Nesterov-Based Alternating Optimization for Nonnegative Tensor Factorization: Algorithm and Parallel Implementation

Athanasios P. Liavas, *Member, IEEE*, Georgios Kostoulas, Georgios Lourakis, Kejun Huang, *Member, IEEE*, and Nicholas D. Sidiropoulos, *Fellow, IEEE*

***Abstract*—We consider the problem of nonnegative tensor factorization. Our aim is to derive an efficient algorithm that is also suitable for parallel implementation. We adopt the alternating optimization framework and solve each matrix nonnegative least-squares problem via a Nesterov-type algorithm for strongly convex problems. We describe a parallel implementation of the algorithm and measure the attained speedup in a multicore computing environment. It turns out that the derived algorithm is a competitive candidate for the solution of very large-scale dense nonnegative tensor factorization problems.**

***Index Terms*—Tensors, nonnegative tensor factorization, optimal first-order optimization algorithms, parallel algorithms.**

I. Introduction

TENSORS are mathematical objects that have recently gained great popularity due to their ability to model multi-way data dependencies [2], [3], [4], [5]. Tensor factorization (or decomposition) into latent factors is very important for numerous tasks, such as feature selection, dimensionality reduction, compression, data visualization and interpretation. Tensor factorizations are usually computed as solutions of optimization problems [2], [3]. The Canonical Decomposition or Canonical Polyadic Decomposition (CANDECOMP or CPD), also known as Parallel Factor Analysis (PARAFAC), and the Tucker Decomposition are the two most widely used tensor factorization models. In this work, we focus on nonnegative PARAFAC, which, for simplicity, we call Nonnegative Tensor Factorization (NTF).

Alternating Optimization (AO), All-at-Once Optimization (AOO), and Multiplicative Updates (MUs) are among the most commonly used techniques for NTF [3], [6]. Recent work for constrained tensor factorization/completion includes, among others, [7], [8], [9], and [10].

In [7], several NTF algorithms and a detailed convergence analysis have been developed. A general framework for joint matrix/tensor factorization/completion has been developed in [8]. In [9], an Alternating Direction Method of Multipliers (ADMM) algorithm for NTF has been derived, and an architecture for its parallel implementation has been outlined. However, the convergence properties of the algorithm in ill-conditioned cases are not favorable, necessitating additional research towards their improvement. In [10], the authors consider constrained matrix/tensor factorization/completion problems. They adopt the AO framework as outer loop and use the ADMM for solving the inner constrained optimization problem for one matrix factor conditioned on the rest. The ADMM offers significant flexibility, due to its ability to efficiently handle a wide range of constraints.

In [11], two parallel algorithms for unconstrained tensor factorization/completion have been developed and results concerning the speedup attained by their Message Passing Interface (MPI) implementations on a multi-core system have been reported. Related work on parallel algorithms for sparse tensor decomposition includes [12] and [13].

A. Contribution

In this work, we focus on dense NTF problems. Our aim is to derive an efficient NTF algorithm, suitable for parallel implementation. We adopt the AO framework and solve each matrix nonnegative least-squares (MNLS) problem via a first-order optimal (Nesterov-type) algorithm for L-smooth μ-strongly convex problems.[1] Then, we describe in detail an MPI implementation of the AO NTF algorithm and measure the speedup

[1]We note that a closely related algorithm for the solution of MNLS problems has been used in [14] and [15]; we explain in detail later the performance improvement offered by our approach.

Manuscript received April 27, 2017; revised September 12, 2017; accepted November 5, 2017. Date of publication November 24, 2017; date of current version January 16, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Tsung-Hui Chang. This work was supported in part by NSF, under Grants IIS-1447788 and IIS-1704074, and in part by computational time granted from the Greek Research & Technology Network in the National HPC facility—ARIS—under project pa170403. Part of this work was presented at IEEE International Conference on Acoustics, Speech and Signal Processing, New Orleans, LA, USA, March 2017. *(Corresponding author: Athanasios P. Liavas.)*

A. P. Liavas, G. Kostoulas, and G. Lourakis are with the School of Electrical and Computer Engineering, Technical University of Crete, Chania 73100, Greece (e-mail: aliavas@isc.tuc.gr; gkostoulas@isc.tuc.gr; glourakis@isc.tuc.gr).

K. Huang is with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: huang663@umn.edu).

N. D. Sidiropoulos was with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA. He is now with the Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904-4204 USA (e-mail: nikos@virginia.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2017.2777399

attained in a multi-core environment. We conclude that the proposed algorithm is a strong candidate for the solution of very large dense NTF problems. A preliminary version of the results contained in this manuscript has appeared in [1].

B. Notation

Vectors and matrices are denoted by small and capital bold letters, for example, $\mathbf{x}$ and $\mathbf{X}$, while tensors are denoted by calligraphic capital letters, for example, $\mathcal{X}$. $\mathbb{R}_+^{I\times J\times K}$ denotes the set of $(I\times J\times K)$ real nonnegative tensors, while $\mathbb{R}_+^{I\times J}$ denotes the set of $(I\times J)$ real nonnegative matrices. $\|\cdot\|_F$ denotes the Frobenius norm of the tensor or matrix argument, $\mathbf{I}$ denotes the identity matrix of appropriate dimensions, and $(\mathbf{A})_+$ denotes the projection of matrix $\mathbf{A}$ onto the set of element-wise nonnegative matrices. The outer product of vectors $\mathbf{a}\in\mathbb{R}^{I\times 1}$, $\mathbf{b}\in\mathbb{R}^{J\times 1}$, and $\mathbf{c}\in\mathbb{R}^{K\times 1}$ is the rank-one tensor $\mathbf{a}\circ\mathbf{b}\circ\mathbf{c}\in\mathbb{R}^{I\times J\times K}$ with elements $(\mathbf{a}\circ\mathbf{b}\circ\mathbf{c})(i,j,k)=\mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$. The Khatri-Rao (columnwise Kronecker) product of compatible matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted as $\mathbf{A}\odot\mathbf{B}$ and the Hadamard (elementwise) product is denoted as $\mathbf{A}\circledast\mathbf{B}$. Finally, inequality $\mathbf{A}\succeq\mathbf{B}$ means that matrix $\mathbf{A}-\mathbf{B}$ is positive semidefinite.

C. Structure

In Section II, we briefly describe the NTF problem. In Section III, we present the Nesterov algorithm for set-constrained L-smooth μ-strongly convex optimization problems and derive a Nesterov-type algorithm for the MNLS problem with proximal term. In Section IV, we present the associated AO NTF algorithm and in Section V we describe in detail a parallel implementation. In Section VI, we test the efficiency of the proposed algorithm with numerical experiments in both serial and parallel computing environments. Finally, in Section VII, we conclude the paper.

II. Nonnegative Tensor Factorization

Let tensor $\mathcal{X}^o\in\mathbb{R}_+^{I\times J\times K}$ admit a factorization of the form

$$\mathcal{X}^o=[\![\mathbf{A}^o,\mathbf{B}^o,\mathbf{C}^o]\!]=\sum_{r=1}^{R}\mathbf{a}_r^o\circ\mathbf{b}_r^o\circ\mathbf{c}_r^o, \tag{1}$$

where $\mathbf{A}^o=[\mathbf{a}_1^o\ \cdots\ \mathbf{a}_R^o]\in\mathbb{R}_+^{I\times R}$, $\mathbf{B}^o=[\mathbf{b}_1^o\ \cdots\ \mathbf{b}_R^o]\in\mathbb{R}_+^{J\times R}$, and $\mathbf{C}^o=[\mathbf{c}_1^o\ \cdots\ \mathbf{c}_R^o]\in\mathbb{R}_+^{K\times R}$. We observe the noisy tensor $\mathcal{X}=\mathcal{X}^o+\mathcal{E}$, where $\mathcal{E}$ is the additive noise. Estimates of $\mathbf{A}^o$, $\mathbf{B}^o$, and $\mathbf{C}^o$ can be obtained by computing matrices $\mathbf{A}\in\mathbb{R}_+^{I\times R}$, $\mathbf{B}\in\mathbb{R}_+^{J\times R}$, and $\mathbf{C}\in\mathbb{R}_+^{K\times R}$ that solve the optimization problem

$$\min_{\mathbf{A}\geq\mathbf{0},\mathbf{B}\geq\mathbf{0},\mathbf{C}\geq\mathbf{0}}\quad f_{\mathcal{X}}(\mathbf{A},\mathbf{B},\mathbf{C}), \tag{2}$$

where $f_{\mathcal{X}}$ is a function measuring the quality of the factorization and the inequalities are element-wise. A common choice for $f_{\mathcal{X}}$ is

$$f_{\mathcal{X}}(\mathbf{A},\mathbf{B},\mathbf{C})=\frac{1}{2}\left\|\mathcal{X}-[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\right\|_F^2. \tag{3}$$

If $\mathcal{Y}=[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$, then its matrix unfoldings, with respect to the first, second, and third mode, are given by [4]

$$\mathbf{Y}_{\mathbf{A}}=\mathbf{A}\,(\mathbf{C}\odot\mathbf{B})^T,\ \mathbf{Y}_{\mathbf{B}}=\mathbf{B}\,(\mathbf{C}\odot\mathbf{A})^T,$$
$$\mathbf{Y}_{\mathbf{C}}=\mathbf{C}\,(\mathbf{B}\odot\mathbf{A})^T.$$

Thus, $f_{\mathcal{X}}$ can be expressed as

$$\begin{aligned}f_{\mathcal{X}}(\mathbf{A},\mathbf{B},\mathbf{C})&=\frac{1}{2}\left\|\mathbf{X}_{\mathbf{A}}-\mathbf{A}\,(\mathbf{C}\odot\mathbf{B})^T\right\|_F^2\\&=\frac{1}{2}\left\|\mathbf{X}_{\mathbf{B}}-\mathbf{B}\,(\mathbf{C}\odot\mathbf{A})^T\right\|_F^2\\&=\frac{1}{2}\left\|\mathbf{X}_{\mathbf{C}}-\mathbf{C}\,(\mathbf{B}\odot\mathbf{A})^T\right\|_F^2.\end{aligned} \tag{4}$$

These expressions form the basis for the AO NTF in the sense that, if we fix two matrix factors, we can update the third by solving an MNLS problem. For reasons related with the conditioning of the MNLS problems, we propose to add a proximal term. More specifically, if $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$ are the estimates of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, respectively, after the k-th AO iteration, then $\mathbf{A}_{k+1}$ is computed as

$$\begin{aligned}\mathbf{A}_{k+1}:=\underset{\mathbf{A}\geq\mathbf{0}}{\operatorname{argmin}}&\frac{1}{2}\left\|\mathbf{X}_{\mathbf{A}}-\mathbf{A}\,(\mathbf{C}_k\odot\mathbf{B}_k)^T\right\|_F^2\\&+\frac{\lambda_k^{\mathbf{A}}}{2}\|\mathbf{A}-\mathbf{A}_k\|_F^2,\end{aligned} \tag{5}$$

where $\lambda_k^{\mathbf{A}}\geq 0$ determines the weight assigned to the proximal term. If $(\mathbf{C}_k\odot\mathbf{B}_k)$ is a well-conditioned matrix, then it is reasonable to put small weight on the proximal term and compute $\mathbf{A}_{k+1}$ that leads to a large decrease of the cost function $f_{\mathcal{X}}(\mathbf{A},\mathbf{B}_k,\mathbf{C}_k)$. If, on the other hand, $(\mathbf{C}_k\odot\mathbf{B}_k)$ is an ill-conditioned matrix, then it is reasonable to put large weight on the proximal term, leading to a better conditioned problem and easy computation of $\mathbf{A}_{k+1}$ that improves the fit in $f_{\mathcal{X}}(\mathbf{A},\mathbf{B}_k,\mathbf{C}_k)$ but is not very far from $\mathbf{A}_k$. This is the strategy we shall follow for the solution of problem (2) (see also [7], [16]).

The computational efficiency of the AO NTF heavily depends on the algorithm we use for the solution of problem (5). In this work, we adopt the approach of Nesterov for the solution of L-smooth μ-strongly convex problems. The derived algorithm is optimal under the black-box first-order oracle framework [17, Chapter 2] and is very efficient in practice. Furthermore, it leads to an AO NTF algorithm that is suitable for parallel implementation.

III. Optimal First-Order Methods for L-Smooth μ-Strongly Convex MNLS Problems

In this section, we present an optimal first-order algorithm for the solution of L-smooth μ-strongly convex MNLS problems. Optimal first-order methods have recently attracted great research interest because they are strong candidates and, in many cases, the only viable way for the solution of very large optimization problems.

A. Optimal First-Order Methods for L-Smooth μ-Strongly Convex Optimization Problems

We consider optimization problems of smooth and strongly convex functions and briefly present results concerning their information complexity and the associated first-order optimal algorithms (for a detailed exposition see [17, Chapter 2]).

We assume that $f:\mathbb{R}^n\to\mathbb{R}$ is a smooth (that is, differentiable up to a sufficiently high order) convex function, with gradient $\nabla f(\mathbf{x})$ and Hessian $\nabla^2 f(\mathbf{x})$. Our aim is to solve the problem

$$\min_{\mathbf{x}} f(\mathbf{x}), \tag{6}$$

within accuracy $\epsilon>0$. The solution accuracy is defined as follows. If $f^*:=\min_{\mathbf{x}}\ f(\mathbf{x})$, then point $\bar{\mathbf{x}}\in\mathbb{R}^n$ solves problem (6) within accuracy ϵ if $f(\bar{\mathbf{x}})-f^*\le\epsilon$.

Let $0<\mu\le L<\infty$. A smooth convex function f is called L-smooth or, using the notation of [17, p. 66], $f\in\mathcal{S}^{\infty,1}_{0,L}$, if

$$\mathbf{0}\preceq\nabla^2 f(\mathbf{x})\preceq L\mathbf{I},\quad\forall\mathbf{x}\in\mathbb{R}^n, \tag{7}$$

and L-smooth μ-strongly convex, or $f\in\mathcal{S}^{\infty,1}_{\mu,L}$, if

$$\mu\mathbf{I}\preceq\nabla^2 f(\mathbf{x})\preceq L\mathbf{I},\ \ \forall\,\mathbf{x}\in\mathbb{R}^n. \tag{8}$$

The number of iterations that first-order methods need for the solution of problem (6), within accuracy ϵ, is $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ if $f\in\mathcal{S}^{\infty,1}_{0,L}$, and $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$ if $f\in\mathcal{S}^{\infty,1}_{\mu,L}$ [17, Theorem 2.2.2]. The convergence rate in the first case is *sublinear* while, in the second case, it is *linear* and determined by the condition number of the problem, $\mathcal{K}:=\frac{L}{\mu}$. Thus, strong convexity is a very important property that should be exploited whenever possible.

An algorithm that achieves this complexity, and, thus, is first-order optimal, appears in Algorithm 1 (see, also [17, p. 80]). This algorithm can handle both the L-smooth case, by setting $q=0$, and the L-smooth μ-strongly convex case, by setting $q=\frac{\mu}{L}>0$.

If the problem of interest is the constrained problem

$$\min_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x}), \tag{9}$$

where $\mathbb{X}$ is a closed convex set, then the corresponding optimal algorithm is very much alike Algorithm 1, with the only difference being in the computation of $\mathbf{x}_{k+1}$. We now have that [17, p. 90]

$$\mathbf{x}_{k+1}=\Pi_{\mathbb{X}}\left(\mathbf{y}_k-\frac{1}{L}\nabla f(\mathbf{y}_k)\right), \tag{10}$$

where $\Pi_{\mathbb{X}}(\cdot)$ denotes the Euclidean projection onto set $\mathbb{X}$. The convergence properties of this algorithm are the same as those of Algorithm 1. If the projection onto set $\mathbb{X}$ is easy to compute, then the algorithm is both theoretically optimal and very efficient in practice.

B. Nesterov-Type Algorithm for MNLS With Proximal Term

In the sequel, we present a Nesterov-type algorithm for the MNLS problem with proximal term. Let $\mathbf{X}\in\mathbb{R}^{m\times n}$, $\mathbf{A}\in\mathbb{R}^{m\times r}$, $\mathbf{B}\in\mathbb{R}^{n\times r}$, and consider the problem

$$\min_{\mathbf{A}\ge\mathbf{0}} f(\mathbf{A}):=\frac{1}{2}\left\|\mathbf{X}-\mathbf{A}\mathbf{B}^T\right\|_F^2. \tag{11}$$

The gradient and Hessian of f, at point $\mathbf{A}$, are, respectively,

$$\nabla f(\mathbf{A})=-\left(\mathbf{X}-\mathbf{A}\mathbf{B}^T\right)\mathbf{B} \tag{12}$$

and

$$\nabla^2 f(\mathbf{A}):=\frac{\partial^2 f(\mathbf{A})}{\partial\mathrm{vec}(\mathbf{A})\partial\mathrm{vec}(\mathbf{A})^T}=\mathbf{B}^T\mathbf{B}\otimes\mathbf{I}\succeq\mathbf{0}. \tag{13}$$

Let $L:=\max(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$ and $\mu:=\min(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$. If $\mu=0$ (for example, if $r>n$), then problem (11) is L-smooth. If $\mu>0$, then problem (11) is L-smooth μ-strongly convex. A first-order optimal algorithm for the solution of (11) can be derived using the approach of Section III-A. We note that [14] and [15] solved problem (11) using a variation of Algorithm 1, which is equivalent to Algorithm 1 with $\mu=0$. However, if $\mu>0$, then this algorithm is *not* first-order optimal and, as we shall see later, it performs much worse than the optimal.

We note that the values of L and μ are necessary for the development of the Nesterov-type algorithm, thus, their computation is imperative.[2]

As we mentioned in Section II, under the AO framework, in order to avoid very ill-conditioned problems (and guarantee strong convexity), we introduce a proximal term and solve problem

$$\min_{\mathbf{A}\ge\mathbf{0}} f_{\mathtt{P}}(\mathbf{A}):=\frac{1}{2}\left\|\mathbf{X}-\mathbf{A}\mathbf{B}^T\right\|_F^2+\frac{\lambda}{2}\|\mathbf{A}-\mathbf{A}_*\|_F^2, \tag{14}$$

for given $\mathbf{A}_*$ and appropriately chosen λ. We choose λ based on L and μ, and denote this functional dependence as $\lambda=g(L,\mu)$. If $\frac{\mu}{L}\ll 1$, then we may set $\lambda\approx 10\mu$, significantly improving the conditioning of the problem by putting large weight on the proximal term; however, in this case, we expect that the optimal point will be biased towards $\mathbf{A}_*$. Otherwise, we may set $\lambda\lessapprox\mu$, putting small weight on the proximal term and permitting significant progress towards the computation of $\mathbf{A}$ that satisfies approximate equality $\mathbf{X}\approx\mathbf{A}\mathbf{B}^T$ as accurately as possible.

The gradient of $f_{\mathtt{P}}$, at point $\mathbf{A}$, is

$$\nabla f_{\mathtt{P}}(\mathbf{A})=-\left(\mathbf{X}-\mathbf{A}\mathbf{B}^T\right)\mathbf{B}+\lambda(\mathbf{A}-\mathbf{A}_*). \tag{15}$$

Algorithm 1: Nesterov Algorithm for L-Smooth μ-Strongly Convex Optimization Problems.

Input: $\mathbf{x}_0\in\mathbb{R}^n$, μ, L. Set $\mathbf{y}_0=\mathbf{x}_0$, $\alpha_0\in(0,1)$, $q=\frac{\mu}{L}$.

1 k-th iteration
2 $\mathbf{x}_{k+1}=\mathbf{y}_k-\frac{1}{L}\nabla f(\mathbf{y}_k)$;
3 $\alpha_{k+1}\in(0,1)$ from $\alpha_{k+1}^2=(1-\alpha_{k+1})\alpha_k^2+q\alpha_{k+1}$;
4 $\beta_{k+1}=\frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$;
5 $\mathbf{y}_{k+1}=\mathbf{x}_{k+1}+\beta_{k+1}(\mathbf{x}_{k+1}-\mathbf{x}_k)$

[2] An alternative to their direct computation is to estimate L using line-search techniques and overcome the computation of μ using heuristic adaptive restart techniques [18]. However, in our case, this alternative is computationally demanding, especially for large-scale problems, and shall not be considered.

Algorithm 2: Nesterov-type algorithm for MNLS with proximal term.

Input: $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times r}$, $\mathbf{A}_* \in \mathbb{R}^{m \times r}$
1 $L = \max(\text{eig}(\mathbf{B}^T\mathbf{B}))$, $\mu = \min(\text{eig}(\mathbf{B}^T\mathbf{B}))$
2 $\lambda = g(L, \mu)$
3 $\mathbf{W} = -\mathbf{X}\mathbf{B} - \lambda\mathbf{A}_*$, $\mathbf{Z} = \mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}$
4 $q = \frac{\mu+\lambda}{L+\lambda}$
5 $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
6 $\alpha_0 = 1$, $k = 0$
7 **while** (1) **do**
8 $\quad \nabla f_{\text{P}}(\mathbf{Y}_k) = \mathbf{W} + \mathbf{Y}_k\mathbf{Z}$
9 $\quad$ **if** (terminating_condition is TRUE) **then**
10 $\quad\quad$ break
11 $\quad$ **else**
12 $\quad\quad \mathbf{A}_{k+1} = \left(\mathbf{Y}_k - \frac{1}{L+\lambda}\nabla f_{\text{P}}(\mathbf{Y}_k)\right)_+$
13 $\quad\quad \alpha_{k+1}^2 = (1 - \alpha_{k+1})\alpha_k^2 + q\alpha_{k+1}$
14 $\quad\quad \beta_{k+1} = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$
15 $\quad\quad \mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta_{k+1}(\mathbf{A}_{k+1} - \mathbf{A}_k)$
16 $\quad\quad k = k + 1$
17 **return** $\mathbf{A}_k$.

The Karush-Kuhn-Tucker (KKT) conditions for problem (14) are [14]

$$\nabla f_{\text{P}}(\mathbf{A}) \geq \mathbf{0},\ \mathbf{A} \geq \mathbf{0},\ \nabla f_{\text{P}}(\mathbf{A}) \circledast \mathbf{A} = \mathbf{0}. \tag{16}$$

These expressions can be used in a terminating condition. For example, we may terminate the algorithm if

$$\min_{i,j}\left([\nabla f_{\text{P}}(\mathbf{A})]_{i,j}\right) > -\delta_1, \max_{i,j}\left(\left|[\nabla f_{\text{P}}(\mathbf{A}) \circledast \mathbf{A}]_{i,j}\right|\right) < \delta_2, \tag{17}$$

for small positive real numbers δ_1 and δ_2. Of course, other criteria, based, for example, on the (relative) change of the cost function can be used in terminating conditions.

A Nesterov-type algorithm for the solution of the MNLS problem with proximal term (14) is given in Algorithm 2. For notational convenience, we denote Algorithm 2 as

$$\mathbf{A}_{\text{opt}} = \text{Nesterov\_MNLS}(\mathbf{X}, \mathbf{B}, \mathbf{A}_*).$$

1) Computational Complexity of Algorithm 2: Quantities $\mathbf{W}$ and $\mathbf{Z}$ are computed once per algorithm call and cost, respectively, $O(mnr)$ and $O(rn^2)$ arithmetic operations. Quantities L and μ are also computed once and cost at most $O(r^3)$ operations. $\nabla f_{\text{P}}(\mathbf{Y}_k)$, $\mathbf{A}_k$, and $\mathbf{Y}_k$ are updated in every iteration with cost $O(mr^2)$, $O(mr)$, and $O(mr)$ arithmetic operations, respectively.

IV. Nesterov-Based AO NTF

In Algorithm 3, we present the Nesterov-based AO NTF. We start from point $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$ and solve, in a circular manner, MNLS problems with proximal terms, based on the previous estimates.

For later use, we note that the most demanding computations during the update of factor matrix $\mathbf{A}_k$ via the Nesterov-type MNLS algorithm are (see line 3 of Algorithm 2)

$$\begin{aligned}\widetilde{\mathbf{W}}_{\mathbf{A}} &:= -\mathbf{X}_{\mathbf{A}}\left(\mathbf{C}_k \odot \mathbf{B}_k\right), \\ \widetilde{\mathbf{Z}}_{\mathbf{A}} &:= \left(\mathbf{C}_k \odot \mathbf{B}_k\right)^T\left(\mathbf{C}_k \odot \mathbf{B}_k\right) \\ &= \left(\mathbf{C}_k^T\mathbf{C}_k\right) \circledast \left(\mathbf{B}_k^T\mathbf{B}_k\right).\end{aligned} \tag{18}$$

Analogous quantities are computed for the updates of $\mathbf{B}_k$ and $\mathbf{C}_k$.

After the updates of the factor matrices, we use two functions which have been proven very useful in our experiments, in the sense that they significantly reduce the number of outer iterations necessary to reach convergence.

Function "Normalize" normalizes each column of $\mathbf{B}_{k+1}$ and $\mathbf{C}_{k+1}$ to unit Euclidean norm, putting all the power on the respective columns of $\mathbf{A}_{k+1}$. We denote its output as $\mathbf{A}_{k+1}^{\mathcal{N}}$, $\mathbf{B}_{k+1}^{\mathcal{N}}$ and $\mathbf{C}_{k+1}^{\mathcal{N}}$.

Function "Accelerate" implements an acceleration mechanism. The development of efficient acceleration mechanisms is a very important research topic, see, for example, [19], [20], but is beyond the scope of this paper. In our experiments, we adopted the simple acceleration technique used in the function `parafac` of the n-way toolbox [21], which is briefly described as follows.

At iteration $k+1 > k_0$, after the computation and normalization of $\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}$, we compute

$$\mathbf{A}_{\text{new}} = \mathbf{A}_k^{\mathcal{N}} + s_{k+1}(\mathbf{A}_{k+1}^{\mathcal{N}} - \mathbf{A}_k^{\mathcal{N}}), \tag{19}$$

where s_{k+1} is a small positive number; a simple choice for s_{k+1} is $s_{k+1} = (k+1)^{\frac{1}{n}}$, where n is initialized as $n = 3$ and its value may change as the algorithm progresses. In an analogous manner, we compute $\mathbf{B}_{\text{new}}$ and $\mathbf{C}_{\text{new}}$. If $f_{\mathcal{X}}(\mathbf{A}_{\text{new}}, \mathbf{B}_{\text{new}}, \mathbf{C}_{\text{new}}) \leq f_{\mathcal{X}}(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1})$, then the acceleration step is successful, and we set $\mathbf{A}_{k+1} = \mathbf{A}_{\text{new}}$, $\mathbf{B}_{k+1} = \mathbf{B}_{\text{new}}$, and $\mathbf{C}_{k+1} = \mathbf{C}_{\text{new}}$. If the acceleration step fails, then it is ignored and we set $\mathbf{A}_{k+1} = \mathbf{A}_{k+1}^{\mathcal{N}}$, $\mathbf{B}_{k+1} = \mathbf{B}_{k+1}^{\mathcal{N}}$, and $\mathbf{C}_{k+1} = \mathbf{C}_{k+1}^{\mathcal{N}}$ as input to the next AO update. If the acceleration step fails for n_0 iterations, then we set $n = n + 1$, thus, decreasing the exponent of the acceleration step. Typical values of k_0 and n_0 are $k_0 = 5$ and $n_0 = 5$.

It has been shown in [16] that the AO NTF algorithm with proximal term falls under the block successive upper bound minimization (BSUM) framework, which ensures convergence to a stationary point of problem (2).

We can use various termination criteria for the AO NTF algorithm based, for example, on the (relative) change of the cost function and/or the latent factors.

V. Parallel Implementation of AO NTF

In this section, we assume that we have at our disposal $p = p_{\mathbf{A}} \times p_{\mathbf{B}} \times p_{\mathbf{C}}$ processing elements and describe a parallel implementation of the Nesterov-based AO NTF algorithm, which has been motivated by the medium-grained approach

Algorithm 3: Nesterov-based AO NTF.

Input: $\mathcal{X}$, $\mathbf{A}_0 > \mathbf{0}$, $\mathbf{B}_0 > \mathbf{0}$, $\mathbf{C}_0 > \mathbf{0}$.

Set $k = 0$
while (1) **do**
  $\mathbf{A}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{X}_\mathbf{A}, (\mathbf{C}_k \odot \mathbf{B}_k), \mathbf{A}_k)$
  $\mathbf{B}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{X}_\mathbf{B}, (\mathbf{C}_k \odot \mathbf{A}_{k+1}), \mathbf{B}_k)$
  $\mathbf{C}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{X}_\mathbf{C}, (\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}), \mathbf{C}_k)$
  $(\mathbf{A}^\mathcal{N}_{k+1}, \mathbf{B}^\mathcal{N}_{k+1}, \mathbf{C}^\mathcal{N}_{k+1}) = \text{Normalize}(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1})$
  if (terminating_condition is TRUE) **then** break; **endif**
  $(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1}) = \text{Accelerate}(\mathbf{A}^\mathcal{N}_{k+1}, \mathbf{A}^\mathcal{N}_k, \mathbf{B}^\mathcal{N}_{k+1}, \mathbf{B}^\mathcal{N}_k, \mathbf{C}^\mathcal{N}_{k+1}, \mathbf{C}^\mathcal{N}_k)$
  $k = k + 1$
return $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.

of [12].[3] The p processors form a three-dimensional Cartesian grid and are denoted as $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$, and $i_\mathbf{C} = 1, \ldots, p_\mathbf{C}$.

A. Variable Partitioning and Data Allocation

In order to describe the parallel implementation, we introduce certain partitionings of the factor matrices and the tensor matricizations. We partition the factor matrix $\mathbf{A}_k$ into $p_\mathbf{A}$ block rows as

$$\mathbf{A}_k = \begin{bmatrix} \left(\mathbf{A}_k^1\right)^T & \cdots & \left(\mathbf{A}_k^{p_\mathbf{A}}\right)^T \end{bmatrix}^T, \tag{21}$$

with $\mathbf{A}_k^{i_\mathbf{A}} \in \mathbb{R}^{\frac{I}{p_\mathbf{A}} \times R}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$. We partition accordingly the matricization $\mathbf{X}_\mathbf{A}$ and get

$$\mathbf{X}_\mathbf{A} = \begin{bmatrix} \left(\mathbf{X}_\mathbf{A}^1\right)^T & \cdots & \left(\mathbf{X}_\mathbf{A}^{p_\mathbf{A}}\right)^T \end{bmatrix}^T, \tag{22}$$

with $\mathbf{X}_\mathbf{A}^{i_\mathbf{A}} \in \mathbb{R}^{\frac{I}{p_\mathbf{A}} \times JK}$. In a similar manner, we partition $\mathbf{B}_k$ and $\mathbf{X}_\mathbf{B}$ into $p_\mathbf{B}$ block rows, each of size $\frac{J}{p_\mathbf{B}} \times R$ and $\frac{J}{p_\mathbf{B}} \times IK$, respectively, and $\mathbf{C}_k$ and $\mathbf{X}_\mathbf{C}$ into $p_\mathbf{C}$ block rows, each of size $\frac{K}{p_\mathbf{C}} \times R$ and $\frac{K}{p_\mathbf{C}} \times IJ$, respectively.

We partition tensor $\mathcal{X}$ into p subtensors, according to the partitioning of the factor matrices (see Fig. 1), and allocate its parts to the various processors, so that processor $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$ receives subtensor $\mathcal{X}^{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$, defined in (20), at the bottom of this page.

We assume that, at the end of the k-th outer AO iteration,

a) processor $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$ knows $\mathbf{A}_k^{i_\mathbf{A}}$, $\mathbf{B}_k^{i_\mathbf{B}}$, and $\mathbf{C}_k^{i_\mathbf{C}}$;
b) *all* processors know $\mathbf{A}_k^T\mathbf{A}_k$, $\mathbf{B}_k^T\mathbf{B}_k$, and $\mathbf{C}_k^T\mathbf{C}_k$.

B. Communication Groups

We define certain communication groups, also known as communicators [22], over subsets of the p processors, which are used for the efficient collaborative implementation of specific computational tasks, as explained in detail later.

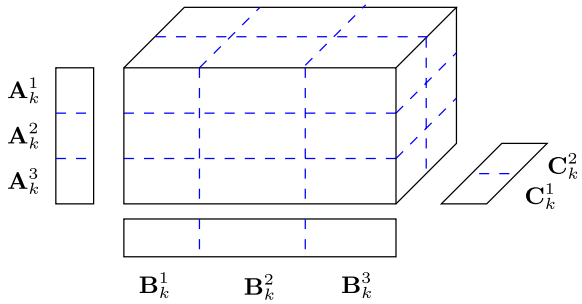


Fig. 1. Tensor $\mathcal{X}$, factors $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$, and their partitioning for $p_\mathbf{A} = p_\mathbf{B} = 3$ and $p_\mathbf{C} = 2$.

First, we define $p_\mathbf{A}$ two-dimensional processor groups, each involving the $p_\mathbf{B} \times p_\mathbf{C}$ processors $p_{i_\mathbf{A}, :, :}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$ (horizontal layers), with the $i_\mathbf{A}$-th processor group used for the collaborative update of $\mathbf{A}_k^{i_\mathbf{A}}$. Similarly, we define groups $p_{:, i_\mathbf{B}, :}$, for $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$, and $p_{:, :, i_\mathbf{C}}$, for $i_\mathbf{C} = 1, \ldots, p_\mathbf{C}$, which are used for the collaborative update of $\mathbf{B}_k^{i_\mathbf{B}}$ and $\mathbf{C}_k^{i_\mathbf{C}}$, respectively.

We define $p_\mathbf{B} \times p_\mathbf{C}$ one-dimensional processor groups, each involving the $p_\mathbf{A}$ processors $p_{:, i_\mathbf{B}, i_\mathbf{C}}$. Each of these groups is used for the collaborative computation of $\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}$. Similarly, we define groups $p_{i_\mathbf{A}, :, i_\mathbf{C}}$ and $p_{i_\mathbf{A}, i_\mathbf{B}, :}$, which are used for the collaborative computation of $\mathbf{B}_{k+1}^T\mathbf{B}_{k+1}$ and $\mathbf{C}_{k+1}^T\mathbf{C}_{k+1}$, respectively.

C. Factor Update Implementation

We describe in detail the update of $\mathbf{A}_k$, which is achieved via the parallel updates of $\mathbf{A}_k^{i_\mathbf{A}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, and consists of the following stages:

1) Processors $p_{i_\mathbf{A}, :, :}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, collaboratively compute the $\frac{I}{p_\mathbf{A}} \times R$ matrix

$$\widetilde{\mathbf{W}}_\mathbf{A}^{i_\mathbf{A}} = -\mathbf{X}_\mathbf{A}^{i_\mathbf{A}} (\mathbf{C}_k \odot \mathbf{B}_k), \tag{23}$$

[3] We note that both the single-core and the multi-core implementations solve the same problem, thus problems that are identifiable in single-core environments remain identifiable in multi-core environments and the solutions, in both cases, are practically the same.

$$\mathcal{X}^{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}} := \mathcal{X}\left((i_\mathbf{A} - 1)\frac{I}{p_\mathbf{A}} + 1 : i_\mathbf{A}\frac{I}{p_\mathbf{A}}, (i_\mathbf{B} - 1)\frac{J}{p_\mathbf{B}} + 1 : i_\mathbf{B}\frac{J}{p_\mathbf{B}}, (i_\mathbf{C} - 1)\frac{K}{p_\mathbf{C}} + 1 : i_\mathbf{C}\frac{K}{p_\mathbf{C}}\right). \tag{20}$$

and the result is scattered among the processors in the group; thus, each processor in the group receives $\frac{I}{p_{\mathbf{A}}\, p_{\mathbf{B}}\, p_{\mathbf{C}}}$ successive rows of $\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}}$. Term $\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}}$ can be computed collaboratively because

$$\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}}}\,(\mathbf{C}_k \odot \mathbf{B}_k) = \sum_{i_{\mathbf{B}}=1}^{p_{\mathbf{B}}} \sum_{i_{\mathbf{C}}=1}^{p_{\mathbf{C}}} \mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}\,(\mathbf{C}_k^{i_{\mathbf{C}}} \odot \mathbf{B}_k^{i_{\mathbf{B}}}), \tag{24}$$

where $\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ is the matricization of $\mathcal{X}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$, with respect to the first mode. Processor $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ knows $\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$, $\mathbf{B}_k^{i_{\mathbf{B}}}$, and $\mathbf{C}_k^{i_{\mathbf{C}}}$, and computes the corresponding term of (24). The sum is computed and scattered among processors $p_{i_{\mathbf{A}},:,:}$ via a reduce-scatter operation.

2) Each processor in the group $p_{i_{\mathbf{A}},:,:}$ uses the scattered part of $\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}}$, $\widetilde{\mathbf{Z}}_{\mathbf{A}} = \mathbf{C}_k^T\mathbf{C}_k \circledast \mathbf{B}_k^T\mathbf{B}_k$, and $\mathbf{A}_k^{i_{\mathbf{A}}}$, and computes the updated part of $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$, via the `while` loop of the Nesterov MNLS algorithm.
3) The updated parts of $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$ are all-gathered at the processors of the group $p_{i_{\mathbf{A}},:,:}$, so that *all* processors in the group learn the updated $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$.
4) By applying an all-reduce operation to $\left(\mathbf{A}_{k+1}^{i_{\mathbf{A}}}\right)^T \mathbf{A}_{k+1}^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1,\ldots,p_{\mathbf{A}}$, on each of the single-dimensional processor groups $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$, for $i_{\mathbf{B}} = 1,\ldots,p_{\mathbf{B}}$ and $i_{\mathbf{C}} = 1,\ldots,p_{\mathbf{C}}$, *all* p processors learn $\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}$.[4]

The updates of $\mathbf{B}_k$ and $\mathbf{C}_k$ are implemented by following analogous steps.

The Euclidean norms of the columns of $\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}$ appear on the diagonals of $\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}^T\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}^T\mathbf{C}_{k+1}$, which are known to all processors. Thus, no communication is necessary for the normalization of the updated matrix factors.

After the normalization step of the $(k+1)$-st AO iteration, processor $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ knows the parts of the normalized factors, that is, $\mathbf{A}_{k+1}^{i_{\mathbf{A}}\mathcal{N}}$, $\mathbf{B}_{k+1}^{i_{\mathbf{B}}\mathcal{N}}$, $\mathbf{C}_{k+1}^{i_{\mathbf{C}}\mathcal{N}}$, as well as $\mathbf{A}_k^{i_{\mathbf{A}}\mathcal{N}}$, $\mathbf{B}_k^{i_{\mathbf{B}}\mathcal{N}}$, and $\mathbf{C}_k^{i_{\mathbf{C}}\mathcal{N}}$, and can compute $\mathbf{A}_{\text{new}}^{i_{\mathbf{A}}}$, $\mathbf{B}_{\text{new}}^{i_{\mathbf{B}}}$, and $\mathbf{C}_{\text{new}}^{i_{\mathbf{C}}}$ (see (19)). The computation of the cost function $f_{\mathcal{X}}$ at points $(\mathbf{A}_{k+1},\mathbf{B}_{k+1},\mathbf{C}_{k+1})$ and $(\mathbf{A}_{\text{new}},\mathbf{B}_{\text{new}},\mathbf{C}_{\text{new}})$ is implemented collaboratively. Each processing element computes its local contribution and, via an all-reduce operation over the whole processor grid, the values of the cost function are computed and become known to all processors, thus, all processors make the same decision regarding the success or failure of the acceleration step.

D. Communication Cost

We focus on the parallel updates of $\mathbf{A}_k^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1,\ldots,p_{\mathbf{A}}$, and present results concerning the associated communication cost. Analogous results hold for the updates of $\mathbf{B}_k^{i_{\mathbf{B}}}$ and $\mathbf{C}_k^{i_{\mathbf{C}}}$.

We assume that an m-word message is transferred from one process to another with communication cost $t_s + t_w m$, where t_s is the latency, or startup time for the data transfer, and t_w is the word transfer time [22].

Communication occurs at three algorithm execution points.

1) The $\frac{I}{p_{\mathbf{A}}} \times R$ matrix $\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}}$ is computed and scattered among the $p_{\mathbf{B}} \times p_{\mathbf{C}}$ processors of group $p_{i_{\mathbf{A}},:,:}$, using a reduce-scatter operation, with communication cost [22, Section 4.2]

$$\mathcal{C}_1^{\mathbf{A}} = t_s\,(p_{\mathbf{B}}\ + p_{\mathbf{C}} - 2) + t_w \frac{IR}{p_{\mathbf{A}} p_{\mathbf{B}} p_{\mathbf{C}}}\,(p_{\mathbf{B}} p_{\mathbf{C}} - 1)\,.$$

2) Processors $p_{i_{\mathbf{A}},:,:}$ learn the updated $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$ through an all-gather operation on its updated parts, each of dimension $\frac{I}{p_{\mathbf{A}}\, p_{\mathbf{B}}\, p_{\mathbf{C}}} \times R$, with communication cost [22, Section 4.2]

$$\mathcal{C}_2^{\mathbf{A}} = t_s\,(p_{\mathbf{B}}\ + p_{\mathbf{C}} - 2) + t_w \frac{IR}{p_{\mathbf{A}} p_{\mathbf{B}} p_{\mathbf{C}}}\,(p_{\mathbf{B}} p_{\mathbf{C}} - 1)\,.$$

3) Finally, $\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}$ is computed by using an all-reduce operation on quantities $\left(\mathbf{A}_{k+1}^{i_{\mathbf{A}}}\right)^T \mathbf{A}_{k+1}^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1,\ldots,p_{\mathbf{A}}$, on each single-dimensional processor group $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$, with communication cost [22, Section 4.3]

$$\mathcal{C}_3^{\mathbf{A}} = \left(t_s + t_w R^2\right)\log_2 p_{\mathbf{A}}\,. \tag{25}$$

The communication that takes place during the acceleration step involves scalar quantities and, thus, is ignored.

When we are dealing with large messages, the t_w terms dominate the communication cost. Thus, if we ignore the startup time, the total communication time is

$$\begin{aligned}\mathcal{C}^{\mathbf{A}} &= t_w \left(\frac{2IR}{p_{\mathbf{A}}\, p_{\mathbf{B}}\, p_{\mathbf{C}}}\,(p_{\mathbf{B}} p_{\mathbf{C}} - 1) + R^2 \log_2 p_{\mathbf{A}}\right)\\ &\approx t_w \left(\frac{2IR}{p_{\mathbf{A}}} + R^2 \log_2 p_{\mathbf{A}}\right)\\ &\approx \frac{2IR\,t_w}{p_{\mathbf{A}}},\end{aligned} \tag{26}$$

with the second approximation being accurate for $R \ll \frac{I}{p_{\mathbf{A}}}$. The presence of $p_{\mathbf{A}}$ in the denominator of the last expression of (26) implies that our implementation is scalable in the sense that, if we double I, then we can have (approximately) the same communication cost per processor by doubling $p_{\mathbf{A}}$.

Analogous results hold for the updates of $\mathbf{B}_k$ and $\mathbf{C}_k$.

VI. Numerical Experiments

A. Matlab Environment

In this subsection, we test the effectiveness of the Nesterov-based AO NTF algorithm with numerical experiments performed in Matlab.

At first, we compare the performance of the algorithm we propose in Algorithm 2 for the solution of the MNLS problem (11) with that of the algorithm proposed in [14] and [15] (for the moment, we ignore the proximal term, thus, we put $\lambda = 0$ in Algorithm 2). As we mentioned in subsection III-B, if matrix $\mathbf{B}^T\mathbf{B}$ is rank deficient, that is, if $\mu = 0$, then both algorithms have practically the same behavior. However, if $\mathbf{B}^T\mathbf{B}$ is full-rank, then the two algorithms exhibit different behavior.

[4] In the cases where $R \gtrapprox \frac{I}{p_{\mathbf{A}}}$ it seems preferable to compute $\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}$ via an all-gather operation on terms $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1,\ldots,p_{\mathbf{A}}$, on each of the single-dimensional processor groups $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$. However, in this work, we mainly focus on small-rank factorizations, thus, in our communication cost analysis and experiments we do not present results for this alternative.

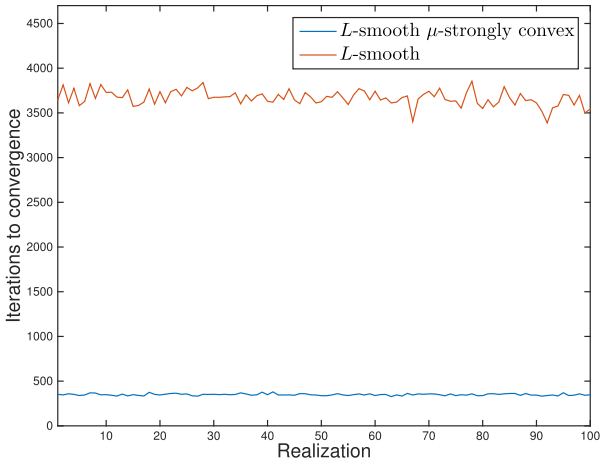


Fig. 2. Number of iterations to convergence for (blue line) Algorithm 2, with $\lambda = 0$, and (red line) algorithm of [14].

In order to illustrate their difference, we perform the following experiment. We generate random matrices $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times r}$ with $m = 300$, $n = 200$, and $r = 100$, with independent and identically distributed (i.i.d) elements, taking values uniformly at random in the interval $[0, 1]$. Then, we solve problem (11) with the two algorithms, starting from the same random point. The terminating conditions are determined by parameters $\delta_1 = \delta_2 = 10^{-3}$. In Fig. 2, we plot the number of iterations needed by the two algorithms to converge over 100 independent realizations. We observe that the Nesterov-type algorithm which exploits strong convexity is much more efficient than the algorithm which does not. Thus, in the sequel, we shall not present performance results involving the algorithm of [14].

Next, we compare the performance of a Matlab implementation of the proposed algorithm with routines `parafac` of the n-way toolbox [21] and `sdf_nls` of tensorlab [23]. Our aim is to provide some general observations about the difficulty of the problems and the behavior of the algorithms and not a strict ranking of the algorithms.[5]

The `parafac` routine essentially implements an AO NTF algorithm, where each MNLS problem is solved via the function `fastnnls`, which is based on [24, Section 23.3]. It also incorporates the normalization and acceleration schemes briefly described in Section IV. The `sdf_nls` routine for NTF first applies a "squaring" transformation to the problem variables [25] and then solves an unconstrained problem via an AOO-based Gauss-Newton method.

In our experiments with synthetic data, we focus on the `cputime` and the Maximum, over the three latent factors, Relative Factor Error (MRFE), which is computed via function `cpd_err` of tensorlab.

In the numerical experiments we present in this subsection, we choose the parameter values that determine the terminating conditions so that all algorithms achieve (approximately) the same average MRFEs (of course, this is not always possible with one set of parameter values). Thus, we set $\mathrm{Tol} = 10^{-5}$ for `parafac`, $\mathrm{TolFun} = 10^{-9}$ for `sdf_nls`, and δ_1 and δ_2, which determine the terminating conditions for the Nesterov-based MNLS, are set to $\delta_1 = \delta_2 = 10^{-2}$. The outer iterations of the Nesterov-based AO NTF terminate if the relative changes of the normalized latent factors become sufficiently small, that is,

$$\frac{\|\mathbf{M}_{k+1}^{\mathcal{N}} - \mathbf{M}_{k}^{\mathcal{N}}\|_F}{\|\mathbf{M}_{k}^{\mathcal{N}}\|_F} < \mathrm{tol}_{\mathrm{AO}}, \text{ for } \mathbf{M} = \mathbf{A}, \mathbf{B}, \mathbf{C}, \tag{27}$$

where $\mathrm{tol}_{\mathrm{AO}} = 10^{-4}$.

The proximal parameter λ is computed as

$$\lambda := g(L, \mu) = \begin{cases} 10\mu, & \text{if } \frac{L}{\mu} > 10^6, \\ \mu, & \text{if } 10^6 > \frac{L}{\mu} > 10^4, \\ \frac{\mu}{10}, & \text{if } 10^4 > \frac{L}{\mu}. \end{cases} \tag{28}$$

All algorithms start from the same triple of random matrices, $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$, which have i.i.d. elements, uniformly distributed in $[0, 1]$.

1) True Latent Factors With i.i.d. Elements: We start with synthetic data by assuming that the true latent factors consist of i.i.d. elements, uniformly distributed in $[0, 1]$. The additive noise is zero-mean white Gaussian with variance σ_N^2.

In Table I, we present the average, over 10 realizations, `cputime` and MRFE for various tensor "shapes," ranks $R = 15, 50$, and noise variances $\sigma_N^2 = 10^{-2}, 10^{-4}$. We observe that the Nesterov-based AO NTF is very competitive in all cases, in the sense that it converges fast, achieving very good accuracy in most of the cases.

2) True Latent Factors With Correlated Elements: It is well-known that, if some columns of (at least) one latent factor are almost collinear, convergence of the AO algorithm tends to be slow (these cases are known as "bottlenecks") [19]. In the sequel, we test the behavior of the three algorithms in cases with one, two, and three bottlenecks. More specifically, we generate the true latent factors with i.i.d. elements as before and we create a single "bottleneck" by modifying the last two columns of one latent factor so that each becomes highly correlated with another column of the same latent factor (the correlation coefficient is larger that 0.98). In an analogous way, we generate double and triple "bottlenecks."

In Table II, we focus on the case $I = J = K = 300$, $R = 50$, $\sigma_N^2 = 10^{-4}$, and present the average, over 10 realizations, `cputime` and MRFE. We observe that the problems become more difficult as the number of bottlenecks increases, in the sense that both the `cputime` and the MRFE increase as the number of bottlenecks increases. Again, the Nesterov-based AO NTF algorithm is very efficient in all cases. Analogous observations have been made in extensive numerical experiments with other tensor shapes and noise levels.

3) Real-World Data: In order to test the behavior of the aforementioned algorithms with real-world data, we use the tensor with size $1021 \times 1343 \times 33$ derived from the hyperspectral image "Souto_Wood_Pile" [26]. Since, in this case, the true latent factors are unknown, we focus on the `cputime` and the

[5]For our experiments, we run Matlab 2014a on a MacBook Pro with a 2.5 GHz Intel Core i7 Intel processor and 16 GB RAM.

TABLE I
AVERAGE, OVER 10 REALIZATIONS, `cputime` AND MAXIMUM RELATIVE FACTOR ERROR FOR NESTEROV-BASED AO NTF, `sdf_nls`, AND `parafac`, FOR TRUE LATENT FACTORS WITH I.I.D. ENTRIES, UNIFORM IN $[0, 1]$

Size	R	σ_N^2	AO-Nesterov		`sdf_nls`		`parafac`	
			`cputime`	`MRFE` $\times 10^4$	`cputime`	`MRFE` $\times 10^4$	`cputime`	`MRFE` $\times 10^4$
$1000 \times 100 \times 100$	15	10^{-2}	29	80	56	79	44	85
		10^{-4}	27	10	52	13	53	8
	50	10^{-2}	77	89	217	91	191	91
		10^{-4}	76	13	221	24	251	9
$500 \times 500 \times 100$	15	10^{-2}	63	35	126	37	72	42
		10^{-4}	64	5	132	10	105	4
	50	10^{-2}	119	39	347	43	250	42
		10^{-4}	124	8	331	20	327	5
$300 \times 300 \times 300$	15	10^{-2}	72	27	84	27	70	38
		10^{-4}	71	5	87	7	106	3
	50	10^{-2}	114	31	171	32	230	34
		10^{-4}	119	8	174	13	279	4

TABLE II
AVERAGE, OVER 10 REALIZATIONS, `cputime` AND MAXIMUM RELATIVE FACTOR ERROR FOR NESTEROV-BASED AO NTF, `sdf_nls`, AND `parafac`, FOR TRUE LATENT FACTORS WITH CORRELATED ENTRIES

Size	R	σ_N^2	Bottleneck	AO-Nesterov		`sdf_nls`		`parafac`	
				`cputime`	`MRFE`	`cputime`	`MRFE`	`cputime`	`MRFE`
$300 \times 300 \times 300$	50	10^{-4}	$\mathbf{A}$	132	0.0074	194	0.0075	356	0.0073
			$\mathbf{A}, \mathbf{B}$	204	0.0116	254	0.0195	412	0.0122
			$\mathbf{A}, \mathbf{B}, \mathbf{C}$	271	0.0206	370	0.1007	779	0.0168

TABLE III
`cputime` AND RELATIVE FACTORIZATION ERROR FOR NESTEROV-BASED AO NTF, `sdf_nls`, AND `parafac`, FOR REAL-WORLD DATA

Size	R	AO-Nesterov		`sdf_nls`		`parafac`	
		`cputime`	`RFE`	`cputime`	`RFE`	`cputime`	`RFE`
$1021 \times 1343 \times 33$	10	127	0.2361	1984	0.2483	156	0.2349
	20	409	0.1741	2660	0.2183	421	0.1738
	30	518	0.1446	3072	0.2202	547	0.1444

Relative Factorization Error (RFE), defined as

$$\texttt{RFE}(\mathbf{A}, \mathbf{B}, \mathbf{C}) := \frac{\|\mathcal{X} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_F}{\|\mathcal{X}\|_F}.$$

In Table III, we present the average `cputime` and `RFE` for ranks $R = 10, 20, 30$. The averages are with respect to the initial points $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$, which are random with i.i.d. elements uniformly distributed in $[0, 1]$, and are computed over 5 realizations. We observe that the Nesterov-based AO NTF is very efficient in these cases as well.

B. Parallel Environment - MPI

We now present results obtained from the MPI implementation described in detail in Section V. The program is executed on a DELL PowerEdge R820 system with SandyBridge - Intel(R) Xeon(R) CPU E5 $-$ 4650v2 (in total, 16 nodes with 40 cores each at 2.4 Gz) and 512 GB RAM per node. The matrix operations are implemented using routines of the C++ library Eigen [27]. We assume a noiseless tensor $\mathcal{X}$, whose true latent factors have i.i.d elements, uniformly distributed in $[0, 1]$. The terminating conditions for MNLS are determined by values $\delta_1 = \delta_2 = 10^{-2}$.

The AO terminates at iteration k if (recall that tensor $\mathcal{X}$ is noiseless)

$$\texttt{RFE}(\mathbf{A}_k, \mathbf{B}_k, \mathbf{C}_k) < 10^{-3}.$$

We test the behavior of our implementation for various tensor sizes and rank $R = 15, 50, 100$. The performance metric we compute is the speedup attained using $p = p_{\mathbf{A}} \times p_{\mathbf{B}} \times p_{\mathbf{C}}$ processors.

In Figs. 3–6, we plot the speedup for the following cases (in all cases with synthetic data, the tensor $\mathcal{X}$ has eight billion entries):

1) Cubic tensor: we set $I = J = K = 2000$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{B}} = p_{\mathbf{C}} = \sqrt[3]{p}$, for $p = 1, 8, 27, 64, 125, 216, 343, 512$.
2) One large dimension: we set $I = 400$, $J = 400$, $K = 50000$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{B}} = 1$, $p_{\mathbf{C}} = p$, for $p = 1, 8, 27, 64, 125, 216, 343, 512$.
3) Two large dimensions: we set $I = 5000$, $J = 320$, $K = 5000$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{C}} = \sqrt{p}$, $p_{\mathbf{B}} = 1$, for $p = 1, 9, 36, 64, 121, 225, 361, 529$.
4) Finally, we use the hyperspectral image from the previous section, with $I = 1021$, $J = 1343$, $K = 33$, and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{B}} = \sqrt{p}$, $p_{\mathbf{C}} = 1$, for $p = 1, 9, 36, 64$.

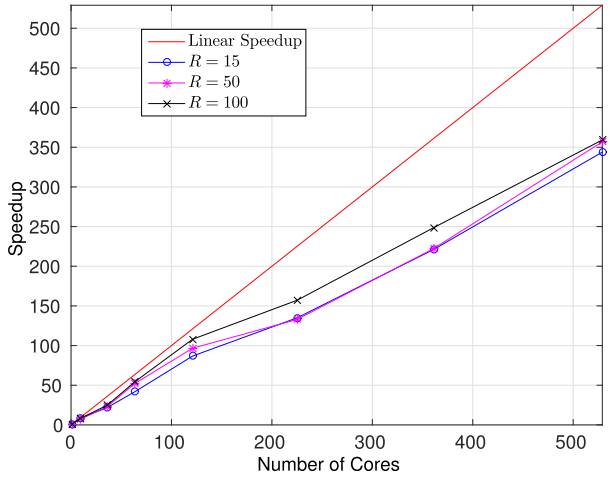


Fig. 3. Speedup achieved for a $2000 \times 2000 \times 2000$ tensor with p cores, for $p = 1, 8, 27, 64, 125, 216, 343, 512$.

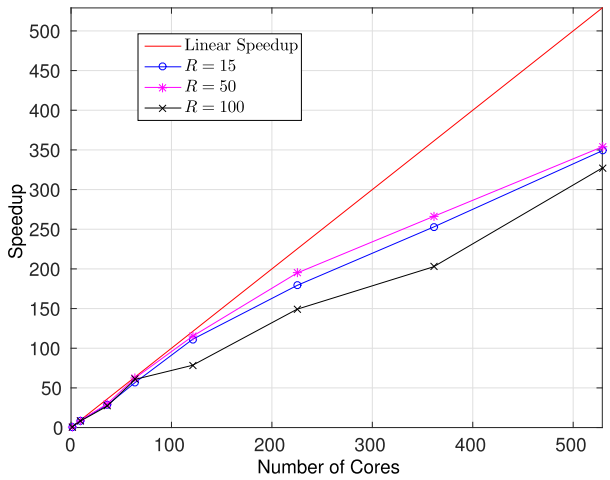


Fig. 4. Speedup achieved for a $400 \times 400 \times 50000$ tensor with p cores, for $p = 1, 8, 27, 64, 125, 216, 343, 512$.

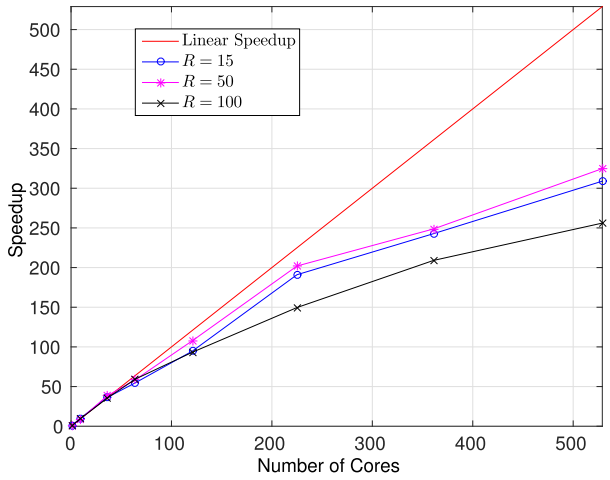


Fig. 5. Speedup achieved for a $5000 \times 320 \times 5000$ tensor with p cores, for $p = 1, 9, 36, 64, 121, 225, 361, 529$.

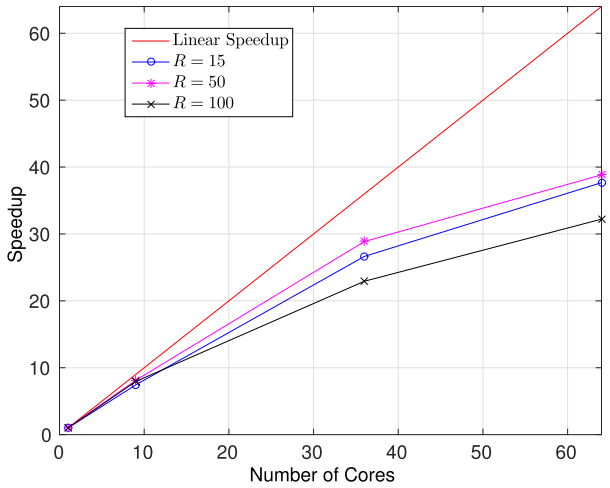


Fig. 6. Speedup achieved for the hyperspectral image "Souto_Wood_Pile" [26] with p cores, for $p = 1, 9, 36, 64$.

We observe that, in all cases, we attain significant speedup, which is rather insensitive to the tensor shape and rank.

VII. Conclusion

We considered the NTF problem. We adopted the AO framework and solved each MNLS problem via a Nesterov-type algorithm for smooth and strongly convex problems. We described in detail a parallel implementation of the algorithm on a three-dimensional processor grid. In extensive numerical experiments, the derived algorithm was proven very efficient, compared with state-of-the-art competitors. Our parallel implementation attained significant speedup, rendering our algorithm a strong candidate for the solution of very large-scale dense NTF problems.

References

[1] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based parallel algorithm for large-scale nonnegative tensor factorization," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, New Orleans, LA, USA, Mar. 5–9, 2017, pp. 5895–5899.

[2] P. M. Kroonenberg, *Applied Multiway Data Analysis*. Hoboken, NJ, USA: Wiley, 2008.

[3] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations*. Hoboken, NJ, USA: Wiley, 2009.

[4] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Sep. 2009.

[5] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, Jul. 2017.

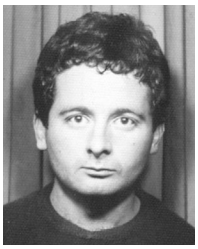
[6] A. Cichocki *et al.*, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, Mar. 2015.

[7] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imaging Sci.*, vol. 6, no. 3, pp. 1758–1789, 2013.

[8] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion," *IEEE J. Sel. Topics Signal Process.*, vol. 9, no. 4, pp. 586–600, Jun. 2015.

[9] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers," *IEEE Trans. Signal Process.*, vol. 63, no. 20, pp. 5450–5463, Oct. 2015.

[10] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient framework for constrained matrix and tensor factorization," *IEEE Trans. Signal Process.*, vol. 64, no. 19, pp. 5052–5065, Oct. 2016.
[11] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format," *Parallel Comput.*, vol. 57, pp. 222–234, 2016.
[12] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," in *Proc. 30th IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 902–911.
[13] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015, pp. 1–11.
[14] N. Guan, D. Tao, Z. Luo, and B. Yuan, "NeNMF: An optimal gradient method for nonnegative matrix factorization," *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 2882–2898, Jun. 2012.
[15] Y. Zhang, G. Zhou, Q. Zhao, A. Cichocki, and X. Wang, "Fast nonnegative tensor factorization based on accelerated proximal gradient and low-rank approximation," *Neurocomputing*, vol. 198, pp. 148–154, 2016.
[16] M. Razaviyayn, M. Hong, and Z.-Q. Luo, "A unified convergence analysis of block successive minimization methods for nonsmooth optimization," *SIAM J. Optim.*, vol. 23, no. 2, pp. 1126–1153, 2013.
[17] Y. Nesterov, *Introductory Lectures on Convex Optimization*. Norwell, MA, USA: Kluwer, 2004.
[18] B. O'Donoghue and E. Candes, "Adaptive restart for accelerated gradient schemes," *Found. Comput. Math.*, vol. 15, no. 3, pp. 715–732, 2015.
[19] M. Rajih, P. Comon, and R. A. Harshman, "Enhanced line search: A novel method to accelerate parafac," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1128–1147, 2008.
[20] L. Sorber, I. Domanov, M. Van Barel, and L. De Lathauwer, "Exact line and plane search for tensor optimization," *Comput. Optim. Appl.*, vol. 63, no. 1, pp. 121–142, 2016.
[21] C. A. Andersson and R. Bro, "The n-way toolbox for Matlab," 2000. [Online]. Available: http://www.models.life.ku.dk/source/nwaytoolbox
[22] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing*, 2nd ed. London, U.K.: Pearson, 2003.
[23] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab 3.0," Mar. 2016. [Online]. Available: http://www.tensorlab.net/
[24] C. L. Lawson and R. J. Hanson, *Solving Lease Squares Problems*. Philadelphia, PA, USA: SIAM, 1995.
[25] J.-P. Royer, N. Thirion-Moreau, and P. Comon, "Computing the polyadic decomposition of nonnegative third order tensors," *Signal Process.*, vol. 91, no. 9, pp. 2159–2171, 2011.
[26] S. M. Nascimento, K. Amano, and D. H. Foster, "Spatial distributions of local illumination color in natural scenes," *Vis. Res.*, vol. 120, pp. 39–44, 2016.
[27] G. Guennebaud *et al.*, "Eigen v3." [Online]. Available: http://eigen.tuxfamily.org, 2010



Athanasios P. Liavas (M'94) received the Diploma and the Ph.D. degrees from the Department of Computer Engineering and Informatics, University of Patras, Patras, Greece, in 1989 and 1993, respectively. From 2001 to 2004, he served as an Assistant Professor with the Department of Mathematics, University of the Aegean. Since 2004, he has been with the School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece. He is currently a Professor and the Dean of the School. His recent research interests include the area of signal processing and machine learning.

Georgios Kostoulas received the Diploma and the M.Sc. degree from the School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece, in 2015 and 2016, respectively. He is currently a Research Engineer with the Information Management Systems Institute, ATHENA Research and Innovation Center, Athens, Greece. His research interested include optimization algorithms, machine learning, semantic web, and large-scale geospatial data integration.

Georgios Lourakis received the Diploma degree from the School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece, in 2014, where he is currently working toward the M.Sc. degree. His research interests focuses on the area of machine learning.

Kejun Huang (M'17) received the Bachelor's degree from the Nanjing University of Information Science and Technology, Nanjing, China, in 2010, and the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, MN, USA, in 2016. He is currently a Postdoctoral Associate with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA. His research interests include signal processing, machine learning, and optimization.

Nicholas D. Sidiropoulos (F'09) received the Ph.D. degree in electrical engineering from the University of Maryland College Park, College Park, MD, USA, in 1992.

He was the faculty member with the University of Virginia, University of Minnesota, and the Technical University of Crete, Greece, prior to his current appointment as the Chair of the Electrical and Computer Engineering Department, University of Virginia, Charlottesville, VA, USA. His research interests include signal processing, communications, optimization, tensor decomposition, and factor analysis, with applications in machine learning and communications. He was the recipient of the NSF/CAREER award in 1998, the IEEE Signal Processing Society (SPS) Best Paper Award in 2001, 2007, and 2011, served as the IEEE SPS Distinguished Lecturer (2008–2009), and currently serves as the Vice President—Membership of IEEE SPS. He was the recipient of the 2010 IEEE Signal Processing Society Meritorious Service Award, and the 2013 Distinguished Alumni Award from the Department of ECE, University of Maryland. He is a Fellow of EURASIP (2014).