Fig. 1, where the left panels show the curves of the objective values $\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0$ versus n of different algorithms, whose colors are respectively cyan (Algorithm A), magenta (Algorithm B), blue (Algorithm C), and red (Algorithm D); the curves of the random value $\mathcal{A}\mathbf{x}_1^{\text{random}} \cdots \mathbf{x}_d^{\text{random}}$ is in black with hexagram markers, while the curve of the upper bounds v^{ub} is in black with diamond markers. The right ones plot the curve of CPU time versus n.

From the left panels, we observe that the objective values generated by Algorithms A, B, C, and D are similar, where Algorithm C performs better; Algorithm D performs the second when $d = 4$, and it is comparable with Algorithm B when $d = 3$; Algorithm A gives the worst results, which may be that Algorithm A does not explore the structure of the problem as much as possible. We also observe that the objective values of all the algorithms are quite close to the upper bound (4.16), which demonstrates the effectiveness of the proposed algorithms. In fact, the ratio of $\frac{\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0}{v^{\text{ub}}}$ is in (0.7, 1), which is far better than the approximation ratios presented in Sect. 3. This implies that at least for this kind of tensors, the approximation ratios might be independent of the size of the tensor. The value $\mathcal{A}\mathbf{x}_1^{\text{random}} \cdots \mathbf{x}_d^{\text{random}}$ is close to zero (the curve almost coincides with the x-axis). Concerning the computational time, Algorithms D is the most efficient one, confirming the theoretical results in Table 1. Algorithm C is the second efficient one. Algorithms A and B do not perform well compared with Algorithms C and D, although their computational complexity is similar in theory. The reason may be because the first two algorithms require for-loop operations, which is time-consuming in Matlab.

We then consider fix $n = 100$ and vary the sparsity ratio sr from 10 to 90% of $\mathcal{A}$ in (4.15), and compare the performance of the four proposed algorithms. r_j in (2.3) is set to $\lfloor (1 - sr)n_j \rfloor$ correspondingly. The results of the objective values $\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0$ together with the upper bound are depicted in the left panels of Fig. 2, from which we still observe that all the algorithms are close to the upper bound (4.16); among them, Algorithm C is still slightly better than the other three, followed by Algorithms B and D. The CPU time is plotted in the right panels of Fig. 2, which still shows that Algorithm D is the most efficient one. In fact,



(a) Objective value and CPU time when $d = 3$.



(b) Objective value and CPU time when $d = 4$.

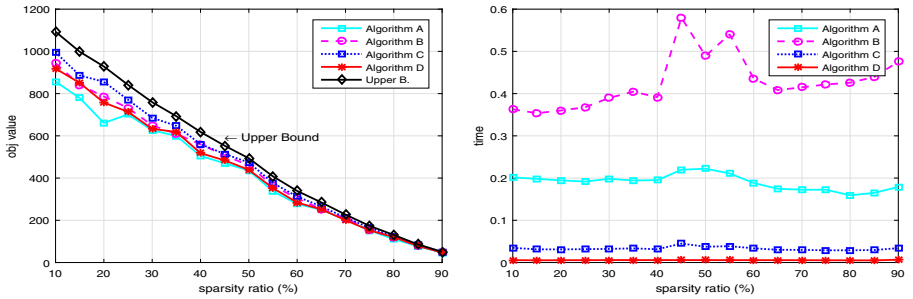
Fig. 1 Comparisons of Algorithms A, B, C, and D for solving (2.3) where $\mathcal{A}$ is given by (4.15). n varies from 5 to 100. Left panels: objective value $\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0$ versus n; right panels: CPU time

in average, Algorithm D is 5 times faster than Algorithm C, which ranks the second, and is about 80 times faster than Algorithm B, which is the slowest one.

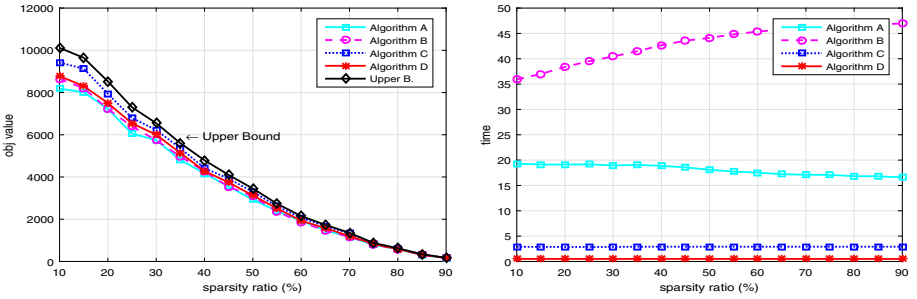
Overall, comparing with Algorithms A and B that find the solutions fiber by fiber, or slide by slide, Algorithm C admits hierarchical structures that take the whole tensor into account, and so it can explore the structure of the data tensor better. This may be the reason why Algorithm C is better than Algorithms A and B in terms of the effectiveness. When compared with Algorithm D, Algorithm C computes each $\mathbf{x}_j^0$ "optimally" via SVD, while Algorithm D computes $\mathbf{x}_j^0$ "sub-optimally" but more efficiently; this explains why Algorithms C performs better than Algorithm D. Concerning the efficiency, Algorithms A and B require for-loop operations, which is known to be slow in Matlab. This leads to that although the algorithms have similar computational complexity in theory (Algorithms B and C), after implementation, their performances are quite different.

Performance of approximation plus iterative algorithms In this part, we first use approximation algorithms to generate $(\mathbf{x}_1^0, \ldots, \mathbf{x}_d^0)$, and then use it as an initializer for iterative algorithms. The goal is to see if approximation algorithms can help in improving the solution quality of iterative algorithms. The iterative algorithm used for solving problem (2.3) is simply an alternating maximization method (termed AM in the sequel) with the scheme

$$\text{(AM)} \quad \mathbf{x}_j^{k+1} \in \arg\max \ \mathcal{A}\mathbf{x}_1^{k+1} \cdots \mathbf{x}_{j-1}^{k+1} \mathbf{x}_j \mathbf{x}_{j+1}^{k} \cdots \mathbf{x}_d^{k} \text{ s.t. } \|\mathbf{x}_j\| = 1, \|\mathbf{x}_j\|_0 \le r_j$$



(a) Objective value and CPU time when $d = 3$.

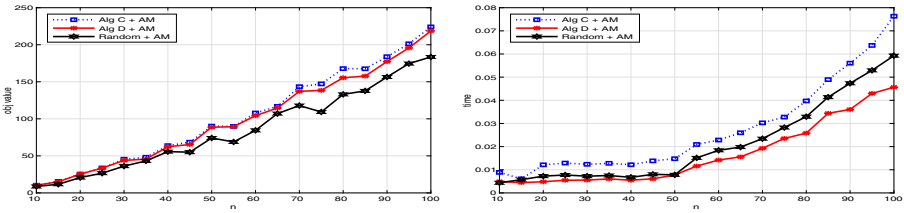


(b) Objective value and CPU time when $d = 4$.

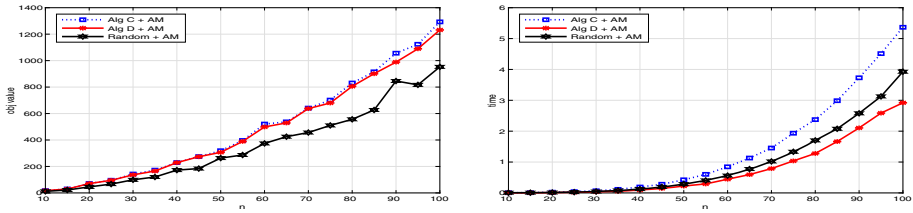
Fig. 2 Comparisons of Algorithms A, B, C, and D for solving (2.3) with fixed n and sparsity ratio varying from 10 to 90%. Figure 2a: $\mathcal{A} \in \mathbb{R}^{100\times 100\times 100}$; Fig. 2b: $\mathcal{A} \in \mathbb{R}^{100\times 100\times 100\times 100}$. Left panels: the objective values $\mathcal{A}\mathbf{x}_1^0 \cdots \mathbf{x}_d^0$ versus sparsity ratio; right panels: CPU time

for $j = 1, \ldots, d$ and $k = 1, 2, \ldots$. The stopping criterion used for AM is $\max_j\{\left\|\mathbf{x}_j^{k+1} - \mathbf{x}_j^k\right\|\} \leq 10^{-5}$, or $k \geq 2000$. We employ Algorithms C and D in this part, and denote the approximation plus iterative algorithms as Algorithm C + AM and D + AM in the sequel. As a baseline, we also evaluate AM initialized by randomly generated feasible point $(\mathbf{x}_1^{\text{random}}, \ldots, \mathbf{x}_d^{\text{random}})$, which is generated the same as the previous part. This algorithm is denoted as Random + AM. The data tensors are also (4.15), where 50 instances are randomly generated for each n. $r_j = \lfloor 0.3n_j \rfloor$. The objective values $\mathcal{A}\mathbf{x}_1^{\text{out}} \cdots \mathbf{x}_d^{\text{out}}$ and CPU time for third- and fourth-order tensors with n varying from 10 to 100 are plotted in Fig. 3a, b, where $(\mathbf{x}_1^{\text{out}}, \ldots, \mathbf{x}_d^{\text{out}})$ is the output of AM. Here the CPU time counts both that of approximation algorithms and AM. Figure 3c depicts the number of iterations of AM initialized by different strategies. Algorithm C + AM is in blue, D + AM is in red, and Random + AM is in black.

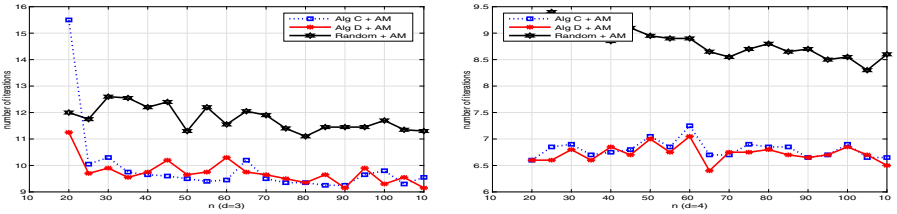
In terms of objective value, we see from Fig. 3a, b that Algorithm C + AM performs the best, while Algorithm D + AM is slightly worse. Both of them are better than Random + AM, demonstrating that approximation algorithms can indeed help in improving the solution quality of iterative algorithms. Considering the efficiency, Algorithm D + AM is the best among the three, followed by Random + AM. This further demonstrates the advantage of approximation algorithms. In fact, from Fig. 3c, we can see that both Algorithm C and D can help in reducing the number of iterations of AM. However, as we have observed in the previous part, Algorithm C is more time-consuming than Algorithm D, leading to that Algorithm C + AM is the slowest one.



(a) Objective value and CPU time when $d = 3$.



(b) Objective value and CPU time when $d = 4$.



(c) Number of iterations. Left: $d = 3$; right: $d = 4$.

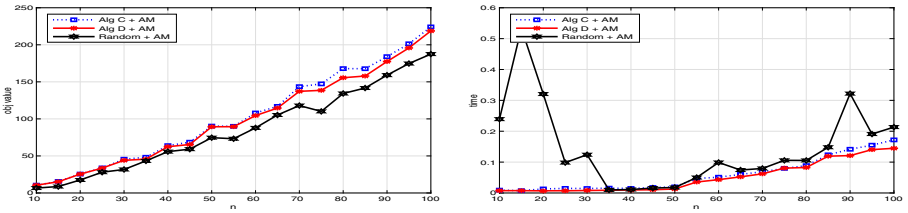
Fig. 3 Performances of Algorithm C + AM, Algorithm D + AM and Random + AM for solving (2.3) where $\mathcal{A}$ is given by (4.15). n varies from 10 to 100. The CPU time counts both that of approximation algorithms and AM

We also try to use Algorithms C and D to initialize AM for ℓ_1 regularized model [1]:

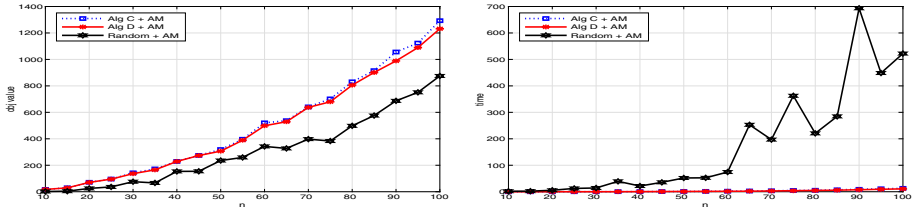
$$\max \ \langle \mathcal{A}, \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_d \rangle - \sum_{j=1}^{d} \rho_j \|\mathbf{x}_j\|_1 \ \text{ s.t. } \ \|\mathbf{x}_j\| \le 1, \ 1 \le j \le d, \tag{4.17}$$

where $\rho_j = 0.2$ is set in the experiment. The algorithms are denoted as Algorithm C + AM (ℓ_1), Algorithm D + AM (ℓ_1) and Random + AM (ℓ_1). The results are plotted in Fig. 4, from which we observe similar results as those of Fig. 3; in particular, the efficiency of approximation algorithms plus AM (ℓ_1) is significantly better than Random + AM (ℓ_1), and their number of iterations is much more stable.

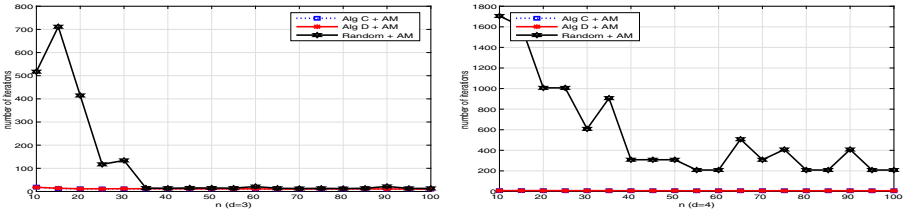
Overall, based on the observations of this part, compared with random initializations, iterative algorithms initialized by approximation solutions generated by approximation algorithms is superior both in terms of the solution quality and the running time.



(a) $d = 3$. Left: $\mathcal{A}\mathbf{x}_1^{\text{out}} \cdots \mathbf{x}_d^{\text{out}}$ versus n; right: CPU time versus n.



(b) $d = 4$. Left: $\mathcal{A}\mathbf{x}_1^{\text{out}} \cdots \mathbf{x}_d^{\text{out}}$ versus n; right: CPU time versus n.



(c) Number of iterations versus n. Left: $d = 3$; right: $d = 4$.

Fig. 4 Performances of Algorithm C + AM (ℓ_1), Algorithm D + AM (ℓ_1) and Random + AM (ℓ_1) for solving (4.17) where $\mathcal{A}$ is given by (4.15). n varies from 10 to 100

Sparse tensor clustering Tensor clustering aims to cluster matrix or tensor samples into their underlying groups: for N samples $\mathcal{A}_i \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, $i = 1, \ldots, N$ with $d \geq 2$ and given clusters $K \geq 2$, a clustering $\psi(\cdot)$ is defined as a mapping $\psi : \mathbb{R}^{n_1 \times \cdots \times n_d} \to \{1, \ldots, K\}$. Several tensor methods have been proposed for tensor clustering; see, e.g., [26, 31, 32]. Usually, one can first perform a dimension reduction to the samples by means of (sparse) tensor decomposition, and then use classic methods such as K-means to the reduced samples for clustering. Here, we use a deflation method for tensor decomposition, including the proposed approximation algorithms as initialization procedures for AM as subroutines. The whole method for tensor clustering is presented in Algorithm STC, which has some similarities to [31, Algorithm 2]. We respectively use STC (A), STC (B), STC (C), STC (D) to distinguish AM involved in STC initialized by different approximation algorithms. We also denote STC (Random) as AM involved in STC with random initializations.

Algorithm $(\psi(\mathcal{A}_1), \ldots, \psi(\mathcal{A}_N)) = \text{sparse\_clustering}(\mathcal{A}_1, \ldots, \mathcal{A}_N)$ (STC)

1. Define $\mathcal{T} \in R^{n_1 \times \cdots \times n_d \times N}$ with $\mathcal{T}(:, \ldots, :, i) = \mathcal{A}_i, i = 1, \ldots, N$.
2. Apply a deflation method with rank-R to $\mathcal{T}$, which employs Algorithm A, B, C, or D)+ AM to the residual tensors $\mathcal{T}, \ldots, \mathcal{T} - \sum_{l=1}^{m} \alpha_l \mathbf{x}_1^l \circ \cdots \circ \mathbf{x}_{d+1}^l, \ldots$, with resulting rank-1 terms $\mathbf{x}_1^m \circ \cdots \circ \mathbf{x}_{d+1}^m$ and weights $\alpha_m = (\mathcal{T} - \sum_{l=1}^{m-1} \alpha_l \mathbf{x}_1^l \cdots \mathbf{x}_{d+1}^l)\mathbf{x}_1^m \cdots \mathbf{x}_{d+1}^m, m = 1, \ldots, R$. Write $X_j := [\mathbf{x}_j^1, \ldots, \mathbf{x}_j^R]$, $j = 1, \ldots, d+1$ and $\boldsymbol{\alpha} := [\alpha_1, \ldots, \alpha_R]^\top$.
3. Denote $\hat{X}_{d+1} = [\alpha_1 \mathbf{x}_{d+1}^1, \ldots, \alpha_R \mathbf{x}_{d+1}^R] \in \mathbb{R}^{N \times R}$; here the i-th row of $\hat{X}_{d+1}$, denoted as $\hat{\mathbf{u}}_i$, is regarded as the reduced sample of $\mathcal{A}_i$. Apply K-means to clustering $\{\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_N\} \to \hat{\psi}(\hat{\mathbf{u}}_j), j = 1, \ldots, N$.
4. Return the cluster assignment of $\{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$: $(\psi(\mathcal{A}_1), \ldots, \psi(\mathcal{A}_N)) = (\hat{\psi}(\hat{\mathbf{u}}_1), \ldots, \hat{\psi}(\hat{\mathbf{u}}_N))$.

Denote ψ_0 as the true clustering mapping and let $|S|$ represent the cardinality of a set S. The following metric is commonly used to measure clustering performance [34]:

$$\text{cluster err.} := \binom{N}{2}^{-1} |\{i, j\} : (\psi(\mathcal{A}_i) = \psi(\mathcal{A}_j)) \neq (\psi_0(\mathcal{A}_i) = \psi_0(\mathcal{A}_j)), i < j|.$$

Synthetic data This experiment is similar to [31, Sect. 5.2]. Consider N sparse matrix samples $\{A_i \in \mathbb{R}^{n_1 \times n_2}, i = 1, \ldots, N\}$ with $n_1 = n_2 = 20$ as follows:

$$A_1 = \cdots = A_{\frac{N}{4}} = \mu^3 \cdot \begin{bmatrix} \Sigma & & \\ & -\Sigma & \\ & & \mathbf{0} \end{bmatrix}, A_{\frac{N}{4}+1} = \cdots = A_{\frac{N}{2}} = \mu^3 \cdot \begin{bmatrix} \Sigma & & \\ & \Sigma & \\ & & \mathbf{0} \end{bmatrix}$$

$$A_{\frac{N}{2}+1} = \cdots = A_{\frac{3N}{4}} = \mu^3 \cdot \begin{bmatrix} -\Sigma & & \\ & \Sigma & \\ & & \mathbf{0} \end{bmatrix}, A_{\frac{3N}{4}+1} = \cdots = A_N = \mu^3 \cdot \begin{bmatrix} -\Sigma & & \\ & -\Sigma & \\ & & \mathbf{0} \end{bmatrix},$$

where $\Sigma = \mathbf{v}\mathbf{v}^\top \in \mathbb{R}^{4 \times 4}$ with $\mathbf{v} = [1, -1, 0.5, -0.5]^\top$, and $\mathbf{0}$ is a zero matrix of size 12×12. We stack these samples into a third order tensor $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times N}$ with $\mathcal{T}(:, :, i) = A_i$. We apply STC (A), STC (B), STC (C), STC (D), and the vanilla K-means to the tensor $\mathcal{T}^* = \frac{\mathcal{T}}{||\mathcal{T}||_F} + \sigma \frac{\mathcal{E}}{||\mathcal{E}||_F}$ where $\mathcal{E}$ is a noisy tensor and σ is the noise level. Here vanilla K-means stands for directly applying K-means to the vectorizations of A_i's without tensor methods. We vary the sample number $N = \{20, 40\}$ and noise level $\sigma = \{0.1, 0.5, 0.9\}$, and set $\mu = 0.5$. To select parameters, we apply a similar Bayesian Information Criterion as [1]: For a set of parameter combinations defined as the cartesian product of the set $\mathfrak{R}$ of given ranks and the sets $\mathfrak{r}$ of tuning parameters, namely, $\mathfrak{R} \times \mathfrak{r} = \{(R, \boldsymbol{r}) | R \in \mathfrak{R}, \boldsymbol{r} \in \mathfrak{r}\}$, we select $(R^*, \boldsymbol{r}^*) = \arg\min_{\mathfrak{R} \times \mathfrak{r}} BIC(R, \boldsymbol{r})$ with

$$BIC(R, \boldsymbol{r}) := \log\left(\frac{\|\mathcal{T} - \sum_{l=1}^{R} \alpha_l \mathbf{x}_1^l \circ \cdots \circ \mathbf{x}_{d+1}^l\|_F^2}{n_1 \ldots n_d N}\right) + \frac{\log(\prod_{j=1}^{d} n_j N)}{\prod_{j=1}^{d} n_j N} \sum_{l=1}^{R} \sum_{j=1}^{d+1} \|\mathbf{x}_j^l\|_0, \tag{4.18}$$

where α_l and $\mathbf{x}_j^l$ are computed by Algorithm STC.

For tuning procedure, we set $\mathfrak{R} = \{4, 6\}$ and $\mathfrak{r} = \{(7, 7, N), (8, 8, N)\}$. For the number K of clusters, we employ the widely used `evalclusters` function in Matlab which evaluate each proposed number of clusters in a set $\{K_1, \cdots, K_m\}$ and select the smallest number of clusters satisfying $Gap(K) \geq Gap(K+1) - SE(K+1)$, where $Gap(K)$ is the gap value for the clustering solution with K clusters, and $SE(K+1)$ is the standard error of the clustering solution with $K+1$ clusters. The results are reported in Table 2 averaged over 50 instances in each case, where bold means that the cluster error is the lowest among all the methods.

Table 2 Sparse tensor clustering via STC (A), STC (B), STC (C), STC (D) on synthetic data with different size N and noise level σ

N	σ	STC (A)		STC (B)		STC (C)		STC (D)		Vanilla K-means	
		Cluster err.	Time	Cluster err.	Time	Cluster err.	Time	Cluster err.	Time	Cluster err.	Time
20	0.1	**0.00E+00**	2.72E−01	**0.00E+00**	5.08E−01	**0.00E+00**	2.41E−01	**0.00E+00**	2.08E−01	9.37E−03	2.10E−03
20	0.5	3.05E−03	3.03E−01	6.53E−03	5.47E−01	**0.00E+00**	2.83E−01	**0.00E+00**	2.27E−01	2.53E−02	1.82E−03
20	0.9	**1.22E−02**	3.72E−01	6.53E−03	5.17E−01	**1.22E−02**	2.83E−01	3.05E−03	2.17E−01	5.21E−02	1.82E−03
40	0.1	**0.00E+00**	3.98E−01	**0.00E+00**	7.10E−01	**0.00E+00**	4.26E−01	**0.00E+00**	5.36E−01	3.10E−03	2.35E−03
40	0.5	3.18E−03	4.26E−01	6.28E−03	6.43E−01	**0.00E+00**	3.85E−01	2.97E−03	3.54E−01	2.26E−02	2.39E−03
40	0.9	3.21E−03	3.77E−01	9.08E−03	6.69E−01	**3.10E−03**	3.57E−01	3.18E−03	3.13E−01	5.33E−02	2.50E−03

"Cluster err." represents the cluster error; "time" denotes the computational time (s)

Table 3 Sparse tensor clustering via STC (D), STC (Random) and vanilla K-means on COIL-20 with different K

K	STC (D)		STC (Random)		vanilla K-means	
	Cluster err.	Time	Cluster err.	Time.	Cluster err.	Time
5	**1.36E−01**	1.62E+02	1.37E−01	1.70E+02	1.46E−01	2.21E−01
6	**1.45E−01**	1.90E+02	1.50E−01	2.02E+02	1.51E−01	2.79E−01
7	**1.47E−01**	2.23E+02	1.49E−01	2.46E+02	1.56E−01	3.53E−01
8	**1.48E−01**	2.63E+02	1.49E−01	2.84E+02	1.55E−01	3.91E−01
9	**1.48E−01**	2.93E+02	1.49E−01	2.97E+02	1.57E−01	4.40E−01
10	1.13E−01	3.09E+02	**1.11E−01**	3.43E+02	1.24E−01	5.39E−01

Table 2 shows that the cluster error of STC with any approximation algorithm is smaller than that of the vanilla K-means in all cases and is even zero in some case when the noise level is not high, where the best one is STC (C), followed by STC (D). This shows the accuracy and robustness of our method. Considering the CPU time, among the four tensor methods, STC (D) is the most efficient one, followed by STC (C), while STC (B) needs more time than the other three. However, the computational time of tensor methods is not as good as that of the vectorized K-means, which is because of the increasing cost of the tuning procedure via (4.18) with a pre-specified set of parameter combinations.

Real data We test the clustering performance of STC (D), STC (Random), and the vanilla K-means on Columbia Object Image Library COIL-20 [23] for image clustering. The data contains 20 objects viewed from varying angles and each image is of size 128×128. The images are in grayscale, and the background of the objects is black, resulting in that the images can be seen as sparse matrices. In this experiment, we consider $K = \{5, 6, 7, 8, 9, 10\}$ objects and pick up 36 images from each object for clustering, giving $\mathcal{T} \in \mathbb{R}^{128\times128\times36K}$. We still tune parameters of Algorithm STC via (4.18) and the `evalclusters` function in Matlab, and set $\mathfrak{R} = \{40, 45\}$ and $\mathfrak{r} = \{(75, 75, 36K), (80, 80, 36K)\}$. The experiment has been repeated 20 times for each K, and the averaged results are shown in Table 3.

Table 3 shows that the cluster error of tensor methods is still better than that of the vanilla K-means, while STC (D) is slightly better than STC (Random). On the other hand, the computational time of STC (D) is less than that of STC (Random). However, it still needs more time than that of the vanilla K-means. There are two possible reasons for this phenomenon: first, it takes more time to tune parameters via (4.18); and second, our approach deals directly with data tensors instead of reshaping them into vectors, which preserves their intrinsic structures but naturally increases the computational complexity at the same time.

Overall, tensor methods armed with an approximation algorithm to generate good initializers show its ability to better exploring the data structure for clustering, and it would be interesting to further improve the efficiency.

5 Conclusions

Sparse tensor BR1Approx problem can be seen as a sparse generalization of the dense tensor BR1Approx problem and a higher-order extension of the matrix BR1Approx problem. In the literature, little attention was paid to approximation algorithms for sparse tensor BR1Approx. To fill this gap, four approximation algorithms were developed in this work, which are of

low computational complexity, easily implemented, and all admit theoretical guaranteed approximation bounds. Some of the proposed algorithms and the associated aproximation bounds generalize their matrix or dense tensor counterparts, while Algorithm D, which is the most efficient one, is even new when reducing to the matrix or dense tensor cases. Numerical experiments on synthetic as well as real data showed the effectiveness and efficiency of the developed algorithms; in particular, we observed that compared with random initializations, our algorithms can improve the performance of iterative algorithms for solving the problem in question. Possible future work is to design algorithms with better approximation ratio; following [12, Theorem 4.3], we conjecture that the best ratio might be $O(\sqrt{\prod_{j=1}^{d}\frac{r_j}{n_j}}\cdot\sqrt{\prod_{j=1}^{d-2}\frac{\ln n_j}{n_j}})$. On the other hand, Qi defined and studied the best rank-1 approximation ratio of a tensor space [27]; it would be interesting to extend this notion to the sparse best rank-1 approximation setting and study its bounds.

Acknowledgements We thank the editor and the anonymous reviewers for their insightful comments and suggestions that helped improve this manuscript. This work was supported by the National Natural Science Foundation of China Grants 11801100 and 12171105, the Fok Ying Tong Education Foundation Grant 171094, and the Innovation Project of Guangxi Graduate Education Grant YCSW2020055. The datasets generated during and/or analysed during the current study are available from [23], and from the corresponding author on reasonable request.

References

1. Allen, G.I.: Sparse higher-order principal components analysis. In: International Conference on Machine Learning, pp. 27–36 (2012)
2. Chan, S.O., Papailliopoulos, D., Rubinstein, A.: On the approximability of sparse PCA. In: Conference on Learning Theory, pp. 623–646 (2016)
3. Chi, E.C., Kolda, T.G.: On tensors, sparsity, and nonnegative factorizations. SIAM J. Matrix Anal. Appl. **33**(4), 1272–1299 (2012)
4. Cichocki, A.: Era of big data processing: a new approach via tensor networks and tensor decompositions (2014). arXiv:1403.2048
5. Cichocki, A., Mandic, D., De Lathauwer, L., Zhou, G., Zhao, Q., Caiafa, C., Phan, H.A.: Tensor decompositions for signal processing applications: from two-way to multiway component analysis. IEEE Signal Process. Mag. **32**(2), 145–163 (2015)
6. Comon, P.: Tensors: a brief introduction. IEEE Signal Process. Mag. **31**(3), 44–53 (2014)
7. Comon, P., Golub, G.H.: Tracking a few extreme singular values and vectors in signal processing. Proc. IEEE **78**(8), 1327–1343 (1990)
8. Comon, P., Luciani, X., De Almeida, A.: Tensor decompositions, alternating least squares and other tales. J. Chemometr. **23**(7–8), 393–405 (2009)
9. d'Aspremont, A., El Ghaoui, L., Jordan, M.I., Lanckriet, G.R.G.: A direct formulation for sparse PCA using semidefinite programming. SIAM Rev. **49**(3), 434–448 (2007)
10. De Lathauwer, L., De Moor, B., Vandewalle, J.: On the best rank-1 and rank-(R_1, R_2, ..., R_n) approximation of higer-order tensors. SIAM J. Matrix Anal. Appl. **21**, 1324–1342 (2000)
11. d'Aspremont, A., Bach, F., El Ghaoui, L.: Approximation bounds for sparse principal component analysis. Math. Progam. **148**(1–2), 89–110 (2014)
12. He, S., Jiang, B., Li, Z., Zhang, S.: Probability bounds for polynomial functions in random variables. Math. Oper. Res. **39**(3), 889–907 (2014)
13. He, S., Li, Z., Zhang, S.: Approximation algorithms for homogeneous polynomial optimization with quadratic constraints. Math. Program. Ser. B **125**, 353–383 (2010)
14. Hillar, C.J., Lim, L.H.: Most tensor problems are NP-hard. J. ACM **60**(6), 45:1–45:39 (2013)
15. Jiang, B., Ma, S., Zhang, S.: Tensor principal component analysis via convex optimization. Math. Program. Ser. A **150**, 423–457 (2015)
16. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Rev. **51**, 455–500 (2009)
17. Kolda, T.G., Bader, B.W., Kenny, J.P.: Higher-order web link analysis using multilinear algebra. In: Fifth IEEE International Conference on Data Mining, pp. 242–249. IEEE (2005)

18. Kolda, T.G., Mayo, J.R.: Shifted power method for computing tensor eigenpairs. SIAM J. Matrix Anal. Appl. **32**(4), 1095–1124 (2011)
19. Lim, L.H.: Singular values and eigenvalues of tensors: a variational approach. In: 2005 1st IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, vol. 1, pp. 129–132 (2005)
20. Luss, R., Teboulle, M.: Conditional gradient algorithms for rank-one matrix approximations with a sparsity constraint. SIAM Rev. **55**(1), 65–98 (2013)
21. Madrid-Padilla, O.H., Scott, J.: Tensor decomposition with generalized lasso penalties. J. Comput. Graph. Stat. **26**(3), 537–546 (2017)
22. Magdon-Ismail, M.: NP-hardness and inapproximability of sparse PCA. Inf. Process. Lett. **126**, 35–38 (2017)
23. Nene, S.A., Nayar, S.K., Murase, H.: Columbia object image library (COIL-20). Technical Report CUCS-005-96 (1996). https://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php
24. Ng, M.K.P., Li, X., Ye, Y.: Multirank: co-ranking for objects and relations in multi-relational data. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1217–1225 (2011)
25. Nie, J., Wang, L.: Semidefinite relaxations for best rank-1 tensor approximations. SIAM J. Matrix Anal. Appl. **35**(3), 1155–1179 (2014)
26. Papalexakis, E.E., Sidiropoulos, N.D., Bro, R.: From K-means to higher-way co-clustering: multilinear decomposition with sparse latent factors. IEEE Trans. Signal Process. **61**(2), 493–506 (2012)
27. Qi, L.: The best rank-one approximation ratio of a tensor space. SIAM J. Matrix Anal. Appl. **32**(2), 430–442 (2011)
28. Qi, L., Chen, H., Chen, Y.: Tensor Eigenvalues and Their Applications, vol. 39. Springer, Berlin (2018)
29. Sidiropoulos, N.D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E.E., Faloutsos, C.: Tensor decomposition for signal processing and machine learning. IEEE Trans. Signal Process. **65**(13), 3551–3582 (2017)
30. Sun, W.W., Li, L.: Store: sparse tensor response regression and neuroimaging analysis. J. Mach. Learn. Res. **18**(1), 4908–4944 (2017)
31. Sun, W.W., Li, L.: Dynamic tensor clustering. J. Am. Stat. Assoc. **114**(528), 1894–1907 (2019)
32. Sun, W.W., Lu, J., Liu, H., Cheng, G.: Provable sparse tensor decomposition. J. R. Stat. Soc. Ser. B **79**(3), 899–916 (2017)
33. Vervliet, N., Debals, O., Sorber, L., Van Barel, M., De Lathauwer, L.: Tensorlab 3.0 (2016). http://www.tensorlab.net
34. Wang, J.: Consistent selection of the number of clusters via crossvalidation. Biometrika **97**(4), 893–904 (2010)
35. Wang, Y., Dong, M., Xu, Y.: A sparse rank-1 approximation algorithm for high-order tensors. Appl. Math. Lett. **102**, 106140 (2020)
36. Witten, D.M., Tibshirani, R., Hastie, T.: A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. Biostatistics **10**(3), 515–534 (2009)
37. Yang, Y., Feng, Y., Huang, X., Suykens, J.A.K.: Rank-1 tensor properties with applications to a class of tensor optimization problems. SIAM J. Optim. **26**(1), 171–196 (2016)
38. Zhang, A., Han, R.: Optimal sparse singular value decomposition for high-dimensional high-order data. J. Am. Stat. Assoc. **114**(528), 1708–1725 (2019)
39. Zhang, X., Qi, L., Ye, Y.: The cubic spherical optimization problems. Math. Comput. **81**(279), 1513–1525 (2012)
40. Zou, H., Hastie, T., Tibshirani, R.: Sparse principal component analysis. J. Comput. Graph. Stat. **15**(2), 265–286 (2006)