Accelerated Nonnegative Tensor Completion via Integer Programming

Wenhao Pan\textsuperscript{1,*}, Anil Aswani\textsuperscript{2} and Chen Chen\textsuperscript{3}

\textsuperscript{1}Departments of Statistics and EECS, University of California, Berkeley, Berkeley, California, USA
\textsuperscript{2}Industrial Engineering and Operations Research, University of California, Berkeley, Berkeley, California, USA
\textsuperscript{3}Industrial and Systems Engineering, The Ohio State University, Columbus, Ohio, USA

Correspondence*: Wenhao Pan
wenhao1102@berkeley.edu

ABSTRACT

The problem of tensor completion has applications in healthcare, computer vision, and other domains. However, past approaches to tensor completion have faced a tension in that they either have polynomial-time computation but require exponentially more samples than the information-theoretic rate, or they use fewer samples but require solving NP-hard problems for which there are no known practical algorithms. A recent approach, based on integer programming, resolves this tension for nonnegative tensor completion. It achieves the information-theoretic sample complexity rate and deploys the Blended Conditional Gradients algorithm, which requires a linear (in numerical tolerance) number of oracle steps to converge to the global optimum. The tradeoff in this approach is that, in the worst case, the oracle step requires solving an integer linear program. Despite this theoretical limitation, numerical experiments show that this algorithm can, on certain instances, scale up to 100 million entries while running on a personal computer. The goal of this paper is to further enhance this algorithm, with the intention to expand both the breadth and scale of instances that can be solved. We explore several variants that can maintain the same theoretical guarantees as the algorithm, but offer potentially faster computation. We consider different data structures, acceleration of gradient descent steps, and the use of the Blended Pairwise Conditional Gradients algorithm. We describe the original approach and these variants, and conduct numerical experiments in order to explore various tradeoffs in these algorithmic design choices.

Keywords: Tensor Completion, Integer Programming, Conditional Gradient Method, Acceleration, Benchmarking

1 INTRODUCTION

A tensor is a multi-dimensional array or a generalized matrix. $\psi$ is called an order-$p$ tensor if $\psi \in \mathbb{R}^{r_1 \times \cdots \times r_p}$, where $r_i$ is the length of $\psi$ in its $i$-th dimension. For example, an RGB image with a size of 30 by 30 pixels is an order-3 tensor in $\mathbb{R}^{30 \times 30 \times 3}$. Since tensors and matrices are closely related, many matrix problems can be naturally generalized to tensors, such as computing a matrix norm and decomposing a matrix. However,
the tensor generalization of such problems can be substantially more challenging in terms of computation [Hillar and Lim 2013].

Like matrix completion, tensor completion uses the observed entries of a partially observed tensor $\psi$ to interpolate the missing entries with a restriction on the rank of the interpolated tensor. The purpose of the rank restriction is to restrict the degree of freedom of the missing entries [Song et al. 2019], e.g. avoiding overfitting. Without this rank restriction, the tensor completion problem is ill-posed because there are too many degrees of freedom to be constrained by the available data. Tensor completion is a versatile model with many important applications in social sciences [Tan et al. 2013], healthcare [Gandy et al. 2011], computer vision [Liu et al. 2013], and many other domains.

In the past decade, there have been major advances in matrix completion [Zhang et al. 2019]. However, for general tensor completion there remains a critical tension. Past approaches either have polynomial-time computation but require exponentially more samples than the information-theoretic rate [Gandy et al. 2011; Mu et al. 2014; Barak and Moitra 2016; Montanari and Sun 2018], or they achieve the information-theoretic rate but require solving NP-hard problems for which there are no known practical numerical algorithms [Chandrasekaran et al. 2012; Yuan and Zhang 2016, 2017; Rauhut and Stojanac 2021].

We note that, aside from matrix completion, for some special cases of tensor completion there are numerical algorithms that can achieve the information-theoretic rate; for instance, nonnegative rank-1 [Aswani 2016] tensors, or symmetric orthogonal [Rao et al. 2015] tensors. In this paper, we focus on a new approach proposed by [Bugg et al. 2022], designed for entrywise-nonnegative tensors, which naturally exist in applications such as image demosaicing. The authors defined a new norm for nonnegative tensors by using the gauge of a specific 0-1 polytope that they constructed. By using this gauge norm, their approach achieved the information-theoretic rate in terms of sample complexity, although the resulting problem is NP-hard to solve. Nevertheless, a practical approach was attained: as the norm is defined by using a 0-1 polytope, the authors embedded integer linear optimization within the Blended Conditional Gradients (BCG) algorithm [Braun et al. 2019], a variant of the Frank-Wolfe algorithm, to construct a numerical computation algorithm that required a linear (in numerical tolerance) number of oracle steps to converge to the global optimum.

This paper proposes several acceleration techniques for the original numerical computation algorithm created by [Bugg et al. 2022]; multiple techniques are tested in combination to evaluate the best configuration for overall speedup. The motivation is to further improve the implementation on large-scale problems. Our variants can maintain the same theoretical guarantees as the original algorithm, while offering potential speedups. Indeed, our experiments demonstrate that such speedups can be attained consistently across a range of problem instances. This paper also fully describes the original numerical computation algorithm and its coding implementation, which was omitted in [Bugg et al. 2022].

We summarize preliminary material and introduce the framework and theory of [Bugg et al. 2022]'s nonnegative tensor completion approach in Section 2. Then, we describe the computation algorithm of their approach in Section 3 and our acceleration techniques in Section 4. Section 5 presents numerical experiment results.

## 2 PRELIMINARIES

Given an order-$p$ tensor $\psi \in \mathbb{R}^{r_1 \times \cdots \times r_p}$, we refer to its entry with indices $x = (x_1, \ldots, x_p)$ as $\psi_x := \psi_{x_1, \ldots, x_p}$. $x_i \in [r_i]$ is the value of the $i$-th index where $[r_i] := \{1, \ldots, r_i\}$. We also define $\rho := \sum_i r_i$. 

Frontiers 2
\[ \pi := \prod_i r_i, \text{ and } \mathcal{R} = [r_1] \times \cdots \times [r_p]. \] The probability simplex \( \Delta^k := \text{conv}\{e_1, \ldots, e_k\} \) is the convex hull of the coordinate vectors in dimension \( k \).

A nonnegative rank-1 tensor \( \psi \) is defined as \( \psi := \theta^{(1)} \otimes \cdots \otimes \theta^{(p)} \) where \( \theta^{(k)} \in \mathbb{R}^{r_k}_{+} \) are nonnegative vectors. Its entry \( \psi_{x} = \prod_{k=1}^{p} \theta_{x_k}^{(k)} \).

\( \text{Bugg et al. [2022]} \) defined the ball of nonnegative rank-1 tensors whose maximum entry is \( \lambda \in \mathbb{R}_{+} \) to be

\[ B_{\lambda} = \{ \psi : \psi_x = \lambda \cdot \prod_{k=1}^{p} \theta_{x_k}^{(k)}, \theta_{x_k}^{(k)} \in [0, 1], \text{ for } x \in \mathcal{R} \} \] (1)

so the nonnegative rank of nonnegative tensor is

\[ \text{rank}_+(\psi) = \min \{ q | \psi = \sum_{k=1}^{q} \psi^k, \psi^k \in B_{\infty} \text{ for } k \in [q] \} \] (2)

where \( B_{\infty} = \lim_{\lambda \to \infty} B_{\lambda} \). For a \( \lambda \in \mathbb{R}_{+} \), consider a finite set of points

\[ S_{\lambda} = \{ \psi : \psi_x = \lambda \cdot \prod_{k=1}^{p} \theta_{x_k}^{(k)}, \theta_{x_k}^{(k)} \in \{0, 1\}, \text{ for } x \in \mathcal{R} \} \] (3)

\( \text{Bugg et al. [2022]} \) established the following connection between \( B_{\lambda} \) and \( S_{\lambda} \):

\[ C_{\lambda} := \text{conv}(B_{\lambda}) = \text{conv}(S_{\lambda}) \] (4)

where \( C_{\lambda} \) is the nonnegative tensor polytope. \( \text{Bugg et al. [2022]} \) also presented three implications of this result that are useful to their nonnegative tensor completion approach. First, \( C_{\lambda} \) is a polytope. Second, the elements of \( S_{\lambda} \) are the vertices of \( C_{\lambda} \). Third, the following relationships hold: \( B_{\lambda} = \lambda B_1, S_{\lambda} = \lambda S_1, \) and \( C_{\lambda} = \lambda C_1 \).

### 2.1 Norm for Nonnegative Tensors

Key to the theoretical guarantee and numerical computation of \( \text{Bugg et al. [2022]} \)’s approach is their construction of a new norm for nonnegative tensors using a gauge (or Minkowski functional) construction.

**Definition 2.1 (Bugg et al. [2022]).** The function defined as

\[ \| \psi \|_{+} := \inf \{ \lambda \geq 0 | \psi \in \lambda C_1 \} \] (5)

is a norm for nonnegative tensors \( \psi \in \mathbb{R}^{r_1 \times \cdots \times r_p}_{+} \).

This norm has an important property that it can be used as a convex surrogate for tensor rank \( \text{Bugg et al. [2022]} \). In other words, if \( \psi \) is a nonnegative tensor, then we have \( \| \psi \|_{\max} \leq \| \psi \|_{+} \leq \text{rank}_+(\psi) \cdot \| \psi \|_{\max} \).

If \( \| \psi \|_{+} = 1 \), then \( \| \psi \|_{+} = \| \psi \|_{\max} \).

### 2.2 Nonnegative Tensor Completion

For a partially observed order-\( p \) tensor \( \psi \), let \( (x(i), y(i)) \in \mathcal{R} \times \mathbb{R} \) for \( i = 1, \ldots, n \) denote the indices and value of \( n \) observed entries. We assume that an entry can be observed multiple times, so let \( U := \)}
\( \{x^{(1)}, \ldots, x^{(u)}\} \) where \( u \leq n \) denote the set of unique indices of observed entries. Since \( \| \cdot \|_+ \) is a convex surrogate for tensor rank, we have the following nonnegative tensor completion problem

\[
\hat{\psi} \in \arg \min_\psi \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \psi x^{(i)})^2 \quad \text{s.t.} \quad \|\psi\|_+ \leq \lambda
\]

(6)

where \( \lambda \in \mathbb{R}_+ \) is given and \( \hat{\psi} \) is the completed tensor. The feasible set \( \{\psi: \|\psi\|_+ \leq \lambda\} \) is equivalent to \( C_\lambda \) by the norm definition (2.1) and \( C_\lambda = \lambda C_1 \).

Although the problem (6) is a convex optimization problem, Bugg et al. [2022] showed that solving it is NP-hard. Nonetheless, Bugg et al. [2022] managed to design an efficient numerical computation algorithm of global minima of the problem (6) by using its substantial structure of the problem. We will explain their algorithm more after showing their findings on the statistical guarantees of the problem (6).

2.2.1 Statistical Guarantees

Bugg et al. [2022] observed that the problem (6) is equivalent to a convex aggregation [Nemirovski, 2000; Tsybakov, 2003; Lecue, 2013] problem for a finite set of functions, so we have the following tight generalization bound for the solution of the problem (6)

\[
\text{Proposition 2.1 ([Lecue, 2013])}. \quad \text{Suppose } |y| \leq b \text{ almost surely. Given any } \delta > 0, \text{ with probability at least } 1 - 4\delta \text{ we have that}
\]

\[
\mathbb{E}((y - \hat{\psi}_x)^2) \leq \min_{\varphi \in C_\lambda} \mathbb{E}((y - \varphi_x)^2) + c_0 \cdot \max[b^2, \lambda^2] \cdot \max[\zeta_n, \frac{\log(1/\delta)}{n}],
\]

(7)

where \( c_0 \) is an absolute constant and

\[
\zeta_n = \begin{cases} \frac{2^\rho}{n}, & \text{if } 2^\rho \leq \sqrt{n} \\ \frac{1}{n} \log \left( \frac{2^\rho}{\sqrt{n}} \right), & \text{if } 2^\rho > \sqrt{n} \end{cases}
\]

(8)

Under specific noise models such as an additive noise model, we have the following corollary to the above proposition combined with the fact that \( \| \cdot \|_+ \) is a convex surrogate for tensor rank.

\[
\text{Corollary 2.1 ([Bugg et al., 2022])}. \quad \text{Suppose } \varphi \text{ is a nonnegative tensor with } \text{rank}_+(\varphi) = k \text{ and } \|\varphi\|_{\text{max}} \leq \mu. \text{ If } (x^{(i)}, y^{(i)}) \text{ are independent and identically distributed with } |y^{(i)} - \varphi x^{(i)}| \leq e \text{ almost surely and } \mathbb{E}y^{(i)} = \varphi x^{(i)}. \text{ Then given any } \delta > 0, \text{ with probability at least } 1 - 4\delta \text{ we have}
\]

\[
\mathbb{E}((y - \hat{\psi}_x)^2) \leq e^2 + c_0 \cdot (\mu k + e)^2 \cdot \max[\zeta_n, \frac{\log(1/\delta)}{n}],
\]

(9)

where \( \zeta_n \) is as in (8) and \( c_0 \) is an absolute constant.

The two results above show that the problem (6) achieves the information-theoretic sample complexity rate when \( k = O(1) \).
3 ORIGINAL COMPUTATION ALGORITHM

Since $\mathcal{C}_1$ is a 0-1 polytope, we can use integer linear optimization to solve the linear separation problem over this polytope. Thus, we can apply the Frank-Wolfe algorithm or its variants to solve the problem (6) to a desired numerical tolerance. Bugg et al. [2022] choose the BCG variant for two reasons.

First, the BCG algorithm can terminate (within numerical tolerance) in a linear number of oracle steps for an optimization problem with a polytope feasible set and a strictly convex objective function over the feasible set. To make the objective function in the problem (6) strictly convex, we can reformulate the problem (6) by changing its feasible set from $\mathcal{C}_\lambda$ to $\text{Proj}_U(\mathcal{C}_\lambda)$. The implementation of this reformulation is to simply discard the unobserved entries of $\psi$. Second, the weak-separation oracle in the BCG algorithm accommodates early termination of the associated integer linear optimization problem, which is formulated as follows

$$
\min_{\varphi, \theta} \langle c, \varphi - \psi \rangle \\
\text{s.t.} \quad \lambda \cdot (1 - p) + \lambda \cdot \sum_{k=1}^{p} \theta_{x_k}^{(k)} \leq \varphi_x \quad x \in \mathcal{R} \\
0 \leq \varphi_x \leq \lambda \cdot \theta_{x_k}^{(k)} \quad k \in [p], x \in \mathcal{R} \\
\theta_{x_k}^{(k)} \in \{0, 1\} \quad k \in [p], x \in \mathcal{R}
$$

The feasible set in the problem above is equivalent to $\mathcal{S}_\lambda$, and the linear constraints above are acquired from standard techniques in integer optimization [Hansen, 1979; Padberg, 1989]. Bugg et al. [2022] also deploy a fast alternating minimization heuristic to solve the weak-separation oracle to avoid (if possible) solving the problem (10) via integer programming oracle.

Next, we will fully describe the Python 3 implementation of the BCG algorithm adapted by Bugg et al. [2022] to solve the problem (6) and its major computation components. This description is important for us to explain how we will accelerate this algorithm later. It also supplements the algorithm description omitted in [Bugg et al., 2022]. Their code is available from [https://github.com/WenhaoP/TensorComp](https://github.com/WenhaoP/TensorComp). For brevity, we do not explain the abstract frameworks of the adapted BCG (ABCG) or its major computation components here, but they can be found in [Braun et al., 2019] and [Bugg et al., 2022].

3.1 Adapted Blended Conditional Gradients

The ABCG is implemented as the function `nonten` in `original_nonten.py`. The inputs are indices of observed entries ($X$), values of observed entries ($Y$), dimension ($r$), $\lambda$ (lpar), and numerical tolerance ($tol$). The output is the completed tensor $\hat{\psi}$ ($sol$). There are three important preparation steps of ABCG. First, it reformulates the problem (6) by changing the feasible set from $\mathcal{C}_\lambda$ to $\text{Proj}_U(\mathcal{C}_\lambda)$. Second, it normalizes the feasible set to $\text{Proj}_U(\mathcal{C}_1)$ and scales $sol$ by $\lambda$ at the end of ABCG. Third, it flattens the iterate ($\psi_{q}$) and recovers the original dimension of $sol$ at the end of ABCG.

We first build the integer programming problem (10) in Gurobi. The iterate is initialized as a tensor of ones; the active vertex set $\{v_1, \ldots, v_k\}$ ($\text{Pts}$ and $\text{Vts}$) and the active vertex weight set $\{\gamma_1, \ldots, \gamma_k\}$ ($\text{lamb}$) are initialized accordingly. $\text{Vts}$ stores the $\theta^{(i)}$’s for constructing each active vertex in $\text{Pts}$. bestbd stores the best lower bound for the optimal objective value found so far. It is initialized as 0, a lower bound on the optimal objective value of problem (6). objVal stores the objective function value of the current iterate. m._gap stores half of the current primal gap estimate, the difference between objVal and the optimal objective function value [Braun et al., 2019].
The iteration procedure of BCG is implemented as a while loop. In each iteration, we first compute the gradient of the objective function with respect to the current iterate (lines 318 - 320). Then, we compute the dot products between the scaled gradient \((lpar*c)\) and each active vertex \((v_1, ..., v_k)\), store them in \(pro\), and use \(pro\) to find the away-step vertex \((\psi_a)\) and the Frank-Wolfe-step vertex \((\psi_f)\). Based on the test that compares \((lpar*c, \psi_a - \psi_f)\) and \(m_.gap\), we either use the simplex gradient descent step (SiGD) or the weak-separation oracle step (LPsep) to find the next iterate. Since we initialize \(m_.gap\) as \(+\infty\), we always solve the problem (10) to solve the weak-separation oracle in the first iteration. At the end of each iteration, we update \(objVal\) and \(bestbd\). The iteration procedure stops when either the primal gap \((2*m_.gap)\) or the largest possible primal gap \(objVal - bestbd\) is smaller than \(tol\).

### 3.2 Simplex Gradient Descent Step

SiGD is implemented in the function \(\text{nonten}\). It determines the next iterate with a lower objective function value via a single descent step that solves the following reformulation of the problem (6)

\[
\hat{\gamma}_1, ..., \hat{\gamma}_k \in \text{arg min}_{\gamma_1, ..., \gamma_k \in \mathbb{R}^+} \frac{1}{n} \sum_{i=1}^{n} \left( y(i) - \left( \sum_{j=1}^{k} \gamma_j v_j \right) x(i) \right)^2 \quad \text{s.t.} \quad \sum_{j=1}^{k} \gamma_j = 1 \quad (11)
\]

For clarity, we ignore the projection in the problem above. Since \(\lambda\) is in \(\Delta^k\) as all its elements are nonnegative and sum to one, SiGD has the term “Simplex” in its name \([\text{Braun et al.}, 2019]\).

We first compute the projection of \(pro\) onto the hyperplane of \(\Delta^k\) and store it in \(d\). If all elements of \(d\) are zero, then we set the next iterate to the first active vertex and end SiGD. Otherwise, we solve the optimization problem \(\hat{\eta} = \text{arg max} \{ \eta \in \mathbb{R}^+ : \gamma - \eta d \geq 0 \} \) where \(\gamma\) is \(\lambda\) and \(d\) is \(d\), and store the optimal solution \(\hat{\eta}\) in \(eta\). Next, we compute \(\psi_n\) which is \(x - \hat{\eta} \sum_i d_i v_i\) where \(x\) is the current iterate, \(d_i\)’s are the elements of \(d\), and \(v_i\) are active vertices. If the objective function value of \(\psi_n\) is smaller or equal to that of the current iterate, then we set the next iterate to \(\psi_n\) and drop any active vertex with zero weight in the updated active vertex set for constructing \(\psi_n\). Otherwise, we perform an exact line search over the line segment between the current iterate and \(\psi_n\), and set the next iterate to the optimal solution.

### 3.3 Weak-Separation Oracle Step

LPsep is implemented in the function \(\text{nonten}\). It has two differences from its implementation in the original BCG. First, whether it finds a new vertex that satisfies the weak-separation oracle or not, it always performs an exact line search over the line segment between the current iterate and that new vertex to find the next iterate. Second, it uses \(bestbd\) to update \(m_.gap\).

\(m_.cmin\) is the dot product between \(lpar*c\) and the current iterate. The flag variable \(oflg\) is True if an improving vertex has not been found that satisfies either the weak-separation oracle or the conditions described below. We first repeatedly use the alternating minimization heuristic (AltMin) to solve the weak-separation oracle. If AltMin fails to do so within 100 repetitions but finds a vertex that shows that \(m_.gap\) is an overestimate, we claim that it satisfies the weak-separation oracle and update \(m_.gap\). Otherwise, we
solve the integer programming problem via the Gurobi global solver to obtain a satisfying vertex and update \( m \). Note that an exact solution is not required; any solution attaining weak separation suffices.

### 3.4 Alternating Minimization

AltMin is implemented as the function `altmin` in the file `original_nonten.py`. Since a vertex is in \( S_\lambda \), the objective function in the integer programming problem (10) becomes \( \langle c, (\theta(1) \otimes \ldots \otimes \theta(p)) - \psi \rangle \) where \( \theta(k) \in \{0, 1\}^{r_k} \). There is a motivating observation for solving this problem. For example, if we fix \( \theta(2), \ldots, \theta(p) \), then the problem (10) is equivalent to

\[
\min_{\theta(1) \in \{0, 1\}^{r_1}} \langle \tilde{c}(1), \theta(1) \rangle
\]

where \( \tilde{c}(k) \) stores in the variable \( fpro \). The difference between the objective function values in (10) of \( \Theta_{T-1} \) and \( \Theta_T \) are guaranteed to be smaller than \( tol \). AltMin outputs \( \Theta_T \) at the end.

4 ACCELERATED COMPUTATION ALGORITHM

To show the efficacy and scalability of their approach, Bugg et al. [2022] conduct numerical experiments on a personal computer. They compare the performance of their approach and three other approaches – alternating least squares (ALS) [Kolda and Bader, 2009], simple low rank tensor completion (SiLRTC) [Liu et al., 2013], and trace norm regularized CP decomposition (TNCP) [Song et al., 2019] – in four sets of different nonnegative tensor completion problems. Normalized mean squared error (NMSE) \( \|\hat{\psi} - \psi\|_F^2 / \|\psi\|_F^2 \) is used to measure the accuracy.

The results of their numerical experiments show that Bugg et al. [2022]’s approach has higher accuracy but requires more computation time than the three other approaches in all four problem sets. We work directly on their code base to design acceleration techniques. Besides basic acceleration techniques such as caching repetitive computation operations, we design five different acceleration techniques based on profiling results. By combining these five techniques differently, we create ten variants of ABCG that can maintain the same theoretical guarantees but offer potentially faster computation.

#### 4.1 Technique 1A: Index

The first technique `Index` is for accelerating AltMin. \( fpro \) was computed using a for-loop with \( u \) iteration. The loop operation can be time-consuming when \( u \) is large (i.e., large sample size). We rewrote the computation of \( pro \) such that it was computed with a for-loop with \( r_k \) iterations for \( k = 1, \ldots, p \). Although each iteration of the new computation takes more time than the original, the numerical experiments show that total computation time drops significantly, especially for problems with large sample sizes.

#### 4.2 Technique 1B: Pattern

The second technique `Pattern` is for accelerating AltMin. It is based on the Index technique described above. In each iteration of the new loop operation in Index, we extract the information from \( U \) for
computation. Since $U$ remains unchanged throughout the algorithm, we can extract the information from $U$ at the beginning of the algorithm, avoiding repetition later on.

4.3 Technique 2: Sparse

The third technique Sparse is for accelerating the test for deciding whether to use SiGD or LPsep to determine the next iterate. At the beginning of each BCG iteration, $Pts$ and $lpar \cdot c$ are used to find $\psi_a$ and $\psi_f$. This operation involves a matrix multiplication between $Pts$ and $lpar \cdot c$, which is time-consuming when $Pts$ has a large size (i.e., large sample size or active vertex set size). We observed the sparsity of $Pts$ as vertices are binary, and used a SciPy [Virtanen et al., 2020] sparse matrix to represent it instead of a NumPy [Harris et al., 2020] array. It not only accelerates the matrix multiplication but also reduces the memory usage for storing $Pts$.

4.4 Technique 3: NAG

The fourth technique NAG is for accelerating SiGD. Since SiGD is a gradient descent method, we can use Nesterov’s accelerated gradient (NAG) to accelerate it. Specifically, we applied [Besançon et al., 2022]’s technique of transforming the problem (6) to its barycentric coordinates so that we minimize over $\Delta^k$ instead of the convex hull of the active vertex set. We restart the NAG when the active vertex set changes.

4.5 Technique 4: BPCG

The fifth technique BPCG is for accelerating SiGD. Tsuji et al. [2021] developed the Blended Pairwise Conditional Gradients (BPCG) by combining the Pairwise Conditional Gradients (PCG) with the blending criterion from BCG. Specifically, we applied the lazified version of BPCG, which only differs from BCG by replacing SiGD with PCG.

4.6 Computation Variants

We create ten computational variants of ABCG by mixing the five acceleration techniques described above. They are described in Table 1 of Appendix 1. There are two observations. First, Index and Pattern are always used together as Pattern relies on Index, so we combine and consider them as a single technique in all following discussions. Second, NAG and BPCG are exclusive to each other as BPCG removes SiGD completely.

5 NUMERICAL EXPERIMENTS

For benchmarking we adopted the same problem instances (four sets of nonnegative tensor completion problems) and setup as [Bugg et al., 2022]. We ran the original algorithm as well as each variant on these instances. The experiments were conducted on a laptop computer with 32GB of RAM and an Intel Core i7 2.2Ghz processor with 6-cores/12-threads. The algorithms were coded in Python 3. Gurobi v9.5.2 [Gurobi Optimization, LLC, 2022] was used to solve the integer programming problem (10). The code is available from https://github.com/WenhaoP/TensorComp. We note that the experiments in [Bugg et al., 2022] were conducted using a laptop computer with 8GB of RAM and an Intel Core i5 2.3Ghz processor with 2-cores/4-threads, algorithms coded in Python 3, and Gurobi v9.1 [Gurobi Optimization, LLC, 2022] to solve the integer programming problem (10).

We used NMSE to measure the accuracy and recorded its arithmetic mean (with its standard error) over one hundred repetitions of each problem instance. We recorded the arithmetic mean (with its standard
**Figure 1.** Results for order-3 nonnegative tensors with size $r \times r \times r$ and $n = 500$ samples.

**Figure 2.** Results for increasing order nonnegative tensors with size $10^p$ and $n = 10,000$ samples.
Figure 3. Results for nonnegative tensors with size $10 \times 6$ and increasing $n$ samples.

Figure 4. Results for nonnegative tensors with size $10 \times 7$ and increasing $n$ samples.
error), geometric mean, minimum, median, and maximum of the computation time over 100 repetitions of each problem instance. Generally speaking, the most effective variants were version 1 (Index + Pattern) and version 8 (BPCG + Index + Pattern), offering consistent speedups over the original version 0 (ABCG). All variants achieve almost the same NMSE as the original algorithm for each problem, which shows that they maintain in practice the same theoretical guarantee as the original algorithm; moreover, their runtimes were mostly competitive with the original.

5.1 Experiments with Order-3 Tensors

The first set of problems is order-3 tensors with increasing dimensions and \( n = 500 \) samples. The results are in Table 2 of Appendix 2 and the arithmetic mean and median computation time is plotted in Figure 1. Among all versions, version 1 (Index + Pattern) has the lowest mean and median computation time for each problem except for the \( 100 \times 100 \times 100 \) tensor problem, where the original version 0 (ABCG) is slightly faster. Thus, we claim version 1 (Index + Pattern) as the best to use for this set of problems.

5.2 Experiments with Increasing Tensor Order

The second set of problems is tensors with increasing tensor order, with each dimension \( r_i = 10 \) for \( i = 1, \ldots, p \), and \( n = 10,000 \) samples. The results are in Table 3 of Appendix 2 and the arithmetic mean and median computation time is plotted in Figure 2. Among all the versions, version 1 (Index + Pattern) and version 8 (BPCG + Index + Pattern) has the lowest mean and median computation time for each problem except for the \( 10 \times 8 \) tensor problem. Thus, we claim versions 1 (Index + Pattern) and 8 (BPCG + Index + Pattern) are the best to use for this set of problems.

For the \( 10 \times 8 \) tensor problem, version 5 (NAG + Index + Pattern) has the lowest median computation time. We observe that its median computation time is much lower than its mean computation time, and the same occurs with other versions involving NAG. The maximum computation time of these versions is also much higher than other versions. This is due to the fact that, in certain cases, AltMin failed to solve the weak-separation oracle, and the Gurobi solver was needed to solve the time-consuming integer programming problem (10).

5.3 Experiments with Increasing Sample Size

The third and fourth set of problems is tensors of size \( 10 \times 6 \) and \( 10 \times 7 \) with increasing sample sizes. The results are in Table 4 and Table 5 of Appendix 2 and the arithmetic mean and median computation time is plotted in Figure 3 and 4. Among all the versions, either version 1 (Index + Pattern) or version 8 (BPCG + Index + Pattern) has the lowest mean and median computation time for each problem. Thus, we claim versions 1 (Index + Pattern) and 8 (BPCG + Index + Pattern) are the best to use for this set of problems.

6 DISCUSSION AND CONCLUSION

This paper proposed and evaluated multiple speedup techniques of the original numerical computation algorithm for nonnegative tensor completion created by Bugg et al. [2022]. We benchmarked these algorithm variants on the same set of problem instances designed by Bugg et al. [2022]. Our benchmarking results were that versions 1 (Index + Pattern) and 8 (BPCG + Index + Pattern) generally had the fastest computation time for solving the nonnegative tensor completion problem (6), offering substantial speedups over the original algorithm. Version 1 had Index and Pattern techniques, and version 8 had BPCG, Index, and Pattern techniques. Because version 8 is version 1 with BPCG, version 8 may be preferred over version 1 for problem instances where the active vertex set could be large. Surprisingly, we found that Sparse failed
to yield improvements. A possible reason is that the implementation of Sparse needs extra operations (e.g., finding the indices of nonzero entries) for converting NumPy arrays to SciPy sparse arrays. If the active vertex set is not large enough so that extra operations take time as least as the time saved from matrix multiplications of the active vertex set, then we cannot see an improvement. These results suggest that Index and Pattern are the most important for acceleration, and that BCG and BPCG work equally well overall.

CONFLICT OF INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

AUTHOR CONTRIBUTIONS

WP, AA, and CC contributed to the conception and design of the study. WP did the programming, ran the numerical experiments, analyzed the results, and wrote the first draft of the manuscript. WP, AA, and CC contributed to the manuscript revision, read, and approved the submitted version.

FUNDING

This material is based upon work partially supported by the NSF under grant CMMI-1847666.

DATA AVAILABILITY STATEMENT

The datasets generated for this study can be found on Github: https://github.com/WenhaoP/TensorComp.

REFERENCES

Aswani, A. (2016). Low-rank approximation and completion of positive tensors. SIAM Journal on Matrix Analysis and Applications 37, 1337–1364
Barak, B. and Moitra, A. (2016). Noisy tensor completion via the sum-of-squares hierarchy. In Conference on Learning Theory (PMLR), 417–445
Besançon, M., Carderera, A., and Pokutta, S. (2022). Frankwolfe. jl: A high-performance and flexible toolbox for frank–wolfe algorithms and conditional gradients. INFORMS Journal on Computing
Braun, G., Pokutta, S., Tu, D., and Wright, S. (2019). Blended conditional gradients. In International Conference on Machine Learning (PMLR), 735–743
Bugg, C. X., Chen, C., and Aswani, A. (2022). Nonnegative tensor completion via integer optimization. In Advances in Neural Information Processing Systems, eds. A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho
Chandrasekaran, V., Recht, B., Parrilo, P. A., and Willsky, A. S. (2012). The convex geometry of linear inverse problems. Foundations of Computational mathematics 12, 805–849
Gandy, S., Recht, B., and Yamada, I. (2011). Tensor completion and low-n-rank tensor recovery via convex optimization. Inverse problems 27, 025010
Gurobi Optimization, LLC (2022). Gurobi optimizer reference manual
Hansen, P. (1979). Methods of nonlinear 0-1 programming. In Annals of Discrete Mathematics (Elsevier), vol. 5, 53–70
Pan et al.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with numpy. *Nature* 585, 357–362

Hillar, C. J. and Lim, L.-H. (2013). Most tensor problems are np-hard. *Journal of the ACM (JACM)* 60, 1–39

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review* 51, 455–500

Lecué, G. (2013). Empirical risk minimization is optimal for the convex aggregation problem. *Bernoulli* 19, 2153–2166

Liu, J., Musialski, P., Wonka, P., and Ye, J. (2013). Tensor completion for estimating missing values in visual data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 208–220. doi:10.1109/TPAMI.2012.39

Montanari, A. and Sun, N. (2018). Spectral algorithms for tensor completion. *Communications on Pure and Applied Mathematics* 71, 2381–2425

Mu, C., Huang, B., Wright, J., and Goldfarb, D. (2014). Square deal: Lower bounds and improved relaxations for tensor recovery. In *International conference on machine learning* (PMLR), 73–81

Nemirovski, A. (2000). Topics in non-parametric statistics. *Lectures on probability theory and statistics (Saint-Flour, 1998)* 1738, 85–277

Padberg, M. (1989). The boolean quadric polytope: some characteristics, facets and relatives. *Mathematical programming* 45, 139–172

Rao, N., Shah, P., and Wright, S. (2015). Forward–backward greedy algorithms for atomic norm regularization. *IEEE Transactions on Signal Processing* 63, 5798–5811

Rauhut, H. and Stojanac, Ž. (2021). Tensor theta norms and low rank recovery. *Numerical Algorithms* 88, 25–66

Song, Q., Ge, H., Caverlee, J., and Hu, X. (2019). Tensor completion algorithms in big data analytics. *ACM Trans. Knowl. Discov. Data* 13. doi:10.1145/3278607

Tan, H., Wu, Y., Feng, G., Wang, W., and Ran, B. (2013). A new traffic prediction method based on dynamic tensor completion. *Procedia-Social and Behavioral Sciences* 96, 2431–2442

Tsuji, K., Tanaka, K., and Pokutta, S. (2021). Sparser kernel herding with pairwise conditional gradients without swap steps. *arXiv preprint arXiv:2110.12650*

Tsybakov, A. B. (2003). Optimal rates of aggregation. In *Learning theory and kernel machines* (Springer), 303–313

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods* 17, 261–272

Yuan, M. and Zhang, C.-H. (2016). On tensor completion via nuclear norm minimization. *Foundations of Computational Mathematics* 16, 1031–1068

Yuan, M. and Zhang, C.-H. (2017). Incoherent tensor norms and their applications in higher order tensor completion. *IEEE Transactions on Information Theory* 63, 6753–6766

Zhang, X., Wang, D., Zhou, Z., and Ma, Y. (2019). Robust low-rank tensor recovery with rectification and alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 238–255

Frontiers

13
### APPENDIX 1: TABLE OF VARIANTS

Table 1  ABCG and its ten variants

| Version | Sparse | NAG | BPCG | Index | Pattern |
|---------|--------|-----|------|-------|---------|
| 0 (ABCG) | FALSE | FALSE | FALSE | FALSE | FALSE |
| 1       | FALSE | FALSE | FALSE | TRUE  | TRUE   |
| 2       | TRUE  | FALSE | FALSE | FALSE | FALSE |
| 3       | FALSE | TRUE  | FALSE | FALSE | FALSE |
| 4       | TRUE  | FALSE | FALSE | TRUE  | TRUE   |
| 5       | FALSE | TRUE  | FALSE | TRUE  | TRUE   |
| 6       | TRUE  | TRUE  | FALSE | TRUE  | TRUE   |
| 7       | FALSE | FALSE | TRUE  | FALSE | FALSE |
| 8       | FALSE | FALSE | TRUE  | TRUE  | TRUE   |
| 9       | TRUE  | FALSE | TRUE  | FALSE | FALSE |
| 10      | TRUE  | FALSE | TRUE  | TRUE  | TRUE   |

Note: True means we implement this technique in this variant, and False means we do not implement this technique in this variant. For example, version 5 implements NAG, Index, and Pattern.
### APPENDIX 2: TABLES OF NUMERICAL EXPERIMENT RESULTS

Table 2  Results for order-3 nonnegative tensors with \( n = 500 \) samples

| Tensor Size | Version | NMSE      | Arithmetic Mean ± SE | Time (s) | Arithmetic Geometric Mean ± SE |
|-------------|---------|-----------|---------------------|----------|-------------------------------|
|             |         |           |                     |          |                               |
| 10×3        | 0       | 0.017 ± 0.001 | 1.874 ± 0.061      | 1.777    | 0.756 1.849 3.935           |
|             | 1       | 0.017 ± 0.001 | 1.054 ± 0.044      | 0.966    | 0.352 1.006 2.688           |
|             | 2       | 0.018 ± 0.001 | 2.116 ± 0.071      | 1.996    | 0.757 2.102 4.114           |
|             | 3       | 0.017 ± 0.001 | 2.225 ± 0.067      | 2.120    | 0.885 2.224 4.234           |
|             | 4       | 0.018 ± 0.001 | 1.346 ± 0.060      | 1.218    | 0.381 1.277 2.998           |
|             | 5       | 0.017 ± 0.001 | 1.450 ± 0.056      | 1.339    | 0.398 1.386 3.168           |
|             | 6       | 0.017 ± 0.001 | 1.868 ± 0.073      | 1.720    | 0.520 1.810 4.035           |
|             | 7       | 0.017 ± 0.001 | 2.014 ± 0.063      | 1.914    | 0.819 2.001 3.383           |
|             | 8       | 0.017 ± 0.001 | 2.834 ± 0.113      | 2.600    | 0.829 2.681 5.694           |
|             | 9       | 0.017 ± 0.001 | 2.093 ± 0.103      | 1.828    | 0.48 1.821 4.795           |
|             | 10      | 0.017 ± 0.001 | 2.093 ± 0.103      | 1.828    | 0.48 1.821 4.795           |
| 20×3        | 0       | 0.115 ± 0.004 | 3.931 ± 0.073      | 3.860    | 2.086 3.952 6.390           |
|             | 1       | 0.115 ± 0.004 | 2.094 ± 0.043      | 2.046    | 1.020 2.139 3.314           |
|             | 2       | 0.118 ± 0.004 | 4.170 ± 0.078      | 4.091    | 1.884 4.272 5.912           |
|             | 3       | 0.118 ± 0.004 | 4.639 ± 0.077      | 4.574    | 2.955 4.728 7.014           |
|             | 4       | 0.118 ± 0.004 | 2.518 ± 0.051      | 2.459    | 1.029 2.588 3.625           |
|             | 5       | 0.118 ± 0.004 | 2.898 ± 0.051      | 2.849    | 1.522 2.980 3.958           |
|             | 6       | 0.118 ± 0.004 | 3.879 ± 0.070      | 3.811    | 1.999 3.915 5.515           |
|             | 7       | 0.114 ± 0.004 | 4.561 ± 0.076      | 4.497    | 2.405 4.636 6.781           |
|             | 8       | 0.114 ± 0.004 | 2.768 ± 0.049      | 2.722    | 1.173 2.742 4.529           |
|             | 9       | 0.117 ± 0.004 | 6.315 ± 0.116      | 6.200    | 3.526 6.309 9.101           |
|             | 10      | 0.117 ± 0.004 | 4.712 ± 0.096      | 4.602    | 2.274 4.747 6.609           |
| 30×3        | 0       | 0.240 ± 0.005 | 2.935 ± 0.068      | 2.852    | 1.547 2.952 4.793           |
|             | 1       | 0.240 ± 0.005 | 1.600 ± 0.039      | 1.551    | 0.865 1.584 2.488           |
|             | 2       | 0.241 ± 0.005 | 2.967 ± 0.077      | 2.871    | 1.659 2.931 5.475           |
|             | 3       | 0.241 ± 0.005 | 3.435 ± 0.072      | 3.360    | 1.818 3.424 5.760           |
|             | 4       | 0.241 ± 0.005 | 1.774 ± 0.051      | 1.705    | 0.973 1.711 3.659           |
|             | 5       | 0.241 ± 0.005 | 2.222 ± 0.053      | 2.161    | 1.228 2.160 3.936           |
|             | 6       | 0.241 ± 0.005 | 2.846 ± 0.073      | 2.757    | 1.600 2.757 5.172           |
|             | 7       | 0.240 ± 0.005 | 3.561 ± 0.082      | 3.467    | 1.808 3.419 5.874           |
|             | 8       | 0.240 ± 0.005 | 2.192 ± 0.058      | 2.112    | 1.060 2.142 3.501           |
|             | 9       | 0.239 ± 0.005 | 4.649 ± 0.139      | 4.433    | 2.079 4.598 8.300           |
|             | 10      | 0.239 ± 0.005 | 3.328 ± 0.117      | 3.112    | 1.475 3.313 5.927           |
| 40×3        | 0       | 0.323 ± 0.004 | 2.171 ± 0.053      | 2.111    | 1.377 2.099 4.007           |
|             | 1       | 0.323 ± 0.004 | 1.314 ± 0.032      | 1.279    | 0.865 1.255 2.528           |
|             | 2       | 0.322 ± 0.004 | 2.250 ± 0.053      | 2.193    | 1.439 2.214 4.380           |
|             | 3       | 0.321 ± 0.004 | 2.661 ± 0.053      | 2.610    | 1.723 2.676 4.692           |
|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 0.322 ± 0.004 | 1.423 ± 0.033 | 1.389 | 0.926 | 1.365 | 2.847 |
| 5 | 0.321 ± 0.004 | 1.812 ± 0.036 | 1.780 | 1.225 | 1.755 | 3.351 |
| 6 | 0.321 ± 0.004 | 2.211 ± 0.045 | 2.172 | 1.546 | 2.114 | 4.350 |
| 7 | 0.321 ± 0.004 | 2.641 ± 0.059 | 2.577 | 1.484 | 2.627 | 4.666 |
| 8 | 0.321 ± 0.004 | 1.684 ± 0.042 | 1.636 | 0.984 | 1.626 | 3.312 |
| 9 | 0.321 ± 0.004 | 3.125 ± 0.083 | 3.022 | 1.911 | 3.034 | 5.673 |
|10 | 0.321 ± 0.004 | 2.248 ± 0.071 | 2.151 | 1.333 | 2.058 | 4.529 |
|50\( \times^3 \) |   |   |   |   |   |   |   |   |   |
| 0 | 0.385 ± 0.005 | 1.876 ± 0.042 | 1.835 | 1.327 | 1.725 | 3.046 |
| 1 | 0.385 ± 0.005 | 1.305 ± 0.027 | 1.281 | 0.900 | 1.216 | 2.289 |
| 2 | 0.383 ± 0.004 | 2.025 ± 0.045 | 1.982 | 1.321 | 1.885 | 3.622 |
| 3 | 0.382 ± 0.005 | 2.272 ± 0.047 | 2.229 | 1.676 | 2.162 | 3.866 |
| 4 | 0.383 ± 0.004 | 1.444 ± 0.029 | 1.418 | 1.077 | 1.360 | 2.535 |
| 5 | 0.382 ± 0.005 | 1.691 ± 0.032 | 1.663 | 1.248 | 1.618 | 2.760 |
| 6 | 0.382 ± 0.005 | 2.053 ± 0.035 | 2.026 | 1.566 | 1.974 | 3.193 |
| 7 | 0.384 ± 0.004 | 2.181 ± 0.054 | 2.121 | 1.447 | 2.010 | 3.839 |
| 8 | 0.384 ± 0.004 | 1.541 ± 0.038 | 1.499 | 1.055 | 1.425 | 2.697 |
| 9 | 0.384 ± 0.005 | 2.659 ± 0.060 | 2.601 | 1.889 | 2.459 | 4.401 |
|10 | 0.384 ± 0.005 | 2.033 ± 0.046 | 1.989 | 1.504 | 1.897 | 3.544 |
|60\( \times^3 \) |   |   |   |   |   |   |   |   |   |
| 0 | 0.444 ± 0.004 | 1.821 ± 0.031 | 1.797 | 1.409 | 1.733 | 2.765 |
| 1 | 0.444 ± 0.004 | 1.403 ± 0.023 | 1.386 | 1.079 | 1.342 | 2.154 |
| 2 | 0.444 ± 0.004 | 1.990 ± 0.033 | 1.965 | 1.550 | 1.887 | 2.965 |
| 3 | 0.444 ± 0.004 | 2.205 ± 0.044 | 2.166 | 1.565 | 2.013 | 3.808 |
| 4 | 0.444 ± 0.004 | 1.577 ± 0.024 | 1.560 | 1.213 | 1.504 | 2.348 |
| 5 | 0.444 ± 0.004 | 1.777 ± 0.031 | 1.751 | 1.309 | 1.641 | 2.991 |
| 6 | 0.444 ± 0.004 | 2.197 ± 0.035 | 2.171 | 1.626 | 2.078 | 3.722 |
| 7 | 0.445 ± 0.005 | 2.013 ± 0.044 | 1.970 | 1.442 | 1.879 | 3.631 |
| 8 | 0.445 ± 0.005 | 1.576 ± 0.032 | 1.546 | 1.138 | 1.478 | 2.842 |
| 9 | 0.446 ± 0.004 | 2.469 ± 0.049 | 2.427 | 1.791 | 2.322 | 4.521 |
|10 | 0.446 ± 0.004 | 2.047 ± 0.041 | 2.012 | 1.483 | 1.953 | 3.967 |
|70\( \times^3 \) |   |   |   |   |   |   |   |   |   |
| 0 | 0.497 ± 0.004 | 1.865 ± 0.032 | 1.841 | 1.386 | 1.791 | 3.426 |
| 1 | 0.497 ± 0.004 | 1.573 ± 0.027 | 1.552 | 1.174 | 1.520 | 2.861 |
| 2 | 0.498 ± 0.004 | 2.054 ± 0.032 | 2.031 | 1.577 | 1.995 | 3.699 |
| 3 | 0.495 ± 0.005 | 2.187 ± 0.042 | 2.152 | 1.548 | 2.084 | 4.448 |
| 4 | 0.498 ± 0.004 | 1.768 ± 0.026 | 1.751 | 1.342 | 1.728 | 3.095 |
| 5 | 0.495 ± 0.005 | 1.947 ± 0.036 | 1.920 | 1.332 | 1.895 | 4.123 |
| 6 | 0.495 ± 0.005 | 2.384 ± 0.039 | 2.357 | 1.689 | 2.332 | 4.698 |
| 7 | 0.498 ± 0.004 | 2.045 ± 0.043 | 2.006 | 1.480 | 1.933 | 3.964 |
| 8 | 0.498 ± 0.004 | 1.746 ± 0.037 | 1.713 | 1.246 | 1.640 | 3.412 |
| 9 | 0.500 ± 0.004 | 2.507 ± 0.055 | 2.462 | 1.891 | 2.387 | 5.654 |
|10 | 0.500 ± 0.004 | 2.229 ± 0.049 | 2.190 | 1.694 | 2.118 | 5.11 |
|80\( \times^3 \) |   |   |   |   |   |   |   |   |   |
| 0 | 0.546 ± 0.004 | 1.943 ± 0.036 | 1.915 | 1.439 | 1.856 | 3.640 |
| 1 | 0.546 ± 0.004 | 1.768 ± 0.031 | 1.743 | 1.32 | 1.700 | 3.273 |
| 2 | 0.545 ± 0.004 | 2.163 ± 0.035 | 2.138 | 1.665 | 2.092 | 3.992 |
| 3 | 0.542 ± 0.004 | 2.282 ± 0.041 | 2.249 | 1.654 | 2.158 | 3.858 |
|  | 0.545 ± 0.004 | 1.998 ± 0.032 | 1.976 | 1.552 | 1.935 | 3.759 |
|---|---|---|---|---|---|---|
| 5 | 0.542 ± 0.004 | 2.140 ± 0.037 | 2.112 | 1.566 | 2.043 | 3.686 |
| 6 | 0.542 ± 0.004 | 2.645 ± 0.041 | 2.617 | 1.959 | 2.568 | 4.670 |
| 7 | 0.547 ± 0.004 | 2.125 ± 0.044 | 2.087 | 1.589 | 2.002 | 4.312 |
| 8 | 0.547 ± 0.004 | 1.956 ± 0.043 | 1.920 | 1.859 | 1.859 | 4.544 |
| 9 | 0.546 ± 0.004 | 2.600 ± 0.054 | 2.557 | 1.902 | 2.484 | 6.003 |
| 10| 0.546 ± 0.004 | 2.443 ± 0.050 | 2.405 | 1.858 | 2.371 | 5.699 |

| 90^3 | 0 | 0.580 ± 0.004 | 2.049 ± 0.039 | 2.016 | 1.416 | 1.928 | 3.341 |
|---|---|---|---|---|---|---|---|
| 1 | 0.580 ± 0.004 | 1.991 ± 0.038 | 1.959 | 1.441 | 1.876 | 3.380 |
| 2 | 0.580 ± 0.004 | 2.296 ± 0.036 | 2.271 | 1.663 | 2.216 | 3.516 |
| 3 | 0.575 ± 0.004 | 2.416 ± 0.037 | 2.387 | 1.669 | 2.366 | 3.210 |
| 4 | 0.580 ± 0.004 | 2.269 ± 0.035 | 2.244 | 1.697 | 2.189 | 3.367 |
| 5 | 0.575 ± 0.004 | 2.404 ± 0.038 | 2.375 | 1.671 | 2.367 | 3.488 |
| 6 | 0.575 ± 0.004 | 2.987 ± 0.042 | 2.958 | 2.101 | 2.987 | 4.231 |
| 7 | 0.579 ± 0.004 | 2.257 ± 0.045 | 2.216 | 1.541 | 2.108 | 3.658 |
| 8 | 0.579 ± 0.004 | 2.215 ± 0.045 | 2.174 | 1.474 | 2.062 | 3.642 |
| 9 | 0.579 ± 0.004 | 2.774 ± 0.042 | 2.744 | 1.854 | 2.693 | 4.205 |
| 10| 0.579 ± 0.004 | 2.729 ± 0.043 | 2.698 | 1.886 | 2.679 | 4.089 |

| 100^3 | 0 | 0.611 ± 0.004 | 2.150 ± 0.038 | 2.119 | 1.520 | 2.074 | 3.353 |
|---|---|---|---|---|---|---|---|
| 1 | 0.611 ± 0.004 | 2.295 ± 0.041 | 2.262 | 1.660 | 2.195 | 3.702 |
| 2 | 0.611 ± 0.004 | 2.451 ± 0.037 | 2.425 | 1.801 | 2.373 | 3.946 |
| 3 | 0.603 ± 0.004 | 2.624 ± 0.054 | 2.575 | 1.671 | 2.484 | 4.746 |
| 4 | 0.611 ± 0.004 | 2.575 ± 0.039 | 2.547 | 1.839 | 2.530 | 4.168 |
| 5 | 0.603 ± 0.004 | 2.708 ± 0.054 | 2.659 | 1.815 | 2.589 | 4.811 |
| 6 | 0.603 ± 0.004 | 3.285 ± 0.059 | 3.237 | 2.173 | 3.170 | 5.938 |
| 7 | 0.608 ± 0.004 | 2.542 ± 0.065 | 2.468 | 1.645 | 2.380 | 4.636 |
| 8 | 0.608 ± 0.004 | 2.963 ± 0.097 | 2.845 | 1.831 | 2.698 | 8.809 |
| 9 | 0.609 ± 0.004 | 3.149 ± 0.061 | 3.097 | 2.128 | 3.005 | 6.003 |
| 10| 0.609 ± 0.004 | 3.178 ± 0.062 | 3.126 | 2.027 | 3.029 | 6.136 |

Note: For each problem, the lowest mean and median computation time are highlighted.
### Table 3  Results for increasing order nonnegative tensors and $n = 10,000$ samples

| Tensor Size | Version | **NMSE** Arithmetic Mean ± SE | **Arithmetic Mean ± SE** | **Time (s)** Arithmetic Geometric Mean Min Median Max |
|-------------|---------|-------------------------------|--------------------------|-----------------------------------------------|
| $10^{4}$    | 0       | 0.010 ± 0.000 14.620 ± 0.160 | 14.533 11.13 14.723 20.162 | |
|             | 1       | 0.010 ± 0.000 4.515 ± 0.081 | 4.443 2.658 4.497 7.198  |
|             | 2       | 0.010 ± 0.000 14.736 ± 0.194 | 14.622 11.543 14.569 25.271 | |
|             | 3       | 0.010 ± 0.000 15.814 ± 0.194 | 15.698 11.103 15.732 21.805 | |
|             | 4       | 0.010 ± 0.000 4.335 ± 0.084 | 4.260 2.725 4.237 8.669  |
|             | 5       | 0.010 ± 0.000 5.608 ± 0.098 | 5.521 2.963 5.590 8.890  |
|             | 6       | 0.010 ± 0.000 5.784 ± 0.107 | 5.686 2.974 5.752 9.049  |
|             | 7       | 0.010 ± 0.000 14.761 ± 0.194 | 14.643 11.197 14.555 22.625 | |
|             | 8       | 0.010 ± 0.000 4.200 ± 0.074 | 4.138 2.524 4.194 8.091  |
|             | 9       | 0.010 ± 0.000 14.802 ± 0.199 | 14.678 10.361 14.700 23.429 | |
|             | 10      | 0.010 ± 0.000 4.419 ± 0.088 | 4.332 2.424 4.442 8.617  |
| $10^{5}$    | 0       | 0.025 ± 0.001 25.214 ± 0.275 | 25.063 18.646 25.012 33.507 | |
|             | 1       | 0.025 ± 0.001 5.512 ± 0.105 | 5.409 3.140 5.537 8.445  |
|             | 2       | 0.025 ± 0.001 25.195 ± 0.276 | 25.041 17.925 25.227 33.771 | |
|             | 3       | 0.025 ± 0.001 25.598 ± 0.275 | 25.449 18.907 25.801 33.583 | |
|             | 4       | 0.025 ± 0.001 5.504 ± 0.113 | 5.383 3.106 5.585 8.536  |
|             | 5       | 0.025 ± 0.001 6.391 ± 0.106 | 6.297 3.458 6.433 9.155  |
|             | 6       | 0.025 ± 0.001 6.582 ± 0.109 | 6.487 3.697 6.590 9.444  |
|             | 7       | 0.025 ± 0.001 24.640 ± 0.305 | 24.462 18.711 24.558 39.111 | |
|             | 8       | 0.025 ± 0.001 4.899 ± 0.068 | 4.850 3.126 4.914 6.599  |
|             | 9       | 0.025 ± 0.001 24.502 ± 0.290 | 24.340 17.616 24.007 38.234 | |
|             | 10      | 0.025 ± 0.001 5.174 ± 0.084 | 5.106 3.203 5.127 8.583  |
| $10^{6}$    | 0       | 0.057 ± 0.002 33.437 ± 0.475 | 33.076 17.539 33.360 50.010 | |
|             | 1       | 0.057 ± 0.002 6.562 ± 0.111 | 6.467 3.647 6.566 9.963  |
|             | 2       | 0.058 ± 0.002 33.606 ± 0.393 | 33.357 17.525 33.712 41.967 | |
|             | 3       | 0.058 ± 0.003 32.549 ± 0.436 | 32.241 18.479 32.502 43.975 | |
|             | 4       | 0.058 ± 0.002 6.883 ± 0.098 | 6.809 3.914 6.906 8.725  |
|             | 5       | 0.058 ± 0.003 6.869 ± 0.112 | 6.770 4.069 7.110 9.002  |
|             | 6       | 0.058 ± 0.003 7.310 ± 0.121 | 7.202 4.343 7.522 10.257 | |
|             | 7       | 0.057 ± 0.002 31.687 ± 0.419 | 31.394 16.671 31.747 42.899 | |
|             | 8       | 0.057 ± 0.002 5.750 ± 0.082 | 5.688 3.091 5.777 7.923  |
|             | 9       | 0.057 ± 0.003 32.288 ± 0.399 | 32.024 18.471 32.798 41.716 | |
|             | 10      | 0.057 ± 0.003 6.313 ± 0.097 | 6.233 3.326 6.418 8.507  |
| $10^{7}$    | 0       | 0.145 ± 0.007 42.770 ± 1.192 | 41.208 15.333 41.701 96.280 | |
|             | 1       | 0.145 ± 0.007 8.799 ± 0.233 | 8.504 3.763 8.448 17.070  |
|             | 2       | 0.147 ± 0.008 47.611 ± 1.045 | 46.352 17.773 46.859 78.931 | |
|             | 3       | 0.145 ± 0.008 42.546 ± 1.046 | 41.273 18.386 42.818 82.233 | |
|             | 4       | 0.147 ± 0.008 13.314 ± 0.253 | 13.054 6.425 13.584 19.319 | |
|             | 5       | 0.145 ± 0.008 9.650 ± 0.206 | 9.430 5.242 9.701 16.758 | |
| Problem | Mean ± Standard Error | Median ± Standard Error | Mean 10^8 ± Standard Error | Median 10^8 ± Standard Error |
|---------|-----------------------|-------------------------|----------------------------|-----------------------------|
| 6       | 0.145 ± 0.008         | 12.430 ± 0.254          | 12.168                     | 6.576                       |
| 7       | 0.146 ± 0.008         | 41.994 ± 0.942          | 40.854                     | 21.795                      |
| 8       | 0.146 ± 0.008         | **8.729 ± 0.190**       | 8.522                      | 5.101                       |
| 9       | 0.146 ± 0.008         | 46.652 ± 1.067          | 45.448                     | 21.751                      |
| 10      | 0.146 ± 0.008         | 11.801 ± 0.254          | 11.534                     | 6.208                       |

Note: For each problem, the lowest mean and median computation time are highlighted.

Frontiers
Table 4  Results for nonnegative tensors with size $10 \times 6$ and increasing $n$ samples

| Sample Percent | Version | NMSE Arithmetic Mean ± SE | Arithmetic Geometric Mean ± SE | Time (s) Arithmetic Mean ± SE | Time (s) Geometric Mean ± SE |
|----------------|---------|---------------------------|--------------------------------|--------------------------------|-------------------------------|
| 0.01           | 0       | 0.982 ± 0.004             | 0.276 ± 0.010                  | 0.258 ± 0.009                  | 0.227 ± 0.012                 |
|                | 1       | 0.982 ± 0.004             | 0.243 ± 0.009                  | 0.227 ± 0.012                  | 0.215 ± 0.012                 |
|                | 2       | 0.981 ± 0.004             | 0.329 ± 0.016                  | 0.298 ± 0.015                  | 0.285 ± 0.015                 |
|                | 3       | 0.982 ± 0.004             | 0.304 ± 0.012                  | 0.282 ± 0.014                  | 0.265 ± 0.014                 |
|                | 4       | 0.981 ± 0.004             | 0.299 ± 0.015                  | 0.270 ± 0.013                  | 0.256 ± 0.013                 |
|                | 5       | 0.982 ± 0.004             | 0.265 ± 0.011                  | 0.246 ± 0.129                  | 0.242 ± 0.011                 |
|                | 6       | 0.982 ± 0.004             | 0.292 ± 0.013                  | 0.266 ± 0.131                  | 0.281 ± 0.013                 |
|                | 7       | 0.980 ± 0.004             | 0.284 ± 0.012                  | 0.264 ± 0.151                  | 0.252 ± 0.015                 |
|                | 8       | 0.980 ± 0.004             | 0.250 ± 0.010                  | 0.232 ± 0.131                  | 0.211 ± 0.010                 |
|                | 9       | 0.981 ± 0.004             | 0.328 ± 0.016                  | 0.299 ± 0.148                  | 0.281 ± 0.013                 |
|                | 10      | 0.981 ± 0.004             | 0.293 ± 0.015                  | 0.264 ± 0.129                  | 0.253 ± 0.015                 |
| 0.1            | 0       | 0.564 ± 0.022             | 9.507 ± 0.439                  | 8.565 ± 2.548                  | 8.182 ± 21.995               |
|                | 1       | 0.564 ± 0.022             | 2.977 ± 0.116                  | 2.758 ± 0.843                  | 2.616 ± 6.217                |
|                | 2       | 0.568 ± 0.022             | 9.161 ± 0.370                  | 8.469 ± 2.804                  | 8.037 ± 20.648               |
|                | 3       | 0.571 ± 0.022             | 9.504 ± 0.415                  | 8.647 ± 2.548                  | 8.306 ± 24.41                |
|                | 4       | 0.568 ± 0.022             | 3.176 ± 0.107                  | 3.002 ± 1.087                  | 2.948 ± 6.223                |
|                | 5       | 0.571 ± 0.022             | 3.040 ± 0.116                  | 2.822 ± 0.874                  | 2.836 ± 7.308                |
|                | 6       | 0.571 ± 0.022             | 3.482 ± 0.130                  | 3.237 ± 0.952                  | 3.273 ± 8.053                |
|                | 7       | 0.547 ± 0.023             | 9.270 ± 0.467                  | 8.291 ± 2.227                  | 8.018 ± 31.432               |
|                | 8       | 0.547 ± 0.023             | 2.931 ± 0.140                  | 2.666 ± 0.844                  | 2.669 ± 10.910               |
|                | 9       | 0.562 ± 0.022             | 9.111 ± 0.388                  | 8.329 ± 2.841                  | 8.463 ± 21.631               |
|                | 10      | 0.562 ± 0.022             | 3.334 ± 0.129                  | 3.089 ± 0.965                  | 3.194 ± 7.059                |
| 1              | 0       | 0.057 ± 0.002             | 33.437 ± 0.475                 | 33.076 ± 17.539                | 33.360 ± 50.010              |
|                | 1       | 0.057 ± 0.002             | 6.562 ± 0.111                  | 6.467 ± 3.647                  | 6.566 ± 9.963                |
|                | 2       | 0.058 ± 0.002             | 33.606 ± 0.393                 | 33.357 ± 17.525                | 33.712 ± 41.967              |
|                | 3       | 0.058 ± 0.003             | 32.549 ± 0.436                 | 32.241 ± 18.479                | 32.502 ± 43.975              |
|                | 4       | 0.058 ± 0.002             | 6.883 ± 0.098                  | 6.809 ± 3.914                  | 6.906 ± 8.725                |
|                | 5       | 0.058 ± 0.003             | 6.869 ± 0.112                  | 6.770 ± 4.069                  | 7.110 ± 9.002                |
|                | 6       | 0.058 ± 0.003             | 7.310 ± 0.121                  | 7.202 ± 4.343                  | 7.522 ± 10.257               |
|                | 7       | 0.057 ± 0.002             | 31.687 ± 0.419                 | 31.394 ± 16.671                | 31.747 ± 42.899              |
|                | 8       | 0.057 ± 0.002             | 5.750 ± 0.082                  | 5.688 ± 3.091                  | 5.777 ± 7.923                |
|                | 9       | 0.057 ± 0.003             | 32.288 ± 0.399                 | 32.024 ± 18.471                | 32.798 ± 41.716              |
|                | 10      | 0.057 ± 0.003             | 6.313 ± 0.097                  | 6.233 ± 3.326                  | 6.418 ± 8.507                |

10 ($n = 10^5$)
|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 5 | 0.050 ± 0.002 | 56.653 ± 0.964 | 55.822 | 37.055 | 56.013 | 80.008 |
| 6 | 0.050 ± 0.002 | 56.285 ± 0.975 | 55.431 | 34.631 | 55.497 | 79.487 |
| 7 | 0.050 ± 0.002 | 241.195 ± 3.388 | 238.465 | 138.051 | 242.977 | 333.482 |
| 8 | 0.050 ± 0.002 | **48.434 ± 0.696** | **47.934** | 30.473 | **48.433** | 69.063 |
| 9 | 0.050 ± 0.002 | 244.385 ± 3.051 | 242.218 | 149.075 | 245.842 | 335.246 |
| 10 | 0.050 ± 0.002 | 50.598 ± 0.711 | 50.099 | 34.797 | 50.104 | 73.951 |

Note: For each problem, the lowest mean and median computation time are highlighted.
Table 5 Results for nonnegative tensors with size $10 \times 7$ and increasing $n$ samples

| Sample Percent | Version | NMSE Arithmetic Mean ± SE | NMSE Geometric Mean ± SE | Time (s) Arithmetic Mean | Time (s) Geometric Mean | Min | Median | Max |
|----------------|---------|---------------------------|--------------------------|--------------------------|------------------------|-----|--------|-----|
| 0.01           | 0       | 0.884 ± 0.016             | 9.906 ± 0.733            | 7.952                    | 1.338                  | 7.456 | 44.304 |
|                | 1       | 0.884 ± 0.016             | 5.154 ± 0.371            | 4.266                    | 0.772                  | 3.993 | 25.657 |
|                | 2       | 0.883 ± 0.016             | 10.435 ± 0.717           | 8.677                    | 1.413                  | 8.409 | 41.956 |
|                | 3       | 0.885 ± 0.015             | 8.978 ± 0.637            | 7.337                    | 1.441                  | 7.069 | 35.707 |
|                | 4       | 0.883 ± 0.016             | 6.120 ± 0.385            | 5.259                    | 0.897                  | 5.237 | 25.234 |
|                | 5       | 0.885 ± 0.015             | 4.542 ± 0.307            | 3.846                    | 0.833                  | 3.825 | 20.304 |
|                | 6       | 0.888 ± 0.015             | 6.185 ± 0.496            | 5.055                    | 0.913                  | 4.931 | 34.564 |
|                | 7       | 0.870 ± 0.018             | 8.363 ± 0.612            | 6.796                    | 1.506                  | 6.664 | 40.259 |
|                | 8       | 0.870 ± 0.018             | 4.242 ± 0.286            | 3.585                    | 0.818                  | 3.483 | 18.221 |
|                | 9       | 0.874 ± 0.017             | 9.320 ± 0.663            | 7.599                    | 1.565                  | 7.122 | 32.849 |
|                | 10      | 0.874 ± 0.017             | 5.444 ± 0.373            | 4.546                    | 0.939                  | 4.325 | 21.065 |
| 0.1            | 0       | 0.145 ± 0.007             | 42.770 ± 1.192           | 41.208                   | 15.333                 | 41.701 | 96.280 |
|                | 1       | 0.145 ± 0.007             | 8.799 ± 0.233            | **8.504**                | 3.763                  | **8.448** | 17.070 |
|                | 2       | 0.147 ± 0.008             | 47.611 ± 1.045           | 46.352                   | 17.773                 | 46.859 | 78.931 |
|                | 3       | 0.145 ± 0.008             | 42.546 ± 1.046           | 41.273                   | 18.386                 | 42.818 | 82.233 |
|                | 4       | 0.147 ± 0.008             | 13.314 ± 0.253           | 13.054                   | 6.425                 | 13.584 | 19.319 |
|                | 5       | 0.145 ± 0.008             | 9.650 ± 0.206            | 9.430                    | 5.242                  | 9.701  | 16.758 |
|                | 6       | 0.145 ± 0.008             | 12.430 ± 0.254           | 12.168                   | 6.576                  | 12.144 | 20.844 |
|                | 7       | 0.146 ± 0.008             | 41.994 ± 0.942           | 40.854                   | 21.795                 | 41.982 | 60.821 |
|                | 8       | 0.146 ± 0.008             | **8.729** ± 0.190        | 8.522                    | 5.101                  | 8.807  | 14.279 |
|                | 9       | 0.146 ± 0.008             | 46.652 ± 1.067           | 45.448                   | 21.751                 | 46.404 | 83.582 |
|                | 10      | 0.146 ± 0.008             | 11.801 ± 0.254           | 11.534                   | 6.208                  | 11.542 | 20.518 |
| 1              | 0       | 0.112 ± 0.005             | 245.863 ± 6.133          | 238.338                   | 148.868                | 233.195 | 408.085 |
|                | 1       | 0.112 ± 0.005             | 47.923 ± 0.899           | 47.145                   | 32.281                 | **45.655** | 77.549 |
|                | 2       | 0.112 ± 0.005             | 252.129 ± 5.768          | 245.421                   | 148.470                | 248.945 | 366.094 |
|                | 3       | 0.113 ± 0.005             | 247.820 ± 5.260          | 242.007                   | 145.381                | 264.145 | 350.913 |
|                | 4       | 0.112 ± 0.005             | 53.755 ± 0.978           | 52.931                    | 35.569                 | 52.759  | 95.705  |
|                | 5       | 0.113 ± 0.005             | 53.408 ± 0.811           | 52.784                    | 33.295                 | 53.193  | 73.774  |
|                | 6       | 0.113 ± 0.005             | 55.586 ± 0.824           | 54.963                    | 34.203                 | 55.012  | 76.677  |
|                | 7       | 0.110 ± 0.005             | 233.996 ± 5.571          | 227.430                   | 143.104                | 233.493 | 388.936 |
|                | 8       | 0.110 ± 0.005             | **46.640** ± 0.647       | **46.189**               | 32.440                 | 46.543  | 64.163  |
|                | 9       | 0.109 ± 0.005             | 238.510 ± 5.316          | 232.407                   | 144.561                | 235.277 | 326.239 |
|                | 10      | 0.109 ± 0.005             | 50.178 ± 0.679           | 49.708                    | 34.546                 | 50.580  | 65.287  |
| *(n = 10^6)*   | 0       | 0.108 ± 0.005             | 2830.535 ± 57.672        | 2773.083                  | 1868.109               | 2654.533 | 4015.562 |
|                | 1       | 0.108 ± 0.005             | 1223.087 ± 13.412        | 1216.033                  | 989.056               | 1225.745 | 1662.859 |
|                | 2       | 0.108 ± 0.005             | 2776.876 ± 53.723        | 2726.519                  | 1940.371              | 2576.609 | 3907.286 |
|                | 3       | 0.110 ± 0.005             | 2781.695 ± 55.039        | 2728.281                  | 1899.844             | 2749.232 | 4377.475 |
|                | 4       | 0.108 ± 0.005             | 1138.666 ± 27.790        | 1117.027                  | 910.320               | 1065.029 | 2873.461 |
|    | Mean ± Std Dev | Mean ± Std Dev | Mean ± Std Dev | Mean ± Std Dev | Mean ± Std Dev |
|----|----------------|----------------|----------------|----------------|----------------|
| 5  | 0.110 ± 0.005  | 1182.486 ± 19.024 | 1170.131       | 939.523        | 1139.444       | 2230.349       |
| 6  | 0.110 ± 0.005  | 1137.827 ± 10.332 | 1133.184       | 917.410        | 1116.388       | 1408.491       |
| 7  | 0.108 ± 0.005  | 2636.415 ± 54.039 | 2583.335       | 1730.813       | 2380.683       | 3738.891       |
| 8  | 0.108 ± 0.005  | 1045.224 ± 8.729  | 1041.659       | 899.932        | 1032.678       | 1261.093       |
| 9  | 0.110 ± 0.005  | 2687.159 ± 50.578 | 2640.935       | 1957.054       | 2510.885       | 3761.409       |
| 10 | 0.110 ± 0.005  | 1090.345 ± 8.418  | 1087.132       | 928.105        | 1076.160       | 1310.540       |

Note: For each problem, the lowest mean and median computation time are highlighted.