Structured Data Fusion

Laurent Sorber, Marc Van Barel, *Member, IEEE,* and Lieven De Lathauwer, *Fellow, IEEE*

***Abstract*—We present structured data fusion (SDF) as a framework for the rapid prototyping of knowledge discovery in one or more possibly incomplete data sets. In SDF, each data set—stored as a dense, sparse or incomplete tensor—is factorized with a matrix or tensor decomposition. Factorizations can be coupled, or fused, with each other by indicating which factors should be shared between data sets. At the same time, factors may be imposed to have any type of structure that can be constructed as an explicit function of some underlying variables. With the right choice of decomposition type and factor structure, even well-known matrix factorizations such as the eigenvalue decomposition, singular value decomposition and QR factorization can be computed with SDF. A domain specific language (DSL) for SDF is implemented as part of the software package Tensorlab, with which we offer a library of tensor decompositions and factor structures to choose from. The versatility of the SDF framework is demonstrated by means of four diverse applications, which are all solved entirely within Tensorlab's DSL.**

***Index Terms*—big data, tensor, data fusion, structured matrices, canonical polyadic decomposition, block term decomposition, structured factors, domain specific language**

I. Introduction

SUCCESSOR to the industrial age and fostered by the digital revolution, the information age is characterized by an ever increasing rate of data production and consumption. One of the greatest challenges we face is how to make sense out of the abundance of available information. Although many data sets take the form of matrices, they are increasingly surfacing as multidimensional arrays, also known as higher-order tensors. Due to the curse of dimensionality, aggregating, measuring, computing or storing all entries of high-dimensional tensors is often impracticable. Yet even incomplete tensors—of which only a fraction of entries are known—can require vast amounts of storage. The possibilities for leveraging *big data* continue to evolve rapidly, further fueling a growing demand for new algorithms and software to tap into this wealth of information.

Higher-order tensors also possess properties that are not present on the matrix level. In fact, there is an upcoming trend to (explicitly or implicitly) *tensorize* data sets which are otherwise naturally represented as matrices [1], [2], [3]. Perhaps the most salient property of the jump to the third dimension is that tensor decompositions can be unique under mild conditions without imposing additional constraints [4], [5], [6], [7]. This is in stark contrast to matrix factorizations, which require constraints such as orthogonality or nonnegativity together with sparsity to obtain uniqueness. Factor uniqueness is a desirable property as it is a key facilitator of attaching meaning to factors, or in other words, extracting knowledge from data. However, tensor decompositions and imposing constraints on the factors are not the only means of achieving factor uniqueness.

Integrating and analyzing data from multiple sources, also known as *data fusion*, can be used to develop insights that are deeper and more accurate than those resulting from a single source of data. Similar to tensorization, data fusion provides a powerful means of introducing factor uniqueness. Where a single matrix factorization is not unique without additional constraints on the factors, the joint factorization of two matrices sharing at least one factor can be unique. Intuitively, this can be understood from the fact that if these matrices are of equal size, their joint factorization can be interpreted as a tensor decomposition. As such, tensor decompositions can be seen as a special case of data fusion where the data sets are homogeneous in size. Moreover, data fusion also allows the uniqueness of a single tensor decomposition to be transferred and reinforced between data sets by coupling their factorizations. The latter is area of research that is of special interest to signal processing, cf. the recent papers [8], [9], [10], [11], [12].

Independent component analysis (ICA) [13] is a form of blind source separation (BSS) which embodies the aforementioned concepts: fusion of multiple data sets, exploiting tensorization to obtain unique solutions, and tensor decompositions comprising various types of structured factors. The goal is to separate a matrix of measurement data into two factors called the mixing matrix and source matrix, respectively. To fix the rotational freedom of the two factors, matrix factorizations of the measurement data itself require constraints which may feel forced or unnatural depending on the application. For example, principal component analysis (PCA) is a BSS technique that imposes orthogonality on the factors to obtain uniqueness. In contrast, one variety of ICA instead tensorizes the measurement data into a fourth-order cumulant tensor. The cumulant's factorization in symmetric rank-one tensors is unique under mild conditions and, hence, interpretable. In the case of ICA, the combined effect of tensorization and factorization expresses the desire that the underlying sources should be statistically independent. This can be seen as an implicit constraint which is often strong enough to elicit a unique result.

L. Sorber and M. Van Barel are with NALAG, Department of Computer Science, KU Leuven, Celestijnenlaan 200A, BE-3001 Leuven, Belgium (Laurent.Sorber@cs.kuleuven.be, Marc.VanBarel@cs.kuleuven.be).

L. De Lathauwer is with the Group Science, Engineering and Technology, KU Leuven Kulak, E. Sabbelaan 53, BE-8500 Kortrijk, Belgium (Lieven.DeLathauwer@kuleuven-kulak.be).

L. Sorber and L. De Lathauwer are also with STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics, Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, BE-3001 Leuven, Belgium and iMinds Future Health Department.

Manuscript received July 1, 2014; revised November 8, 2014.

Both fusion by coupled tensor factorization and imposing structure on the corresponding factor matrices have been shown to improve the results of ICA. For example, the cumulant's information can be fused with second-order statistics such as the measurement data's covariance matrix to improve robustness and accuracy [13, Chap. 5]. Convolutive extensions of this type of ICA wherein a block-Hankel structure arises in the factor matrices corresponding to the first three modes of the cumulant and first mode of the covariance matrix have also been proposed [14]. A different approach to ICA identifies the mixing matrix by computing several time-lagged covariance matrices and fusing them with a joint factorization [15], [16]. Furthermore, convolutive extensions of this time-lagged form of ICA can be formulated in terms of joint block diagonalization, which is tantamount to computing a block term decomposition (BTD) wherein the factor matrices display a Toeplitz-block structure [17].

The concept of fusing data by joint factorization has found its way into many other disciplines. Canonical correlation analysis (CCA) is perhaps the first model aiming to capture the commonalities between two data sets [18]. Later, CCA was extended to the analysis of more than two Gramian matrices, which was dubbed multi-set CCA [19], [20], [21]. Data fusion is also known as multi-view [22] and multi-relational [23], [24] learning in artificial intelligence, as multi-set or multi-block data analysis in chemometrics and systems biology [25], [26], [27] and as integrative data analysis in psychometrics [28]. Multi-view data arises when an observation's attributes, or features, consist of two or more disjoint sets, or views. For example, a video's features could be separated into its visual and audio content. Relational data has a natural representation in the form of a tensor. For example, in the relation *student-performs-task*, the two entities *student* and *task* correspond to the tensor's modes or dimensions, while its entries are defined by the relation type *performs* [29]. This relation can then be coupled with the relations *task-requires-skill* and *student-has-skill* to form a multi-relational data set. Singh and Gordon proposed a framework for multi-relational learning called collective matrix factorization [23], [30] (CMF). Later, Acar et al. generalized CMF to coupled matrix and tensor factorization (CMTF) [31] for a Euclidian distance loss function.

Building on these foundations, we present structured data fusion (SDF) as a novel framework for the rapid prototyping of knowledge discovery in one or more possibly incomplete or sparse data sets. Each data set in an SDF problem is represented as a tensor and factorized with a tensor decomposition. The different factorizations are coupled with each other by indicating which (sub)factors should be shared between data sets. Factors may be imposed to have any type of structure that can be constructed as an explicit function of some underlying variables. The scope of this framework reaches far beyond factor analysis alone, encompassing nearly the full breadth of applications resulting from matrix factorizations and tensor decompositions. It subsumes, for example, tasks based on dimensionality reduction such as feature extraction, subspace learning and model order reduction and tasks related to machine learning such as regression, classification, clustering, and imputation of missing data. Examples of applications that fit in the SDF framework and fuse at least two data sets include social network mining [32], (sub)classification of documents [33], [34], [35], images [36] and cancer [37], [38], sentiment analysis [39], cross-social media retrieval [40], collaborative and content-based filtering [41], [23], [42], [43], link prediction [44], [45], fusion of metabolomics data [46], [25], [26], predicting protein-protein interactions [47], [48], [49], multilingual text analysis [50], analysis of batch processes [51], spectral clustering [52], gene expression profiling [53], [54], gene function prediction [24], personalized medicine [55], multi-task learning [56], community discovery [57], compressed sensing [58], and brain computer interfacing [59], [60]. For a more general background in recent tensor methods and applications, cf. [61], [62], [63]. The SDF framework is also directly applicable to a wide range of applications in signal processing [9], [10], [11], [12], [61], [64], [65], [66]. For example, data acquired by widely separated colocated arrays admit coupled tensor decompositions with the common source signals in a shared factor matrix. The terms have rank-one or block structure depending on the propagation (line-of-sight or multipath, respectively) [65], [66]. Moreover, Vandermonde structure is common when employing uniform linear arrays (ULA), in direction of arrival (DOA) applications and in harmonic retrieval [9], [10], [67]. In this paper, we do not aim to solve any one specific problem in signal processing, but rather create a language with which one can rapidly design, implement and solve many of the above problems.

In SDF, the type of tensor decomposition, the coupling between factorizations and the structure imposed on the factors can all be chosen freely without any changes to the solver. By enabling the user to effortlessly switch between the countless combinations arising from these choices, he or she can rapidly iterate towards a solution for the problem at hand. The different types of tensor decompositions and factor structures are completely modularized, which opens the door to building separate libraries of these components. This goal is achieved in part by completely isolating the computational aspects related to the decomposition and factor structure from each other and the SDF solver. Furthermore, SDF is built with big data in mind—attaining linear time complexity in the total number of known elements in the data sets. As a result, the amount of effort required to add new tensor decompositions or factor structures to a software package implementing SDF such as Tensorlab [68] is kept to a minimum. The latter is a MATLAB toolbox that currently offers SDF with three types of tensor decompositions, two types of regularization and a library of 32 factor structures including nonnegativity, orthogonality and many matrix structures such as Toeplitz, Hankel and Vandermonde. With the right choice of decomposition type and factor structure, even well-known matrix factorizations such as the eigenvalue decomposition, singular value decomposition and QR factorization can be computed with SDF.

The paper is organized as follows. Section II introduces SDF framework and reviews the most prominent tensor decompositions. In Section III we propose two families of algorithms for solving SDF problems with a computational complexity per iteration which is linear in the number of known elements of the data sets. Moreover, the resulting

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

algorithms are able to optimize over the complex domain just as easily as over the real domain. In Section IV we apply the SDF framework on four applications: (a) accurately computing eigenvalues, (b) making personalized movie recommendations, (c) designing an alloy to exhibit certain physical properties and (d) predicting user participation in activities. We conclude the paper in Section V.

II. Structured data fusion

Given D data sets $\mathcal{T}^{(d)}$ which are stored as possibly incomplete multidimensional arrays in $\mathbb{C}^{I_1 \times \cdots \times I_{N_d}}$, also called tensors, we are interested in jointly factorizing these data sets in factors which may depend (non)linearly on some underlying variables. The data sets need not be high-dimensional—vector and matrix data sets correspond to N_d equal to 1 and 2, respectively. With each tensor $\mathcal{T}^{(d)}$ we associate a so-called observation tensor $\mathcal{W}^{(d)}$ of the same size with ones in the positions corresponding to known entries of the data set and zeros elsewhere. We will use observation tensors mainly as a notational convenience for blanking out any unknown entries. The underlying variables $\boldsymbol{z}$ are stored as an ordered set of V variables $\{z_1, \ldots, z_V\}$, each defined as a real or complex scalar, vector, matrix or higher-order tensor. An ordered set of F factors $\mathcal{X}(\boldsymbol{z})$ is then defined as $\{x_1(z_{i_1}), \ldots, x_F(z_{i_F})\}$, in which the fth transformation x_f is a smooth[1] function which maps the i_fth variable z_{i_f} to a tensor. For example, x_f could map a sequence of Householder reflectors z_{i_f} to a rectangular matrix with orthonormal columns. These factors will form the building blocks of the data sets' tensor decompositions. Note that a factor $x_f(z_{i_f})$ will sometimes be written as x_f to simplify notation. For each tensor $\mathcal{T}^{(d)}$, we choose a type of decomposition, or model, $\mathcal{M}^{(d)}$ to approximate that tensor with. For the moment, assume the model maps a subset of factors in $\mathcal{X}(\boldsymbol{z})$ to a tensor with dimensions equal to those of the approximated tensor. Examples of such models will be given in the following subsection. Two factorizations can be coupled with each other simply by requiring that their models share at least one factor. We then define structured data fusion as the optimization problem

$$\underset{\boldsymbol{z}}{\text{minimize}} \quad \sum_{d=1}^{D} \frac{\omega_d}{2} \left\| \mathcal{M}^{(d)}(\mathcal{X}(\boldsymbol{z})) - \mathcal{T}^{(d)} \right\|^2_{\mathcal{W}^{(d)}}, \qquad (1)$$

where ω_d are scalar weights and $\|\cdot\|_{\mathcal{W}^{(d)}}$ is defined as the Frobenius norm $\|\mathcal{W}^{(d)} * \cdot\|_{\mathrm{F}}$, in which $*$ is the Hadamard or element-wise product. A good rule of thumb for choosing the weights ω_d may be to ensure they are inversely proportional to the number of known elements of the corresponding data sets $\mathcal{T}^{(d)}$ [27].

The motivation for this choice of norm is threefold. First, it is well-known that minimizing a sum of squares as in (1) gives rise to the maximum likelihood estimate of $\boldsymbol{z}$ in the presence of white Gaussian noise on the data sets $\mathcal{T}^{(d)}$. Second, the Frobenius norm is unique in that it enables a cheap Newton-like update with a computational complexity that is linear in the number of known elements, yet still retaining quadratic convergence near the solution. This is an especially important feature when dealing with big data, given that it takes about 8 hours to read a modern 4 TB hard drive from beginning to end. However, a low computational complexity by itself is not enough to be able to process data sets in the tens to hundreds of terabytes range. In this paper, we aim to lay the groundwork for structured data fusion as a language and computational framework, and showcase its possibilities. Mapping the underlying algorithms, or designing an entirely new family of algorithms, to a massively parallel processing platform such as the Hadoop ecosystem is still a topic for future research. There, stochastic gradient descent (SGD) could be a promising candidate for such a family of SDF solvers due to their simplicity and aptitude for large data sets. In addition, substantial progress has recently been made in the computation of tensor decompositions on Hadoop with algorithms based on alternating least squares (ALS) [69], [70], [71]. Third, let $\mathrm{vec}(\cdot)$ denote the column-wise vectorization of its argument and let n be the number of elements in a tensor $\mathcal{T}$. Then the equivalence of norms states that any vector norm can be used to bound another vector norm from above and below. For example, the L1-norm and infinity-norm are bounded by

$$\begin{aligned} \|\mathcal{T}\|_{\mathrm{F}} \leq \|\operatorname{vec}(\mathcal{T})\|_1 &\leq \sqrt{n}\|\mathcal{T}\|_{\mathrm{F}} \\ \frac{1}{\sqrt{n}}\|\mathcal{T}\|_{\mathrm{F}} \leq \|\operatorname{vec}(\mathcal{T})\|_\infty &\leq \|\mathcal{T}\|_{\mathrm{F}}, \end{aligned}$$

respectively. This suggests that even when a different norm is desired, it may be worthwhile to first minimize the computationally more tractable Frobenius norm as this will likely improve other measures of error as well.

A. Tensor decompositions

The rank of a matrix can be defined as the minimal number of rank-one matrices R whose sum $\sum_{r=1}^{R} \boldsymbol{u}_r \cdot \sigma_r \cdot \boldsymbol{v}_r^{\mathrm{T}}$ is equal to that matrix, or equivalently as the dimension of its column and row spaces, represented by the matrices U and V, respectively, in a factorization of the form $U \cdot S \cdot V^{\mathrm{T}}$. Together with some constraints on the factorization's constituents, we arrive at well-known matrix decompositions such as the singular value decomposition, eigenvalue decomposition and QR factorization, among many others.

Although these definitions of rank are compatible with each other in the matrix case, they disjoin into two different concepts for higher-order tensors. On the one hand we have the rank of a tensor, which is defined as the minimal number of so-called rank-one tensors whose sum is equal to that tensor. On the other hand, we have the multilinear rank of a tensor, which is defined as a tuple comprising the dimensions of the mode-n vector spaces of the tensor. In the matrix case, the dimension of the column space is equal to the dimension of the row space, but this is no longer true for higher-order tensors in general.

The two different views of tensor rank engender two archetypal tensor decompositions. The first is the canonical polyadic decomposition (CPD) [72], [73], [74], [75], which

[1]Here, smooth is to be understood in the sense of having continuous Wirtinger derivatives. In other words, the partial derivatives of x_f with respect to the elements of z_{i_f} and their complex conjugates $\overline{z_{i_f}}$ should exist and be continuous up to all orders.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

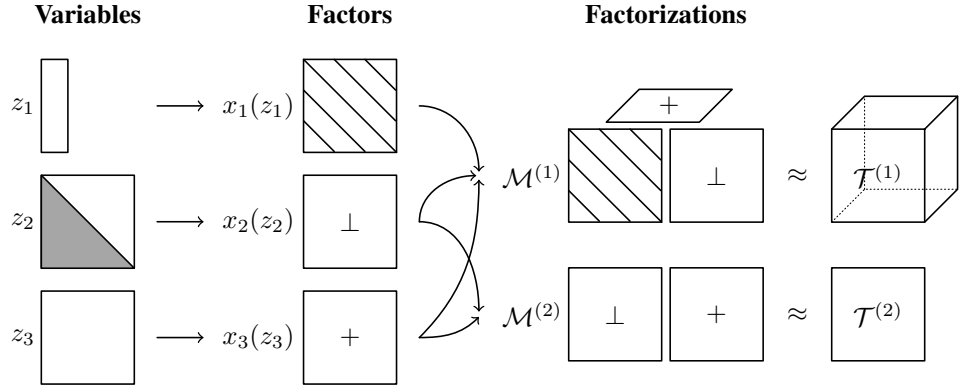

Fig. 1. Schematic of structured data fusion. The vector z_1, upper triangular matrix z_2 (representing a sequence of Householder reflectors) and full matrix z_3 are transformed into a Toeplitz, orthogonal and nonnegative matrix, respectively. The resulting factors are then used to jointly factorize two coupled data sets.

decomposes a tensor as the sum of a minimal[2] number of rank-one tensors (cf. Figure 2). The outer product of two tensors $\mathcal{A} \in \mathbb{C}^{I_1 \times \cdots \times I_P}$ and $\mathcal{B} \in \mathbb{C}^{J_1 \times \cdots \times J_Q}$ is defined by $(\mathcal{A} \circ \mathcal{B})_{i_1 \cdots i_P j_1 \cdots j_Q} := a_{i_1 \cdots i_P} \cdot b_{j_1 \cdots j_Q}$. A rank-one tensor is defined as an outer product of two or more nonzero vectors. For example, the outer product of two nonzero vectors $\boldsymbol{a} \circ \boldsymbol{b}$ is equivalent to the rank-one matrix $\boldsymbol{a} \cdot \boldsymbol{b}^{\mathrm{T}}$. The CPD model can then be written as

$$\mathcal{M}_{\mathrm{CPD}}(U^{(1)}, \ldots, U^{(N)}) := \sum_{r=1}^{R} \boldsymbol{u}_r^{(1)} \circ \cdots \circ \boldsymbol{u}_r^{(N)}, \quad (2)$$

where $\boldsymbol{u}_r^{(n)}$ denotes the rth column of the nth factor matrix $U^{(n)}$. Here, we arrive at a first example of a tensor decomposition which can be applied in an SDF model. Concretely, we could choose $\mathcal{M}^{(d)}(\mathcal{X}(\boldsymbol{z})) := \mathcal{M}_{\mathrm{CPD}}(U^{(1)}, \ldots, U^{(N)})$ in order to approximate the dth data set $\mathcal{T}^{(d)}$ by a CPD. The factor matrices $U^{(n)} := x_n(z_{i_n})$ may be structured by defining x_n to transform some underlying variables z_{i_n} into a matrix of required shape, or they may be unstructured by defining x_n as the identity function.

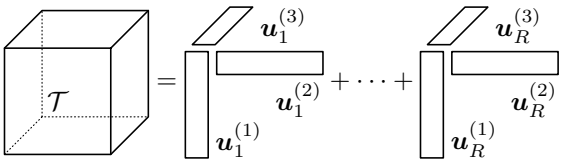

Fig. 2. Canonical polyadic decomposition of a third-order tensor.

The smallest integer R for which a (noise-free) tensor's CPD is exactly equal to that tensor is called the tensor's rank and the corresponding decomposition is called the tensor's rank decomposition.

In addition, the rank decomposition of a higher-order tensor is unique under relatively mild conditions. Clearly, the rank-one terms in (2) may be arbitrarily permuted without affecting their sum. The vectors in a single rank-one term may also be arbitrarily scaled, as long as their product remains the same. A rank decomposition is called (essentially) unique when it is subject only to these permutation and scaling ambiguities. The most well-known sufficient condition for uniqueness is due to Kruskal [4], [5], which is generically satisfied if

$$R \leq \left\lceil \frac{1}{2} \sum_{n=1}^{N} \min(I_n - 1, R - 1) \right\rceil. \quad (3)$$

More recent and powerful frameworks for the uniqueness of tensor decompositions can be found in [6], [7].

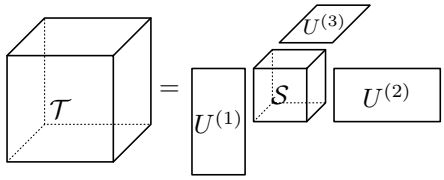

Fig. 3. Low multilinear rank approximation of a third-order tensor.

The low multilinear rank approximation (LMLRA) or Tucker decomposition [76], [62] is related to the second concept of tensor rank. It decomposes a tensor as a so-called core tensor $\mathcal{S}$, multiplied by matrices $U^{(n)}$ along each of its N modes (cf. Figure 3). A tensor's mode-n matricization is a matrix whose columns comprise the tensor's mode-n vectors—i.e., column vectors, row vectors, and so on. Formally, the mode-n matricization $T_{(n)}$ of a tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$ maps tensor element with indices $(i_1, \ldots, i_N)$ to matrix element (i_n, j) such that

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k \quad \text{with} \quad J_k := \prod_{\substack{m=0 \\ m \neq n}}^{k-1} I_m, \quad (4)$$

wherein $I_0 := 1$. The mode-n rank of a tensor is then defined to be the rank of its mode-n matricization and the tensor's multilinear rank is represented by a tuple of all mode-n ranks. Furthermore, the mode-n tensor-matrix product $\mathcal{S} \bullet_n U^{(n)}$ of a tensor $\mathcal{S} \in \mathbb{C}^{R_1 \times \cdots \times R_N}$ and a matrix $U^{(n)} \in \mathbb{C}^{I_n \times R_n}$ is

[2]In practice, most decompositions of higher-order tensors are only approximate. Accordingly, minimal is to be understood in the sense of Occam's razor—i.e., the smallest value for which the model explains the data sufficiently well.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

defined by the matricization $(\mathcal{S} \bullet_n U^{(n)})_{(n)} = U^{(n)} \cdot S_{(n)}$. In other words, in a mode-n tensor-matrix product each mode-n vector of a given tensor is premultiplied by a given matrix. For example, the expression $U \cdot S \cdot V^{\mathrm{T}}$ is equivalent to $S \bullet_1 U \bullet_2 V$. The LMLRA model can now be formulated as

$$\mathcal{M}_{\text{LMLRA}}(U^{(1)}, \ldots, U^{(N)}, \mathcal{S}) := \mathcal{S} \bullet_1 U^{(1)} \cdots \bullet_N U^{(N)}. \quad (5)$$

Similarly to the CPD, the LMLRA is another decomposition that can be applied in the SDF framework. To approximate the dth data set in an SDF model with a LMLRA, we set $\mathcal{M}^{(d)}(\mathcal{X}(\boldsymbol{z})) := \mathcal{M}_{\text{LMLRA}}(U^{(1)}, \ldots, U^{(N)}, \mathcal{S})$. The factors $U^{(n)} := x_n(z_{i_n})$ and $\mathcal{S} := x_{N+1}(z_{i_{N+1}})$ may be structured by defining x_n to transform some underlying variables z_{i_n} into a matrix or core tensor of required shape, or they may be unstructured by defining x_n as the identity function.

Lastly, the block term decomposition (BTD) framework [77], [78], [79] unifies the CPD and LMLRA by combining the CPD's mild conditions for uniqueness with the more general low-rank structure of a LMLRA. A BTD decomposes a tensor as a sum of low multilinear rank terms, each of which may have a different multilinear rank. Therefore, the BTD model is given by

$$\begin{aligned} &\mathcal{M}_{\text{BTD}}(\{U_r^{(n)}\}_{r=1,n=1}^{R,N}, \{\mathcal{S}\}_{r=1}^{R}) \\ &:= \sum_{r=1}^{R} \mathcal{S}_r \bullet_1 U_r^{(1)} \cdots \bullet_N U_r^{(N)}. \end{aligned} \quad (6)$$

To approximate the dth data set in an SDF model with a BTD, we choose $\mathcal{M}^{(d)}(\mathcal{X}(\boldsymbol{z})) := \mathcal{M}_{\text{BTD}}(\{U_r^{(n)}\}_{r=1,n=1}^{R,N}, \{\mathcal{S}\}_{r=1}^{R})$, wherein $U_r^{(n)} := x_{f_{(r,n)}}$ and $\mathcal{S}_r := x_{f_{(r,S)}}$ may depend nonlinearly on some underlying set of variables $\boldsymbol{z}$.

In principle, it would suffice to exclusively implement the BTD model in the SDF framework since it subsumes both the CPD and LMLRA. For instance, the LMLRA can be seen as a single term BTD. However, as we will see there are computational advantages to be found in integrating the CPD model separately. One of the SDF framework's strengths is that it is decomposition agnostic. As such, support for other models such as the tensor train decomposition [80] can also be incorporated without having to change the underlying algorithms.

III. Algorithms for SDF

The two most prevalent classes of algorithms to tackle (1) are quasi-Newton (QN) methods and nonlinear least squares (NLS) methods. Both iteratively improve an initial solution with additive updates obtained by minimizing a second-order approximation of the objective function based solely on first-order derivatives. In the following, let $\mathcal{F}^{(d)}$ and $f^{(d)}$ be the dth residual tensor $\mathcal{M}^{(d)}(\mathcal{X}(\boldsymbol{z})) - \mathcal{T}^{(d)}$ and objective function $1/2 \, \|\mathcal{F}^{(d)}\|^2_{\mathcal{W}^{(d)}}$, respectively. The SDF objective function is then $f := \sum_d \omega_d f^{(d)}$. Additionally, we identify tensors and ordered sets of variables and factors with their vectorized versions where convenient, especially in or around derivatives. For instance, the Jacobian $\partial \mathcal{F}^{(d)} / \partial \mathcal{X}^{\mathrm{T}}$ should be read as $\partial \operatorname{vec}(\mathcal{F}^{(d)}) / \partial \operatorname{vec}(\mathcal{X})^{\mathrm{T}}$, where $\operatorname{vec}(\mathcal{X})$ is the vector $\begin{bmatrix} \operatorname{vec}(x_1)^{\mathrm{T}} & \cdots & \operatorname{vec}(x_F)^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$.

Given an objective function f and a current iterate $\boldsymbol{z}_k$, the key idea of QN and NLS is to minimize a second-order Taylor series approximation of f around $\boldsymbol{z}_k$. Like matrix factorizations, tensor decompositions may also involve complex-valued factors. Under these circumstances, the objective function f is necessarily nonanalytic in $\boldsymbol{z}$ due to the Frobenius norm. In other words, f depends on the complex conjugate $\overline{\boldsymbol{z}}$ and because of this the Taylor series of f around $\boldsymbol{z}$ does not exist in general. The de facto solution is to consider the Taylor series of f around the real and imaginary parts of the variables $\boldsymbol{z}$. However, this easily obfuscates any structure present in the objective function and its derivatives. Recently, a complex optimization framework [81] has been introduced with which QN and NLS algorithms were generalized to both real and complex variables. In complex optimization, the real Taylor series is replaced by a complex Taylor series of f around $\boldsymbol{z}_k$ based on complex derivatives. More specifically, $f(\boldsymbol{z}_k + \boldsymbol{p})$ is approximated by the second-order complex Taylor series expansion

$$m_k^f(\boldsymbol{p}) := f(\boldsymbol{z}_k) + \overset{c}{\boldsymbol{p}}^{\mathrm{T}} \cdot \frac{\partial f}{\partial \overset{c}{\boldsymbol{z}}}(\boldsymbol{z}_k) + \frac{1}{2} \overset{c}{\boldsymbol{p}}^{\mathrm{H}} \cdot B_k \cdot \overset{c}{\boldsymbol{p}}, \quad (7)$$

wherein $\boldsymbol{z}$ and $\boldsymbol{p}$ are implicitly vectorized and $\overset{c}{\cdot}$ denotes the vertical concatenation of its argument and its complex conjugate, e.g., $\overset{c}{\boldsymbol{z}}$ is equivalent to $\begin{bmatrix} \boldsymbol{z}^{\mathrm{T}} & \overline{\boldsymbol{z}}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$. Analogously, the complex gradient $\partial f / \partial \overset{c}{\boldsymbol{z}}$ concatenates the so-called cogradient $\partial f / \partial \boldsymbol{z}$ and conjugate cogradient $\partial f / \partial \overline{\boldsymbol{z}}$. By definition, these derivatives are to be interpreted as partial derivatives with respect to elements of $\boldsymbol{z}$ ($\overline{\boldsymbol{z}}$), while treating $\overline{\boldsymbol{z}}$ ($\boldsymbol{z}$) as constant. They are also known, especially in the German literature, as Wirtinger derivatives [82]. For real-valued functions f, the cogradients are each other's complex conjugates. Consequently, only one of the two cogradients need be computed. In practice, most algorithms ask for the so-called scaled conjugate cogradient $2\partial f / \partial \overline{\boldsymbol{z}}$ for two reasons. First, the negative conjugate cogradient is the objective function's steepest descent direction. Second, the factor two ensures the scaled conjugate cogradient coincides with the real gradient if all variables are real-valued.

By construction, the matrix B_k is a positive semidefinite approximation of the complex Hessian $\partial^2 f / (\partial \overset{c}{\overline{\boldsymbol{z}}} \partial \overset{c}{\boldsymbol{z}}^{\mathrm{T}})$. A good candidate step to take is the approximate Newton step $\boldsymbol{p}_{\text{AN}} := \arg\min_{\boldsymbol{p}} m_k^f(\boldsymbol{p})$, which can be obtained by setting the model's cogradients equal to zero and solving the resulting linear system

$$B_k \cdot \overset{c}{\boldsymbol{p}}_{\text{AN}} = -\frac{\partial f}{\partial \overset{c}{\overline{\boldsymbol{z}}}}(\boldsymbol{z}_k). \quad (8)$$

Depending on the total number of elements in $\boldsymbol{z}$, storing (let alone inverting) B_k can become prohibitively expensive. Fortunately, both QN and NLS methods have their own way of mitigating this cost. Assuming we have (approximately) solved (8), the next iterate can then be computed as $\boldsymbol{z}_{k+1} \leftarrow \boldsymbol{z}_k + \boldsymbol{p}_*$, where $\boldsymbol{p}_*$ is usually a linear combination of the steepest descent direction $\boldsymbol{p}_{\text{SD}} := -2\partial f / \partial \overline{\boldsymbol{z}}$ at $\boldsymbol{z}_k$ and the approximate Newton step $\boldsymbol{p}_{\text{AN}}$. The former is used to guarantee that the objective function value will improve and the latter enables super-linear or even quadratic convergence near the solution.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

Algorithms that compute $\boldsymbol{p}_*$ given the steepest descent direction and approximate Newton step are called globalization strategies and can be categorized into line search and trust-region approaches.

A. Quasi-Newton

Limited memory BFGS (L-BFGS) and, by extension, nonlinear conjugate gradient (NCG) are examples of QN methods [83]. Instead of storing B_k, L-BFGS represents its inverse B_k^{-1} as the sum of a scaled identity matrix plus m rank-two updates. The inverse Hessian approximation at the next iterate B_{k+1}^{-1} is computed as a rank-two correction of B_k^{-1}. This BFGS update is designed so that B_{k+1}^{-1} is, among all matrices that satisfy the so-called secant equation, in some sense as close as possible to B_k^{-1}. NCG can be seen as a special case of L-BFGS where $m = 1$ together with a few other assumptions. In practice, it has been observed that L-BFGS often performs better than NCG, which is likely due to its greater flexibility in representing second-order information.

Not only does a diagonal plus low-rank matrix representation of B_k^{-1} save memory, it also allows for a very efficient solution to (8). In fact, the L-BFGS two-loop recursion algorithm generalized to complex optimization [81] can compute $\boldsymbol{p}_{\text{AN}}$ directly with the same amount of (albeit complex) floating point operations as the equivalent algorithm in real optimization.

In summary, QN methods require only two pieces of information: a way to evaluate the objective function f and its scaled conjugate cogradient $2\partial f/\partial \overline{\boldsymbol{z}}$.

B. Nonlinear least squares

The SDF problem given by (1) consists of a weighted sum of NLS objective functions $f^{(d)}$. NLS methods build a second-order model of the form (7) for each such $f^{(d)}$ by approximating the corresponding residual tensors $\mathcal{F}^{(d)}$ around $\boldsymbol{z}_k$ with the linear model

$$m_k^{\mathcal{F}^{(d)}}(\boldsymbol{p}) := \mathcal{F}^{(d)}(\boldsymbol{z}_k) + J_k^{(d)} \cdot \overset{c}{\boldsymbol{p}}, \tag{9}$$

wherein $J_k^{(d)} := \partial \mathcal{F}^{(d)}/\partial \overset{c}{\boldsymbol{z}}^{\mathrm{T}}$ is the residual tensor's complex Jacobian at $\boldsymbol{z}_k$. Substituting this model for $\mathcal{F}^{(d)}$ in the SDF objective function results in a complex second-order Taylor series expansion of f [81], [84] with

$$B_k = \sum_{d=1}^{D} \omega_d \left(\operatorname{vec}(\mathcal{W}^{(d)}) * J_k^{(d)}\right)^{\mathrm{H}} \cdot \left(\operatorname{vec}(\mathcal{W}^{(d)}) * J_k^{(d)}\right), \tag{10}$$

where the Hadamard product implicitly expands singleton dimensions of its operands. In other words, dimensions of length one of an operand that do not match the size of the corresponding dimensions in the other operand are repeated until their sizes match. Notice that the complex Jacobian $J_k^{(d)}$ does not depend on the data in $\mathcal{T}^{(d)}$. However, the approximate Hessian B_k now does depend on the locations of the known entries of all data sets in the problem (stored in $\mathcal{W}^{(d)}$).

Depending on the model $\mathcal{M}^{(d)}$, a relatively sparse correction matrix can be added to the Gauss–Newton approximation (10) to obtain the objective function's complex Hessian [84]. Coupled with the fact that such corrections are scaled by residuals in $\mathcal{F}^{(d)}$, it may be expected that the Gauss–Newton approximation rapidly approaches the complex Hessian as the residuals decrease in magnitude. Consequently, NLS methods can converge close to quadratically near solutions with small residuals.

For many model parameterizations the Gauss–Newton approximation of the Hessian is rank deficient due to indeterminacies such as the scaling ambiguity in the CPD [84]. To solve (8) under these circumstances, one approach is to compute the Moore–Penrose pseudo-inverse of B_k. Another is to regularize the solution by using the Levenberg–Marquardt approximation of the Hessian, which adds a scaled identity matrix to the Gauss–Newton approximation.

However, tensor decompositions' Jacobians are also often very structured and we wish to exploit this structure to save memory and reduce the computational complexity of computing $\boldsymbol{p}_{\text{AN}}$. Inexact NLS methods only approximately solve (8) with the preconditioned conjugate gradient (PCG) algorithm [83]. Since PCG only requires a function to evaluate $B_k \cdot \boldsymbol{y}$ for different $\boldsymbol{y}$, we can store B_k in a data-sparse way and easily exploit its rank structure in its matrix-vector product. Moreover, singular B_k no longer pose a problem because the solution can be regularized by limiting the number of PCG iterations. The convergence rate of PCG depends on how well the system's eigenvalues are clustered and the magnitude of its condition number. The eigenvalue distribution can be improved by applying a so-called preconditioner, which can be thought of as a cheap and approximate inverse of B_k. Designing preconditioners is a difficult task which is usually done one a case-by-case basis. We will focus on preconditioners for a single unstructured CPD or BTD, and leave the design of more advanced preconditioners as a topic for future work.

In summary, inexact NLS methods require four pieces of information: a way to evaluate the objective function f, its scaled conjugate cogradient $2\partial f/\partial \overline{\boldsymbol{z}}$, the approximate Hessian-vector product $B_k \cdot \boldsymbol{y}$ and, optionally, a preconditioner.

C. Modularizing decompositions and structure

Now that we have settled on which components need to be implemented, we will further subdivide these components into parts so as to isolate the decomposition models from any structure imposed on the factors. Modularizing decompositions and factor structure has two major advantages. First, it enables us to build libraries of both models and factor structures to be mixed and matched as desired. Second, separating the model and factor structure from each other ensures both remain transparent and independently exploitable.

Objective function. By definition, $f(\boldsymbol{z}_k)$ is equal to the weighted sum $\sum_d \omega_d f^{(d)}(\boldsymbol{z}_k)$. To compute the latter, simply evaluate $\mathcal{X}(\boldsymbol{z}_k)$ once and then use the resulting factors to evaluate the models $\mathcal{M}^{(d)}$ at the known entries of the corresponding data sets $\mathcal{T}^{(d)}$.

Gradient. Analogously to the objective function, we may compute the scaled conjugate cogradient $2\partial f/\partial \overline{\boldsymbol{z}}$ as $\sum_d \omega_d (2\,\partial f^{(d)}/\partial \overline{\boldsymbol{z}})$. To decouple the factor structure from the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

model, we apply the chain rule [81]

$$\frac{\partial f^{(d)}}{\partial \overline{\boldsymbol{z}}} = \left(\frac{\partial \mathcal{X}}{\partial \boldsymbol{z}^{\mathrm{T}}}\right)^{\mathrm{H}} \cdot \frac{\partial f^{(d)}}{\partial \overline{\mathcal{X}}} + \left(\frac{\partial \mathcal{X}}{\partial \overline{\boldsymbol{z}}^{\mathrm{T}}}\right)^{\mathrm{T}} \cdot \frac{\partial f^{(d)}}{\partial \mathcal{X}} \tag{11}$$

to separate the conjugate cogradient $\partial f^{(d)}/\partial \overline{\boldsymbol{z}}$ into partial derivative of the objective function $f^{(d)}$ with respect to the factors $\mathcal{X}$ and a partial derivative of the factors $\mathcal{X}$ with respect to the variables $\boldsymbol{z}$. As mentioned at the beginning of this section, we transparently interchange tensors and ordered sets with their vectorized equivalents in and around derivatives. Since $f^{(d)}$ is real-valued, its cogradient is the complex conjugate of its conjugate cogradient so that we can implement (11) by first computing an ordered set of factors $\mathcal{H}$ as $\partial f^{(d)}/\partial \overline{\mathcal{X}}$ and then applying the map

$$h_f \mapsto \left(\frac{\partial x_f}{\partial z_{i_f}^{\mathrm{T}}}\right)^{\mathrm{H}} \cdot h_f + \overline{\left(\frac{\partial x_f}{\partial \overline{z_{i_f}}^{\mathrm{T}}}\right)^{\mathrm{H}} \cdot h_f} \tag{12}$$

to each of its constituents h_f. In other words, the components of the conjugate cogradient $\partial f^{(d)}/\partial \overline{\boldsymbol{z}}$ can be obtained by computing the partial derivative of the objective function $f^{(d)}$ with respect to the complex conjugate of the unstructured factors x_f, vectorizing the result h_f and then applying the linear contraction (12). Consequently, imposing a factor structure defined by the transformation x_f not only requires the ability to evaluate the expansion $x_f(z_{i_f})$, but also its associated linear contraction given by (12). Hence, to implement a factor structure, we simply need to implement these two components. For instance, a factorization may depend on a factor which is the matrix inverse of another factor. In that case, $x_f(z_{i_f}) := z_{i_f}^{-1}$ for certain indices f and i_f and it can be shown that the associated linear contraction is given by

$$h_f \mapsto -z_{i_f}^{-\mathrm{H}} \cdot h_f \cdot z_{i_f}^{-\mathrm{H}}. \tag{13}$$

When the model $\mathcal{M}^{(d)}$ is $\mathcal{M}_{\text{CPD}}$ or $\mathcal{M}_{\text{BTD}}$, the partial derivative of the objective function $f^{(d)}$ with respect to the factors $\mathcal{X}$ depends on the string of Khatri–Rao and Kronecker products

$$\mathrm{kr}_\sigma(\alpha_m) := \bigodot_{\substack{m=0 \\ N-m \notin \sigma}}^{N-1} \overline{\alpha_{N-m}} \quad \text{and} \tag{14a}$$

$$\mathrm{kron}_\sigma(\alpha_m) := \bigotimes_{\substack{m=0 \\ N-m \notin \sigma}}^{N-1} \overline{\alpha_{N-m}}, \tag{14b}$$

respectively, in which N is implicitly defined as the length of the sequence α_m, which usually corresponds to an ordered set of factors such as $\{x_{f_m}\}_{m=1}^{N}$. The Khatri–Rao product $A \odot B$ of two matrices A and B with an equal number of columns R is defined as the column-wise Kronecker product $\begin{bmatrix}\boldsymbol{a}_1 \otimes \boldsymbol{b}_1 & \cdots & \boldsymbol{a}_R \otimes \boldsymbol{b}_R\end{bmatrix}$. In short, the functions kr_σ and kron_σ compute a string of Khatri–Rao or Kronecker products of the sequence α_m from back to front, with the exception of the indices specified by the set σ.

Assuming the dth data set $\mathcal{T}^{(d)}$ is modeled by $\mathcal{M}_{\text{CPD}}$ defined by the factor matrices x_{f_n}, $n = 1, \ldots, N_d$, the partial derivative with respect to the complex conjugate of the nth factor matrix is given by [84]

$$2\frac{\partial f^{(d)}}{\partial \overline{x_{f_n}}} = (W_{(n)}^{(d)} * F_{(n)}^{(d)}) \cdot \mathrm{kr}_{\{n\}}(x_{f_m}). \tag{15}$$

Similarly, if the dth data set $\mathcal{T}^{(d)}$ is modeled by $\mathcal{M}_{\text{BTD}}$ defined by the factor matrices $x_{f_{(r,n)}}$ and core tensors $x_{f_{(r,S)}}$, $r = 1, \ldots, R_d$ and $n = 1, \ldots, N_d$, the partial derivative with respect to the complex conjugate of the nth factor matrix and core tensor of the rth term are given by

$$2\frac{\partial f^{(d)}}{\partial \overline{x_{f_{(r,n)}}}} = (W_{(n)}^{(d)} * F_{(n)}^{(d)}) \cdot \mathrm{kron}_{\{n\}}(x_{f_{(r,m)}}) \cdot (x_{f_{(r,S)}})_{(n)}^{\mathrm{H}} \tag{16a}$$

and

$$2\frac{\partial f^{(d)}}{\partial \overline{x_{f_{(r,S)}}}} = \mathrm{vec}(\mathcal{W}^{(d)} * \mathcal{F}^{(d)})^{\mathrm{T}} \cdot \mathrm{kron}_{\varnothing}(x_{f_{(r,m)}}), \tag{16b}$$

respectively. In both (15) and (16), the Hadamard product of the observation tensor $\mathcal{W}^{(d)}$ and residual tensor $\mathcal{F}^{(d)}$ ensures that only the residuals corresponding to known entries of the data set contribute to the gradient.

The matrices $\mathrm{kr}_{\{n\}}(x_{f_m})$ and $\mathrm{kron}_{\varnothing}(x_{f_{(r,m)}})$ are highly structured and often appear as the right operand in matricized tensor-matrix products, such as in (15) and (16). Concretely, they represent the matrices $\overline{x_{f_{N_d}}} \odot \cdots \odot \overline{x_{f_{n+1}}} \odot \overline{x_{f_{n-1}}} \odot \cdots \odot \overline{x_{f_1}}$ and $\overline{x_{f_{(r,N_d)}}} \otimes \cdots \otimes \overline{x_{f_{(r,1)}}}$, respectively. It is desirable to avoid explicitly forming these matrices where possible. Efficient algorithms computing the matricized-tensor times string of Khatri–Rao products are available [85], [86], and can be generalized to strings of Kronecker products with little effort. Although these algorithms obviate the need to permute the tensor's elements in memory or explicitly form the Khatri-Rao or Kronecker products, they are ill-suited for big data because of their sizable intermediate results.

We now have the components necessary to solve SDF problems using gradient-based algorithms. Each model $\mathcal{M}^{(d)}$ should define how it can be evaluated given a set of factors and how to compute the gradient of its corresponding objective function $f^{(d)}$ with respect to those factors. Additionally, each transformation x_f should define how to expand the variable z_{i_f} into the factor $x_f(z_{i_f})$ and how to apply the associated linear contraction (12). The final two components—the approximate Hessian-vector product $B_k \cdot \boldsymbol{y}$ and an optional preconditioner—are all that remain to enable the NLS family of algorithms as well.

Approximate Hessian-vector product. Although the approximate Hessian B_k in (10) does not depend on the tensor's known entries, it does depend on their locations through $\mathcal{W}^{(d)}$. In the case of big data, even reading the observation tensor $\mathcal{W}^{(d)}$ could take an unreasonable amount of time. Moreover, the presence of the observation tensor destroys the rank structure which would otherwise be present in the approximate Hessian. For these reasons, we redefine the approximate Hessian as

$$B_k := \sum_{d=1}^{D} \omega_d \, j_k^{(d)\mathrm{H}} j_k^{(d)}, \tag{17}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

where the Jacobian $\dot{J}_k^{(d)}$ is defined as $\partial\mathcal{F}^{(d)}/\partial\boldsymbol{z}^{\mathrm{T}}$ at $\boldsymbol{z}_k$. For the sake of simplicity, we have assumed that $\mathcal{F}^{(d)}$ is analytic in $\boldsymbol{z}$ so that (8) reduces to

$$B_k \cdot \boldsymbol{p}_{\mathrm{AN}} = -2\frac{\partial f}{\partial \overline{\boldsymbol{z}}}(\boldsymbol{z}_k). \tag{18}$$

Because the observation tensor $\mathcal{W}^{(d)}$ was omitted, the approximate Hessian's (Frobenius) norm is now likely to be larger than that of the approximate Hessian in (10). As a result, the approximate Newton step $\boldsymbol{p}_{\mathrm{AN}}$ which solves (18) may be much smaller than intended. A reasonably effective yet simple solution is to instead solve the system

$$B_k \cdot \boldsymbol{p}_{\mathrm{AN}} = -\sum_{d=1}^{D} \sigma_d \omega_d \cdot 2\frac{\partial f^{(d)}}{\partial \overline{\boldsymbol{z}}}(\boldsymbol{z}_k), \tag{19}$$

where σ_d is defined as the product of dth tensor's dimensions, divided by its number of known elements. The function of the scalars σ_d is to make up for the missing residuals corresponding to the unknown entries in the data sets. If each tensor's known entries are distributed more or less randomly across the tensor and if the residuals $\mathcal{F}^{(d)}$ are close to equal in magnitude, it may be expected that the right-hand side of (19) approaches the scaled conjugate cogradient of the same SDF problem wherein all tensors' entries are known.

Again, we use the chain rule to decouple the factor structure defined by the transformations x_f from the models $\mathcal{M}^{(d)}$. By the assumption that $\mathcal{F}^{(d)}$ is analytic in $\boldsymbol{z}$ we obtain the decomposition

$$B_k = \left(\sum_{d=1}^{D} \omega_d \left(\frac{\partial \mathcal{X}}{\partial \boldsymbol{z}^{\mathrm{T}}}\right)^{\mathrm{H}} \cdot \ddot{J}_k^{(d)\mathrm{H}} \ddot{J}_k^{(d)}\right) \cdot \frac{\partial \mathcal{X}}{\partial \boldsymbol{z}^{\mathrm{T}}}, \tag{20}$$

where the so-called (Jacobian's) Gramian $\ddot{J}_k^{(d)\mathrm{H}} \ddot{J}_k^{(d)}$ is defined as $(\partial\mathcal{F}^{(d)}/\partial\mathcal{X}^{\mathrm{T}})^{\mathrm{H}} \cdot \partial\mathcal{F}^{(d)}/\partial\mathcal{X}^{\mathrm{T}}$ at $\boldsymbol{z}_k$. Notice that we discarded the matrix multiplication symbol in the Gramian, emphasizing that it should be thought of as a structured matrix that can be multiplied by a vector in one action instead of two. The approximate Hessian-vector product $B_k \cdot \boldsymbol{y}$ can now be computed in three stages. First, for each transformation x_f expand the corresponding variable y_{i_f} in the ordered set of variables $\boldsymbol{y}$ into a factor g_f with the map

$$y_{i_f} \mapsto \frac{\partial x_f}{\partial z_{i_f}^{\mathrm{T}}} \cdot y_{i_f}. \tag{21}$$

Then, multiply each residual function's Gramian by the resulting ordered set of factors $\mathcal{G}$. Since the Gramian is square, the result can again be interpreted as an ordered set of factors $\mathcal{H}^{(d)}$. Finally, contract $\mathcal{H}^{(d)}$ into variables by reusing the previously defined map (12), multiply by ω_d and sum up the contributions over all data sets. Returning back to our example transformation $x_f(z_{i_f}) := z_{i_f}^{-1}$ which computes the fth factor as the matrix inverse of another factor, the linear expansion (21) can be implemented as

$$y_f \mapsto -z_{i_f}^{-1} \cdot y_f \cdot z_{i_f}^{-1}. \tag{22}$$

Depending on the model, the Gramian can exhibit a certain rank structure that we would like to exploit in its matrix-vector product. To this end, we introduce the string of Hadamard products

$$\mathrm{hdm}_\sigma(\alpha_m) := \underset{\substack{m=0 \\ N-m\notin\sigma}}{\overset{N-1}{*}} \overline{\alpha_{N-m}}. \tag{23}$$

There is an interesting relationship between the Khatri–Rao and Hadamard product given by

$$\mathrm{kr}_\sigma(\alpha_m)^{\mathrm{H}} \cdot \mathrm{kr}_\sigma(\alpha_m) \equiv \mathrm{hdm}_\sigma(\alpha_m^{\mathrm{H}} \cdot \alpha_m), \tag{24}$$

which is a straightforward consequence of the fact that the columns of a string of Khatri–Rao products are vectorized rank-one tensors. Furthermore, we introduce the modified string of Hadamard and Kronecker products

$$\mathrm{hdm}_{\sigma,\rho}(\alpha_m, \beta_m) := \underset{\substack{m=0 \\ N-m\notin\sigma}}{\overset{N-1}{*}} \begin{cases} \overline{\alpha_{N-m}} & N-m \notin \rho \\ \overline{\beta_{N-m}} & \text{otherwise} \end{cases} \tag{25a}$$

$$\mathrm{kron}_{\sigma,\rho}(\alpha_m, \beta_m) := \underset{\substack{m=0 \\ N-m\notin\sigma}}{\overset{N-1}{\otimes}} \begin{cases} \overline{\alpha_{N-m}} & N-m \notin \rho \\ \overline{\beta_{N-m}} & \text{otherwise} \end{cases}, \tag{25b}$$

respectively. These functions generalize hdm_σ and kron_σ so that operands in the string corresponding to indices in the set ρ can be replaced by those of a second sequence β_m.

Assuming the dth data set $\mathcal{T}^{(d)}$ is modeled by $\mathcal{M}_{\mathrm{CPD}}$ defined by the factor matrices x_{f_n}, $n = 1, \ldots, N_d$, the factor matrix with index f_n of the Gramian-vector product is given by [84]

$$\begin{aligned}(\ddot{J}_k^{(d)\mathrm{H}} \ddot{J}_k^{(d)} \cdot \mathcal{G})_{f_n} &= g_{f_n} \cdot \mathrm{hdm}_{\{n\}}(x_{f_m}^{\mathrm{H}} \cdot x_{f_m}) \\ &\quad + x_{f_n} \cdot \sum_{\substack{\tilde{n}=1 \\ \tilde{n}\neq n}}^{N_d} \mathrm{hdm}_{\{n\},\{\tilde{n}\}}(x_{f_m}^{\mathrm{H}} \cdot x_{f_m}, g_{f_m}^{\mathrm{H}} \cdot x_{f_m}).\end{aligned} \tag{26}$$

Herein, g_{f_n} is the factor matrix with index f_n in the ordered set of factors $\mathcal{G}$.

Similarly, if the dth data set $\mathcal{T}^{(d)}$ is modeled by $\mathcal{M}_{\mathrm{BTD}}$ defined by the factor matrices $x_{f_{(r,n)}}$ and core tensors $x_{f_{(r,S)}}$, $r = 1, \ldots, R_d$ and $n = 1, \ldots, N_d$, it can be shown that the factor matrix with index $f_{(r,n)}$ and core tensor with index $f_{(r,S)}$ of the Gramian-vector product are given by

$$\begin{aligned}(\ddot{J}_k^{(d)\mathrm{H}} \ddot{J}_k^{(d)} \cdot \mathcal{G})_{f_{(r,n)}} &= \sum_{\tilde{r}=1}^{R_d} \Bigg(x_{f_{(\tilde{r},n)}} \cdot \Bigg((g_{f_{(\tilde{r},S)}})_{(n)} \\ &\quad \cdot \mathrm{kron}_{\{n\}}(x_{f_{(\tilde{r},m)}}^{\mathrm{H}} \cdot x_{f_{(r,m)}}) + (x_{f_{(\tilde{r},S)}})_{(n)} \\ &\quad \cdot \sum_{\substack{\tilde{n}=1 \\ \tilde{n}\neq n}}^{N_d} \mathrm{kron}_{\{n\},\{\tilde{n}\}}(x_{f_{(\tilde{r},m)}}^{\mathrm{H}} \cdot x_{f_{(r,m)}}, g_{f_{(\tilde{r},m)}}^{\mathrm{H}} \cdot x_{f_{(r,m)}}) \Bigg) \\ &\quad + g_{f_{(\tilde{r},n)}} \cdot (x_{f_{(\tilde{r},S)}})_{(n)} \cdot \mathrm{kron}_{\{n\}}(x_{f_{(\tilde{r},m)}}^{\mathrm{H}} \cdot x_{f_{(r,m)}}) \Bigg) \\ &\quad \cdot (x_{f_{(r,S)}})_{(n)}^{\mathrm{H}}\end{aligned} \tag{27a}$$

and

$$\begin{aligned}(\ddot{J}_k^{(d)\mathrm{H}} \ddot{J}_k^{(d)} \cdot \mathcal{G})_{f_{(r,S)}} &= \sum_{\tilde{r}=1}^{R_d} \Bigg(\mathrm{vec}(g_{f_{(\tilde{r},S)}})^{\mathrm{T}} \\ &\quad \cdot \mathrm{kron}_{\varnothing}(x_{f_{(\tilde{r},m)}}^{\mathrm{H}} \cdot x_{f_{(r,m)}}) + \mathrm{vec}(x_{f_{(\tilde{r},S)}})^{\mathrm{T}} \\ &\quad \cdot \sum_{\tilde{n}=1}^{N_d} \mathrm{kron}_{\varnothing,\{\tilde{n}\}}(x_{f_{(\tilde{r},m)}}^{\mathrm{H}} \cdot x_{f_{(r,m)}}, g_{f_{(\tilde{r},m)}}^{\mathrm{H}} \cdot x_{f_{(r,m)}}) \Bigg),\end{aligned} \tag{27b}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

respectively. The Gramian-vector products of (26) and (27) are designed to fully exploit the Gramian's rank structure. For example, in the case of a CPD the Gramian consists exclusively of diagonal and rank-one blocks [84]. Instead of computing and storing these blocks explicitly, (26) directly applies their action on the factors g_f. In much the same way, (27) takes advantage of the rank structure of the block term decomposition's Gramian. As with the computation of the gradient, notice that all of the terms in these expressions can be implemented as matricized-tensor times string of Kronecker products.

Preconditioner. The convergence rate of computing the approximate Newton step with PCG can be improved by modifying the system (19) with a preconditioner M_k which resembles B_k, yet is cheap to invert [83]. Designing preconditioners for SDF problems is complicated by the fact that B_k is a weighted sum of Gramians transformed by linear expansion and contraction operators. In contrast to the previous components such as the gradient and approximate Hessian-vector product, there is no straightforward way of building a preconditioner for each of the countless combinations of decomposition types, coupling and factor structure. Instead, we restrict ourselves to the approximation of a single data set with a CPD or BTD in which there is no structure or symmetry imposed on the factors.

In the case of a single data set PCG computes the approximate Newton step from the modified system

$$M_k^{-1} \cdot B_k \cdot \boldsymbol{p}_{\text{AN}} = -\sigma M_k^{-1} \cdot 2\frac{\partial f}{\partial \overline{\boldsymbol{z}}}(\boldsymbol{z}_k), \tag{28}$$

where M_k is a symmetric positive definite matrix and σ is the scaling coefficient introduced in (19).

Since the Gramian exhibits a block structure for many tensor decompositions, a block-diagonal preconditioner is a natural first choice. These so-called block-Jacobi type preconditioners can be interpreted as a divide-and-conquer approach to inverting the approximate Hessian as each of the blocks may be inverted independently of each other. Not only are the diagonal blocks much smaller, and hence cheaper to invert, but they may also exhibit some structure which can be exploited in computing their inverse.

When the data set is approximated by a CPD defined by the factor matrices x_{f_n}, $n = 1, \ldots, N$, the factor matrix with index f_n of the block-Jacobi preconditioner-vector product $M_k^{-1} \cdot \boldsymbol{y}$ can be computed as [84]

$$(M_k^{-1} \cdot \mathcal{Y})_{f_n} = y_{f_n} \cdot \text{hdm}_{\{n\}}^{-1}(x_{f_m}^{\text{H}} \cdot x_{f_m}). \tag{29}$$

Likewise, when the data set is approximated by a BTD defined by the factor matrices $x_{f_{(r,n)}}$ and core tensors $x_{f_{(r,S)}}$, $r = 1, \ldots, R_d$ and $n = 1, \ldots, N_d$, the factor matrix with index $f_{(r,n)}$ and core tensor with index $f_{(r,S)}$ of the block-Jacobi preconditioner-vector product are given by [84]

$$\begin{aligned}(M_k^{-1} \cdot \mathcal{Y})_{f_{(r,n)}} = y_{f_{(r,n)}} \cdot \Big(&(x_{f_{(r,S)}})_{(n)} \\ &\cdot \text{kron}_{\{n\}}(x_{f_{(r,m)}}^{\text{H}} \cdot x_{f_{(r,m)}}) \cdot (x_{f_{(r,S)}})_{(n)}^{\text{H}} \Big)^{-1}\end{aligned} \tag{30a}$$

and

$$\begin{aligned}(M_k^{-1} \cdot \mathcal{Y})_{f_{(r,S)}} = &\text{vec}(y_{f_{(r,S)}})^{\text{T}} \\ &\cdot \text{kron}_{\varnothing}((x_{f_{(r,m)}}^{\text{H}} \cdot x_{f_{(r,m)}})^{-1}),\end{aligned} \tag{30b}$$

respectively.

D. Generalizations

The SDF framework as presented can be improved even further with a number of generalizations. One such improvement would be to allow chaining of transformations, i.e., allow factors of the form $x_g(x_f(z_{i_f}))$. This would enable combining factor structures in novel ways. For example, a nonnegative Toeplitz structure could be imposed by using x_f to square the elements of a generator vector z_{i_f} and then applying x_g to generate the Toeplitz matrix. Other convenient improvements could include allowing constant factors, which can be used to incorporate prior knowledge, and nested variables, which are variables that themselves consist of an ordered set of variables. The latter is especially useful when a single variable is most naturally represented by a set of vectors, matrices or tensors. For example, a factor matrix with orthonormal columns $x_f(\boldsymbol{z}_f)$ can be parametrized by a sequence of Householder reflectors, which could be stored as a set of vectors of increasing length $\{z_{f_k}\}_{k=0}^{K} =: \boldsymbol{z}_f$.

With Tensorlab [68], we offer a domain specific language (DSL) for defining SDF problems which includes the above and other improvements such as the ability to compose factors out of subfactors. At the time of writing the supported models include the CPD, LMLRA and BTD and a library of 32 factor structures is provided. With the latter, the user can impose constraints such as nonnegativity, orthonormal columns, columns as sums of kernel functions and banded, Toeplitz, Hankel or Vandermonde structures, among others. As an example of the modularity of the framework, two additional models representing an L1- and L2-regularization term in the objective function are offered. Efficient complex QN and NLS algorithms are included for solving the resulting SDF problems, as well as tools for estimating model parameters and generating high-quality initializations.

IV. Numerical experiments

A. Eigenvalue decomposition as a structured tensor decomposition

To demonstrate the flexibility of SDF, we compare the relative accuracy of the eigenvalues of a certain matrix as computed by LAPACK's driver routine DGEEV with those computed by the Tensorlab implementation of SDF. Consider the colleague matrix

$$A = \begin{bmatrix} 0 & 1/2 & & \\ 1 & 0 & 1/2 & \\ & 1/2 & 0 & \ddots \\ & & \ddots & \ddots \end{bmatrix}$$

of order n, corresponding to the Chebyshev polynomial $T_n(x) := \cos(n \arccos(x))$. The eigenvalues of the matrix A

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

are the zeros of $T_n(x)$, which are known to be given in closed form by

$$\lambda_i = \cos\left(\frac{\pi(i+\frac{1}{2})}{n}\right), \qquad i = 0, \ldots, n-1. \tag{31}$$

For reference, we evaluate (31) in high precision and then truncate the result to double precision. To compute the eigenvalue decomposition of A with SDF, we define two variables z_1 and z_2 as a matrix of order n and a row vector of length n, respectively. Then, we define three transformations $x_1(z_1) := z_1$, $x_2(z_1) := z_1^{-\mathrm{T}}$ and $x_3(z_2) := z_2$. Finally, we model the eigenvalue decomposition of A as a CPD defined by the factor matrices x_1, x_2 and x_3. Here, we interpret A as an $n \times n \times 1$ tensor wherein x_3 is associated with its third dimension. We compute the eigenvalue decomposition of A with LAPACK and use the resulting eigenvectors and eigenvalues to initialize the variables z_1 and z_2. In Tensorlab, we choose the `sdf_nls` nonlinear least squares solver and set the convergence criteria `TolX` and `TolFun` (related to the relative difference between iterates and objective function, respectively) to 0, ensuring that the solver stops when the maximum accuracy has been reached. The relative accuracy of the eigenvalues compared to the reference solution is shown in Figure 4 for $n = 200$. About 60% of the eigenvalues computed by SDF have a relative error which is at least 10 times smaller than those of LAPACK, and this increases to over 75% for $n = 500$.

```
% Define and initialize variables.
model.variables.z1 = randn(n,n);
model.variables.z2 = randn(1,n);

% Define factors as transformed variables.
model.factors.x1 = 'z1';
model.factors.x2 = {'z1',@struct_invtransp};
model.factors.x3 = 'z2';

% Define factorizations using the factors.
model.factorizations.evd.data = A;
model.factorizations.evd.cpd  = {'x1','x2','x3'};

% Solve the SDF problem.
sol = sdf_nls(model); % sol.variables, sol.factors
```

Listing 1. Defining and solving an eigenvalue decomposition with Tensorlab's DSL for SDF.

The strength of SDF is not so much the efficiency with which the eigenvalue decomposition is computed, but rather the expressive power of the framework and, in this example, the improved accuracy of the resulting eigenvalues. To make this more concrete, Listing 1 gives an impression of how an eigenvalue decomposition could be implemented using Tensorlab's DSL. The SDF problem is created as a MATLAB structure that defines the model's variables, factors and factorizations. The function `@struct_invtransp` encodes the transformation x_2 and its associated left and right Jacobian-vector products (12) and (21), wherein it exploits all structure related to the transformation. Structure at the tensor decomposition level is exploited by their gradient and approximate Hessian-vector products. After solving the model with `sdf_minf` or `sdf_nls`—corresponding to the quasi-Newton and NLS family of algorithms discussed in the previous section—the resulting MATLAB structure `sol` contains the optimized variables and factors in `sol.variables` and `sol.factors`, respectively.

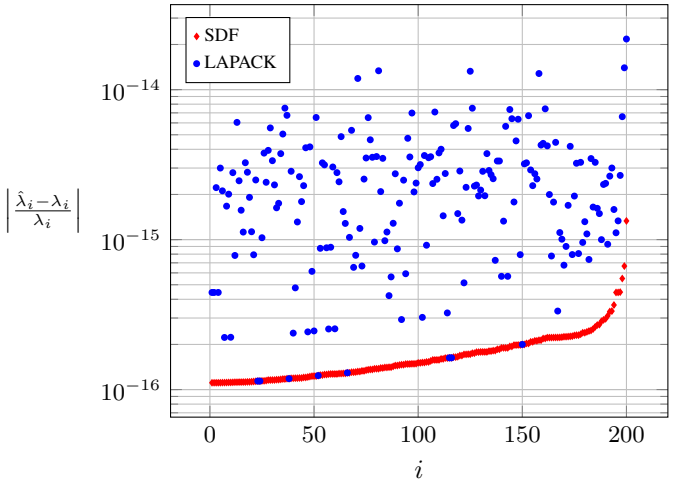

Fig. 4. Relative accuracy of the eigenvalues of the colleague matrix A, as computed by LAPACK and Tensorlab's implementation of SDF.

B. Recommending movies: the Netflix prize

In 2006, the online DVD rental company Netflix organized a contest to improve its recommender system. The company released a training set of more than 100 million ratings given by about 480,000 customers to almost 18,000 movies over a period of 6 years. The goal of the contest is to predict unseen ratings more accurately than Netflix's proprietary Cinematch algorithm, enabling better personalized movie recommendations. The Netflix data set can be described by an incomplete user $\times$ movie $\times$ date tensor $\mathcal{R}$ containing integer ratings between 1 and 5 stars given by users to movies on certain dates. The data set fits in about 2 GB of memory, which is not considered big data, but still large enough to represent a formidable computational challenge. A standardized validation set of about 2 million ratings, called the probe set, was also supplied by Netflix.

Matrix factorization techniques were a key component in most solutions, though it took some time before participants understood how to effectively incorporate temporal data into their models. With a tensor representation of the data set, the temporal data is naturally taken into account by employing tensor factorizations. Here, we use SDF to extend the core matrix factorization model of the prize winning entry [87] to a tensor factorization. We will model each rating r_{umd} as

$$r_{umd} \approx \mu + b_u^{(1)} + b_m^{(2)} + b_d^{(3)} + \sum_{k=1}^{K} z_{uk}^{(1)} z_{mk}^{(2)} z_{dk}^{(3)}, \tag{32}$$

where μ is the mean rating and the vectors $\boldsymbol{b}^{(1)}$, $\boldsymbol{b}^{(2)}$ and $\boldsymbol{b}^{(3)}$ represent a bias due to users, movies and temporal effects, respectively. For example, the user bias models the fact that some users consistently rate movies a certain amount of stars

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

TABLE I
RMSE ON THE PROBE SET BY THE MEAN μ, NETFLIX'S CINEMATCH ALGORITHM AND MODEL (32).

Model	Mean	Cinematch	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	$K=6$	Winning model
RMSE	1.1296	0.9474	0.9447	0.9387	0.9372	0.9326	0.9298	0.9275	≈0.8527

more or less than the average user. Taking a closer look at (32), we notice that the bias terms are in fact structured rank-one tensors in which the outer product is taken with all-one vectors. Accordingly, after removing the mean, these terms can be assimilated together with the matrices $Z^{(1)}$, $Z^{(2)}$, $Z^{(3)}$ into factor matrices $x_1 = \begin{bmatrix}\boldsymbol{b}^{(1)} & 1_{U\times 2} & Z^{(1)}\end{bmatrix}$, $x_2 = \begin{bmatrix}1_{M\times 1} & \boldsymbol{b}^{(2)} & 1_{M\times 1} & Z^{(2)}\end{bmatrix}$ and $x_3 = \begin{bmatrix}1_{D\times 2} & \boldsymbol{b}^{(3)} & Z^{(3)}\end{bmatrix}$ corresponding to a structured CPD.

We factorize the tensor $\mathcal{R}$ using SDF to implement the model (32) as a structured CPD and add a regularization term to the objective function of the form $\lambda(\|x_1\|_{\mathrm{F}}^2+\|x_2\|_{\mathrm{F}}^2+\|x_3\|_{\mathrm{F}}^2)$ to prevent overfitting. To determine the hyperparameter λ, we perform a simple grid search and select the value for which the model best explains the ratings in the validation set. The resulting SDF problem was a strenuous challenge for MATLAB, requiring a high-end server with several tens of gigabytes of memory to solve due to the size of the data set. Even so, a single gradient call took about 30 seconds to evaluate, which implies a solution time of about 1 hour per model assuming convergence is reached after 120 iterations.

Netflix chose to measure the predictive power of its models with the root mean square error (RMSE), defined as the square root of the mean squared error on the individual ratings. This measure has the tendency to amplify the penalization of false positives ("trust busters") and false negatives ("missed opportunities"). Table IV-B shows the RMSE on the probe set of a baseline model that predicts each rating as the mean rating μ, Netflix's Cinematch algorithm and model (32) for various values of the rank parameter K. Here, we compute each model using the `sdf_minf` L-BFGS algorithm with the maximum number of L-BFGS updates set to 20, and `TolX` and `TolFun` set to 10^{-4} and 10^{-8}, respectively[3]. Because of the significant computational effort required, we have limited our experiments to a maximum of $K = 6$ rank-one terms. Even a relatively simple bias plus rank-one model can improve on Cinematch's performance, while larger values of K provide consistent improvements to the RMSE. The Netflix prize was awarded to the first team to improve on Cinematch's performance by 10%, which corresponds to an RMSE of 0.8527 on the probe set. Table IV-B shows that a bias plus rank-6 model already accounts for a 2.1% improvement over Cinematch. For comparison, the prize winning entry consisted of a blend of 107 different models, the majority of which are matrix factorization models with ranks in the order of hundreds to thousands [87].

[3]Note that the SDF objective function (1) is a sum of squares, implying that a relative change of 10^{-8} in the objective function corresponds to a relative change of 10^{-4} after applying the square root.

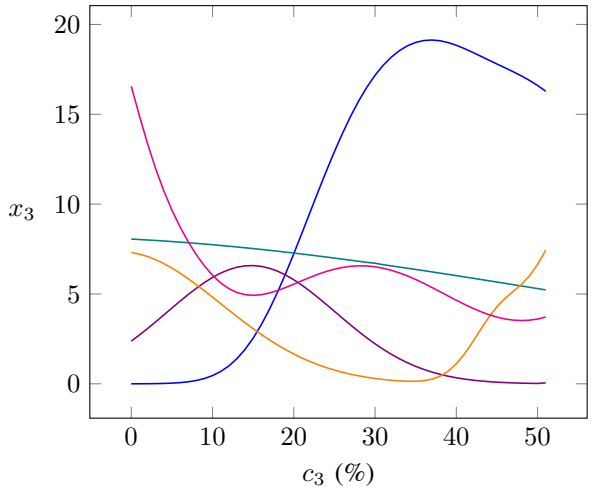

Fig. 5. The $R = 5$ columns of the third factor matrix x_3 in the structured CPD of the InsPyro data set. Each column is a sum of $D = 8$ RBF kernels, sampled at different concentrations c_3. Here, the columns are also evaluated at non-integer concentrations.

C. Designing an alloy

By varying the concentrations of an alloy's constituent metals, materials with different physical properties can be designed. For example, we might be interested in the tensile strength of an alloy comprising four metals. Since the concentrations should sum to one, the concentration of the fourth metal is fixed once the concentrations of the first three are chosen. Assuming we discretize the concentrations of each of the constituent metals between 0% and 99% in steps of 1%, we can arrange the tensile strength of the resulting alloys in a $100 \times 100 \times 100$ tensor $\mathcal{T}$ where the nth dimension corresponds to the concentration of the nth constituent. Since measuring or computing each of the one million combinations is impractical, we would like to sample only a few of the tensor's entries. Based on these samples, the tensile strength of the other combinations can then be inferred.

InsPyro NV—a KU Leuven spin-off—kindly provided us with a data set containing more than 35,000 measurements of a certain material property of a four-material mixture. Although we cannot describe the material property or the material's constituents in more detail, we will refer to former as the tensile strength and to the latter as metals for the sake of this example. The metals' concentrations are varied in steps of 1% starting at 11% for the first metal and 1% for the remaining metals, up to a maximum of 95%, 49% and 52% for the three metals, respectively. As such, the data set can be represented as an incomplete $85 \times 49 \times 52$ tensor $\mathcal{T}$ in which 16.31% of the entries are known.

One fact we wish to exploit in modelling this data set is that the tensile strength is expected to vary relatively smoothly as

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

the concentrations are varied. We can take advantage of this piece of information by approximating the tensor with a CPD in which the factor matrices' columns are sampled smoothly varying functions. Let $z^{(n)} \in \mathbb{R}^{3\times D\times R}$, $n = 1,\ldots,3$, represent the parameters of R functions, each of which is the sum of D radial basis function (RBF) kernels. The ith element of the rth column of the nth factor matrix $x_n(z^{(n)})$ is then described by

$$(x_n(z^{(n)}))_{ir} := \sum_{d=1}^{D} z_{1dr}^{(n)} \exp\left(-\frac{(c_n(i) - z_{2dr}^{(n)})^2}{2(z_{3dr}^{(n)})^2}\right), \quad (33)$$

where $c_n(i)$ maps the index i to the ith concentration of the nth constituent, and the first, second and third row of $z^{(n)}$ correspond to the amplitude, location and spread of the individual RBF kernels.

We choose $D = 8$ and $R = 5$ and implement the structured CPD with factor matrices x_1, x_2 and x_3 defined by (33) with Tensorlab's DSL. We choose the `sdf_minf` L-BFGS algorithm and set `TolX` and `TolFun` set to 0, but limit the maximum number of iterations `MaxIter` to 10^4. The model is trained on two thirds of the data, and validated on the remaining one third. The relative error of the optimized model was 0.27% for both training and validation set.

Figure 5 plots the columns of the third factor matrix x_3 as a function of the concentration c_3, and illustrates that by modelling the columns of the factor matrices as continuous functions, we have the added benefit of being able to evaluate the melting point at non-integer concentrations. Figure 6 shows a slice of the modelled tensor, where the concentration of the first constituent c_1 is kept fixed at 20%. The remaining two concentrations c_2 and c_3 are free. Because the sample data is not uniformly distributed over the tensor but is rather clustered per slice, we have coloured the slice only partially opaque to indicate where the model is supported by nearby data points. Straying too far from the samples may lead to inaccurate extrapolations of the tensile strength even though the validation error is small. Computing a simple rank-5 CPD of the tensor with alternating least squares (ALS) results in a comparable error on both training and validation set, although at the cost of obtaining factors that are significantly less smooth and losing the ability to evaluate the factors at noninteger concentrations. The resulting irregular factors suggest that the solution computed by ALS will generalize less well to data points that lie far from the sampled region.

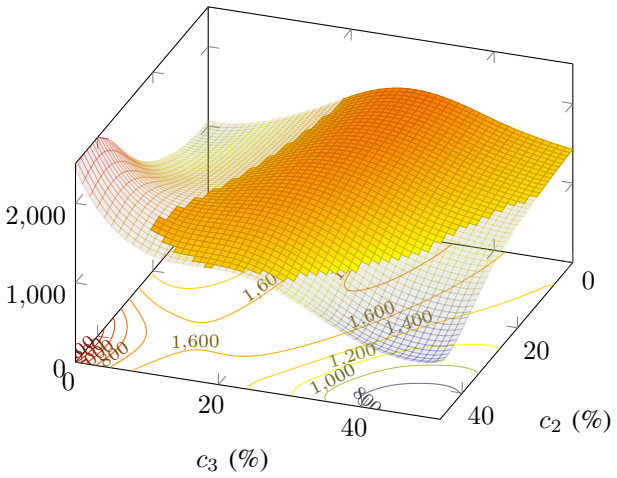

Fig. 6. The tensile strength of an alloy comprising four metals where c_1 is fixed to 20%.

D. Predicting user participation in activities

When visiting a city for the first time, it would be useful to be presented with a personalized list of recommended activities or points of interest (POI) near your current location. In [45], a multi-relational data set is presented in which 164 users' global positioning system (GPS) coordinates were tracked over a period of 2.5 years. The data set consists of 12,765 GPS trajectories with a total length of close to 140 megametres. The raw coordinates were clustered into 168 meaningful locations and the user comments attached to the GPS data were manually parsed into activity annotations for these locations. A total of five activity categories were defined: food and drink, shopping, movies and shows, sports and exercise, and tourism and amusement.

A binary user $\times$ location $\times$ activity tensor $\mathcal{T}^{(1)}$ is constructed from the above data in which element $t_{ula}^{(1)}$ has the value 1 if the user u participated in the activity a at location l, and 0 otherwise. The data set also includes four auxiliary sources of information: a user $\times$ user matrix $\mathcal{T}^{(2)}$ that measures the similarity between users obtained from social network data, a location $\times$ feature matrix $\mathcal{T}^{(3)}$ in which each feature represents a normalized number of POI at a location, an activity $\times$ activity matrix $\mathcal{T}^{(4)}$ that measures the similarity between activities obtained by analysis of internet search engine results, and a binary user $\times$ location matrix $\mathcal{T}^{(5)}$ in which element $t_{ul}^{(5)}$ has the value 1 if the user u has visited location l, and 0 otherwise. We preprocess the data by removing the 18 users who have not participated in any activity and normalize the columns of $\mathcal{T}^{(3)}$ so that they sum to 1. In summary, we have 146 users, 168 locations, 5 activities and 14 location features.

In similar vein to the experiments in [44], we consider two scenarios. In a first scenario, we randomly select 80% of the entries in $\mathcal{T}^{(1)}$ and remove them. In a second scenario, all information (i.e., all slices) of 50 randomly selected users is removed from $\mathcal{T}^{(1)}$. The task is then to predict the missing entries with the remaining information. We model each data set $\mathcal{T}^{(d)}$ with a simple rank-2 CPD $\mathcal{M}^{(d)}$, defined by the user factor $U \in \mathbb{R}^{146\times 2}$, location factor $L \in \mathbb{R}^{168\times 2}$, activity factor $A \in \mathbb{R}^{5\times 2}$ and feature factor $F \in \mathbb{R}^{14\times 2}$. Furthermore, we introduce the factors $\lambda, \mu, \nu \in \mathbb{R}^{1\times 2}$ to absorb any differences in scale between factorizations. For example, we model the user $\times$ location matrix $\mathcal{T}^{(5)}$ by the CPD $\mathcal{M}^{(5)}(U, L, \nu) := U \cdot \mathrm{diag}(\nu) \cdot L^{\mathrm{T}}$. For the first scenario, two different models

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

are computed by solving the regularized SDF problem

$$\begin{aligned}\underset{U,L,A,F,\lambda,\mu,\nu}{\text{minimize}} \quad & \frac{\omega_1}{2}\left\|\mathcal{M}^{(1)}(U,L,A)-\mathcal{T}^{(1)}\right\|^2_{\mathcal{W}^{(1)}} \\ &+\frac{\omega_2}{2}\left\|\mathcal{M}^{(2)}(U,U,\lambda)-\mathcal{T}^{(2)}\right\|^2_{\mathrm{F}}+\frac{\omega_3}{2}\left\|\mathcal{M}^{(3)}(L,F)-\mathcal{T}^{(3)}\right\|^2_{\mathrm{F}} \\ &+\frac{\omega_4}{2}\left\|\mathcal{M}^{(4)}(A,A,\mu)-\mathcal{T}^{(4)}\right\|^2_{\mathrm{F}}+\frac{\omega_5}{2}\left\|\mathcal{M}^{(5)}(U,L,\nu)-\mathcal{T}^{(5)}\right\|^2_{\mathrm{F}} \\ &+\frac{\omega_6}{2}\left(\|U\|^2_{\mathrm{F}}+\|L\|^2_{\mathrm{F}}+\|A\|^2_{\mathrm{F}}+\|F\|^2_{\mathrm{F}}+\|\lambda\|^2_{\mathrm{F}}+\|\mu\|^2_{\mathrm{F}}+\|\nu\|^2_{\mathrm{F}}\right)\end{aligned} \tag{34}$$

for two choices of the weights ω_d. In the first case, all weights are nonzero and we exploit all available information. More specifically, we set $\omega_1 := 1/N_1$, $\omega_2 := 0.05/N_2$, $\omega_3 := 0.01/N_3$, $\omega_4 := 0.001/N_4$, $\omega_5 := 0.001/N_5$ and $\omega_6 := 0.05/N_6$, wherein N_d is the number of residuals (or known elements) of the dth term in the objective function (34). In the second case we choose $\omega_1 := 1$ and $\omega_2 := \omega_3 := \omega_4 := \omega_5 := 0$, corresponding to a regularized CPD of $\mathcal{T}^{(1)}$ without data fusion. We solve (34) for the two choices of weights with the `sdf_nls` nonlinear least squares family of algorithms, and set both `TolX` and `TolFun` to 0 for maximal accuracy.

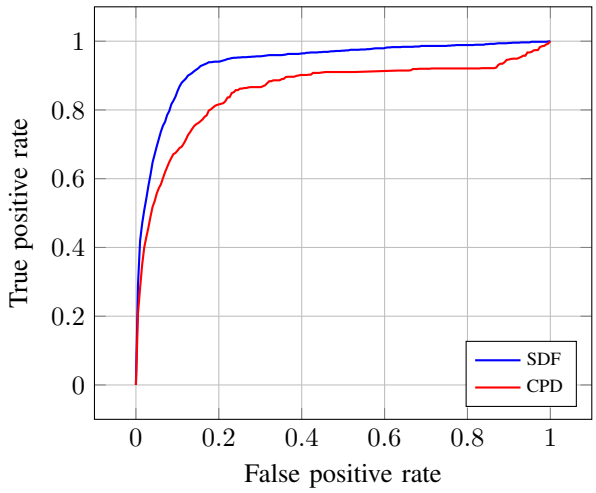

Fig. 7. ROC curves for the prediction of the 80% missing user-location-activity links in $\mathcal{T}^{(1)}$. The SDF model fuses all five data sources with coupled rank-2 CPDs and has an AUC of 93.78%. The CPD model is a regularized CPD of $\mathcal{T}^{(1)}$ and has an AUC of 85.59%.

Figure 7 shows the receiver operating characteristic (ROC) curves of the resulting models. Each ROC curve is obtained by evaluating the corresponding model at the unknown entries of $\mathcal{T}^{(1)}$ and varying a threshold with which we discriminate between the negative class 0 and positive class 1. As the threshold value increases, the number of both false and true positives resulting from model's predictions decreases monotonically. The area under curve (AUC) is a standard measure of a binary classifier's quality. The SDF model, which exploits all available information, has an AUC of 93.78% and clearly outperforms the regularized CPD model, which only exploits information available in $\mathcal{T}^{(1)}$ and has an AUC of 85.59%.

The second scenario corresponds to the difficult case of making predictions for new users, also known as the cold start problem. In this scenario, we must make use of the auxiliary data sets since $\mathcal{T}^{(1)}$ by itself no longer has enough information to fully determine the factor U. To improve predictor robustness, we impose the additional constraint that all factors should be nonnegative. This constraint is enforced by modifying the SDF problem (34) so that the transformation $x(z) := z * z$ is applied to each factor, effectively squaring all entries. In Tensorlab, this can be achieved by applying the transformation `struct_nonneg` to all underlying variables (cf. Listing 2).

```
% Define factors as transformed variables.
model.factors.U = {'z1',@struct_nonneg};
model.factors.L = {'z2',@struct_nonneg};
model.factors.A = {'z3',@struct_nonneg};
model.factors.F = {'z4',@struct_nonneg};
```

Listing 2. Defining nonnegative factors as transformed variables with Tensorlab's DSL for SDF.

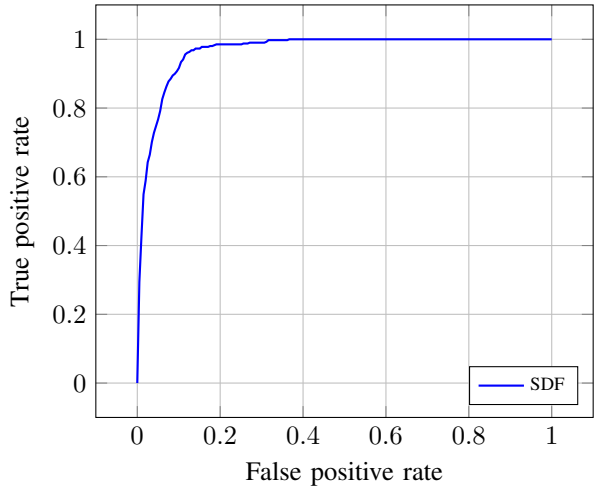

Fig. 8. ROC curve for the prediction of the user-location-activity links of 50 missing users. The SDF model fuses all five data sources with coupled nonnegative rank-2 CPDs and has an AUC of 96.75%.

The ROC curve of the resulting model is shown in Figure 8 and has an AUC of 96.75%. Experimenting with different factor structures and tensor decompositions is as simple as indicating which transformations and factors should be used. For example, changing the model $\mathcal{M}^{(1)}$ from a CPD to a LMLRA only requires adding an extra factor in the SDF problem to represent the LMLRA's core tensor. However, in our experiments the LMLRA did not offer a significant advantage in performance compared to the CPD.

V. Conclusion

We presented structured data fusion (SDF) as a language and computational framework for the rapid prototyping of coupled tensor factorizations with structured factors. The ability to create libraries of both tensor decompositions and factor structures is facilitated by decoupling these components from the underlying optimization algorithm. New decompositions and constraints can be added to these libraries by implementing a small number of functions with which the structure inherent

to each can be fully exploited by means of efficient matrix-vector products. The countless combinations of decomposition types, coupling between factorizations and factor structures and ease of switching between them allows users to rapidly iterate towards a solution. As a special case, well-known matrix factorizations such as the singular value decomposition can be computed by selecting the right decomposition type and factor structures. The SDF framework was implemented as a domain specific language (DSL) in Tensorlab [68], a MATLAB toolbox for tensor computations. The versatility of the framework was demonstrated by solving four applications ranging from the computation of eigenvalue decompositions to activity recommendation entirely within the Tensorlab's DSL for SDF.

ACKNOWLEDGMENTS

Laurent Sorber is supported by a doctoral fellowship of the Flanders agency for Innovation by Science and Technology (IWT).

Marc Van Barel's research is supported by (1) the Research Council KU Leuven: (a) OT/10/038, (b) PF/10/002 Optimization in Engineering (OPTEC), (2) the Research Foundation Flanders (FWO): G.0828.14N, and by (3) the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office.

Lieven De Lathauwer's research is supported by: (1) Research Council KU Leuven: GOA-MaNet, CoE EF/05/006 Optimization in Engineering (OPTEC), (2) F.W.O.: G.0830.14N, G.0881.14N, (3) the Belgian Federal Science Policy Office: IUAP P7 (DYSCO II, Dynamical systems, control and optimization, 2012-2017), (4) EU: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Advanced Grant: BIOTENSORS (no. 339804). This paper reflects only the authors' views and the Union is not liable for any use that may be made of the contained information.

REFERENCES

[1] L. De Lathauwer, "Blind separation of exponential polynomials and the decomposition of a tensor in rank-$(L_r, L_r, 1)$ terms," *SIAM J. Matrix Anal. Appl.*, vol. 32, no. 4, pp. 1451–1474, Dec. 2011.

[2] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, "Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis," *IEEE Signal Proc. Mag.*, vol. 31, no. 5, pp. 71–79, Sep. 2014.

[3] O. Debals, M. Van Barel, and L. De Lathauwer, "Blind signal separation of rational functions using Löwner-based tensorization," Department of Electrical Engineering (ESAT), KU Leuven, ESAT-STADIUS Internal Report 14-167, 2014.

[4] J. B. Kruskal, "Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics," *Linear Algebra Appl.*, vol. 18, no. 2, pp. 95–138, 1977. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0024379577900696

[5] ——, *Rank, decomposition, and uniqueness for 3-way and N-way arrays*. Elsevier Science Publishers B.V., 1989, pp. 7–18.

[6] I. Domanov and L. De Lathauwer, "On the uniqueness of the canonical polyadic decomposition — Part I: Basic results and uniqueness of one factor matrix," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 3, pp. 855–875, 2013.

[7] ——, "On the uniqueness of the canonical polyadic decomposition — Part II: Overall uniqueness," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 3, pp. 876–903, 2013.

[8] M. Sorensen and L. De Lathauwer, "Coupled canonical polyadic decomposition — part I: Uniqueness," Department of Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium, ESAT-STADIUS Internal Report 13-143, 2013.

[9] ——, "Multidimensional harmonic retrieval via coupled canonical polyadic decompositions," Department of Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium, ESAT-STADIUS Internal Report 13-240, 2013.

[10] ——, "Coupled tensor decompositions in array processing," Department of Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium, ESAT-STADIUS Internal Report 13-241, 2013.

[11] ——, "Double coupled CPD with applications," Department of Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium, ESAT-STADIUS Internal Report 13-242, 2013.

[12] I. Domanov and L. De Lathauwer, "Canonical polyadic decomposition of third-order tensors: reduction to generalized eigenvalue decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 35, no. 2, pp. 636–660, Apr. 2014.

[13] P. Comon and C. Jutten, *Handbook of Blind Source Separation: Independent Component Analysis and Applications*. Academic Press, 2010.

[14] C. E. Fernandes, G. Favier, and J. C. Mota, "PARAFAC-based blind identification of convolutive MIMO linear systems," vol. 15, no. 1, pp. 1704–1709, 2009.

[15] A. Belouchrani, K. Abed-Meraim, J.-F. Cardoso, and E. Moulines, "A blind source separation technique using second order statistics," *IEEE Trans. Sig. Proc.*, vol. 45, no. 2, pp. 434–444, 1997.

[16] L. De Lathauwer and J. Castaing, "Blind identification of underdetermined mixtures by simultaneous matrix diagonalization," *IEEE Trans. Sig. Proc.*, vol. 56, no. 3, pp. 1096–1105, 2008.

[17] H. Bousbia-Salah, A. Belouchrani, and K. Abed-Meraim, "Jacobi-like algorithm for blind signal separation of convolutive mixtures," *Electr. Lett.*, vol. 37, no. 16, pp. 1049–1050, 2001.

[18] H. Hotelling, "Relations between two sets of variates," *Biom.*, vol. 28, no. 3/4, pp. 321–377, 1936.

[19] J. Levin, "Simultaneous factor analysis of several Gramian matrices," *Psychometrika*, vol. 31, no. 3, pp. 413–419, 1966.

[20] N. M. Correa, T. Adali, Y.-O. Li, and V. D. Calhoun, "Canonical correlation analysis for data fusion and group inferences," *IEEE Signal Proc. Mag.*, vol. 27, no. 4, pp. 39–50, 2010.

[21] Y.-O. Li, T. Adali, W. Wei, and V. D. Calhoun, "Joint blind source separation by multiset canonical correlation analysis," *IEEE Signal Proc. Mag.*, vol. 57, no. 10, pp. 3918–3929, 2009.

[22] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proc. of the Conference on Computational Learning Theory*, 1998.

[23] A. P. Singh and G. J. Gordon, "Relational learning via collective matrix factorization," in *Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[24] C. Lippert, S. H. Weber, Y. Huang, V. Tresp, M. Schubert, and H.-P. Kriegel, "Relation prediction in multi-relational domains using matrix factorization," in *Proc. of the NIPS 2008 Workshop: Structured Input-Structured Output*, 2008.

[25] I. Van Mechelen and A. K. Smilde, "A generic linked-mode decomposition model for data fusion," *Chemometr. Intell. Lab. Syst.*, vol. 104, no. 1, pp. 83–94, 2010.

[26] A. K. Smilde, M. J. van der Werf, S. Bijlsma, B. J. C. van der Werff-van der Vat, and R. H. Jellema, "Fusion of mass spectrometry-based metabolomics data," *Anal. Chem.*, vol. 77, no. 20, pp. 6729–6736, 2005.

[27] T. Wilderjans, E. Ceulemans, and I. Van Mechelen, "Simultaneous analysis of coupled data blocks differing in size: A comparison of two weighting schemes," *Comp. Stat. Data An.*, vol. 53, no. 4, pp. 1086–1098, 2009.

[28] P. J. Curran and A. M. Hussong, "Integrative data analysis: The simultaneous analysis of multiple data sets," *Psychol. Methods.*, vol. 14, no. 2, pp. 81–100, 2009.

[29] N. Thai-Nghe, L. Drumond, T. Horváth, and L. Schmidt-Thieme, "Multi-relational factorization models for predicting student performance," in *Proc. of the KDD Workshop on Knowledge Discovery in Educational Data*, 2011.

[30] A. P. Singh and G. J. Gordon, "A unified view of matrix factorization models," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5212, pp. 358–373.

[31] E. Acar, T. G. Kolda, and D. M. Dunlavy, "All-at-once optimization for coupled matrix and tensor factorizations," in *Proc. of Mining and Learning with Graphs*, 2011.

[32] E. E. Papalexakis, T. M. Mitchell, N. D. Sidiropoulos, C. Faloutsos, P. P. Talukdar, and B. Murphy, "Scoup-SMT: Scalable coupled sparse matrix-tensor factorization," *Comp. Research Repos.*, vol. abs/1302.7, 2013.

[33] K. Yu, S. Yu, and V. Tresp, "Multi-label informed latent semantic indexing," in *Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2005.

[34] S. Zhu, K. Yu, Y. Chi, and Y. Gong, "Combining content and link for classification using matrix factorization," in *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[35] W.-J. Li and D.-Y. Yeung, "Relation regularized matrix factorization," in *Proc. of the 21st International Joint Conference on Artificial Intelligence*, 2009.

[36] Y. Zhu, Y. Chen, Z. Lu, S. J. Pan, G.-R. Xue, Y. Yu, and Q. Yang, "Heterogeneous transfer learning for image classification," in *Proc. of the 25th AAAI Conference on Artificial Intelligence*, 2011.

[37] W. Liu, J. Chan, J. Bailey, C. Leckie, and K. Ramamohanarao, "Mining labelled tensors by discovering both their common and discriminative subspaces," in *Proc. of the 2013 SIAM International Conference on Data Mining (SDM)*, 2013.

[38] L. Badea, "Multi-relational factorizations for cancer subclassification," in *Proc. of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, 2010.

[39] T. Li, Y. Zhang, and V. Sindhwani, "A non-negative matrix tri-factorization approach to sentiment classification with lexical prior knowledge," in *Proc. of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 2009.

[40] S. K. Gupta, D. Phung, B. Adams, and S. Venkatesh, "A matrix factorization framework for jointly analyzing multiple nonnegative data," in *Proc. of the 2011 SIAM International Conference on Data Mining (SDM)*, 2011.

[41] J. Yoo and S. Choi, "Weighted nonnegative matrix co-tri-factorization for collaborative prediction," in *Advances in Machine Learning*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5828, pp. 396–411.

[42] Y. Fang and L. Si, "Matrix co-factorization for recommendation with rich side information and implicit feedback," in *Proc. of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, 2011.

[43] A. Krohn-Grimberghe, L. Drumond, C. Freudenthaler, and L. Schmidt-Thieme, "Multi-relational matrix factorization using bayesian personalized ranking for social network data," in *Proc. of the 5th ACM international conference on Web search and data mining*, 2012.

[44] B. Ermiş, E. Acar, and A. T. Cemgil, "Link prediction in heterogeneous data via generalized coupled tensor factorization," *Data Min. Knowl. Disc*, pp. 1–34, 2013.

[45] V. W. Zheng, B. Cao, Y. Zheng, X. Xie, and Q. Yang, "Collaborative filtering meets mobile recommendation: A user-centered approach," in *Proc. of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010.

[46] E. Acar, G. Gurdeniz, M. A. Rasmussen, D. Rago, L. O. Dragsted, and R. Bro, "Coupled matrix factorization with sparse factors to identify potential biomarkers in metabolomics," in *Proc. of the 12th IEEE International Conference on Data Mining Workshops (ICDMW)*, 2012.

[47] Q. Xu and Q. Yang, "A survey of transfer and multitask learning in bioinformatics," *J. of Comp. Sc. and Eng.*, vol. 5, no. 3, pp. 257–268, 2011.

[48] Q. Xu, H. Xiang, and Q. Yang, "Protein-protein interaction prediction via collective matrix factorization," in *Proc. of the 2010 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2010.

[49] K. Zhang, J. W. Gray, and B. Parvin, "Sparse multitask regression for identifying common mechanism of response to therapeutic targets," *Bioinformatics*, vol. 26, no. 12, pp. 97–105, 2010.

[50] J. Kim, R. Monteiro, D. C, and H. Park, "Group sparsity in nonnegative matrix factorization," in *Proc. of the 2012 SIAM International Conference on Data Mining (SDM)*, 2012.

[51] A. K. Smilde, J. A. Westerhuis, and R. Boqué, "Multiway multiblock component and covariates regression models," *J. Chem.*, vol. 14, no. 3, pp. 301–331, 2000.

[52] B. Long, Z. Zhang, X. Wú, and P. S. Yu, "Spectral clustering for multi-type relational data," in *Proc. of the 23rd International Conference on Machine Learning*, 2006.

[53] S. P. Ponnapalli, M. A. Saunders, C. F. Van Loan, and O. Alter, "A higher-order generalized singular value decomposition for comparison of global mRNA expression from multiple organisms." *PLoS One*, vol. 6, no. 11, 2011, article e28072.

[54] L. Badea, "Extracting gene expression profiles common to colon and pancreatic adenocarcinoma using simultaneous nonnegative matrix factorization," in *Pacific Symposium on Biocomputing*, vol. 290, no. 13, 2008, pp. 279–290.

[55] C. H. Lee, B. O. Alpert, P. Sankaranarayanan, and O. Alter, "GSVD comparison of patient-matched normal and tumor aCGH profiles reveals global copy-number alterations predicting glioblastoma multiforme survival," *PLoS One*, vol. 7, no. 1, 2012, article e30098.

[56] A. Argyriou, T. Evgeniou, and M. Pontil, "Convex multi-task feature learning," *Mach. Learn.*, vol. 73, no. 3, pp. 243–272, 2008.

[57] Y.-R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher, "Metafac: Community discovery via relational hypergraph factorization," in *Proc. of the 15th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, 2009.

[58] J. Yoo and S. Choi, "Matrix co-factorization on compressed sensing," in *Proc. of the Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[59] H. Lee, J. Yoo, and S. Choi, "Semi-supervised nonnegative matrix factorization," *IEEE Sig. Proc. Letters*, vol. 17, no. 1, pp. 4–7, 2010.

[60] H. Lee and S. Choi, "Group nonnegative matrix factorization for EEG classification," in *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[61] A. Cichocki, D. Mandic, C. Caiafa, A. H. Phan, G. Zhou, Q. Zhao, and L. De Lathauwer, "Multiway component analysis: Tensor decompositions for signal processing applications," *IEEE Signal Proc. Mag.*, Mar. 2015, to appear.

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

[63] E. Acar and B. Yener, "Unsupervised multiway data analysis: A literature survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 21, no. 1, pp. 6–20, 2009.

[64] M. Sorensen, I. Domanov, D. Nion, and L. De Lathauwer, "Coupled canonical polyadic decompositions and (coupled) decompositions in multilinear rank-$(L_r, n, L_r, n, 1)$ terms — part II: Algorithms," Department of Electrical Engineering (ESAT), KU Leuven, ESAT-STADIUS Internal Report 13-144, 2013.

[65] M. Sorensen and L. De Lathauwer, "Multidimensional ESPRIT: A coupled canonical polyadic decomposition approach," in *Proc. of the Eighth IEEE Sensor Array and Multichannel Signal Processing Workshop*, 2014.

[66] ——, "Coupled tensor decompositions for applications in array signal processing," in *Proc. of the 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 2013.

[67] ——, "Multiple invariance ESPRIT: A coupled matrix-tensor factorization approach," Department of Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium, ESAT-STADIUS Internal Report 14-183, 2014.

[68] L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab v2.0," Available online at http://www.tensorlab.net, Jan. 2014.

[69] N. D. Sidiropoulos, E. E. Papalexakis, and C. Faloutsos, "Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition," *IEEE Signal Proc. Mag.*, vol. 31, no. 5, pp. 57–70, Sep. 2014.

[70] A. L. F. de Almeida and A. Y. Kibangou, "Distributed large-scale tensor decomposition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2014*, May 2014, pp. 26–30.

[71] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via the alternating direction method of multipliers," *Comp. Research Repos.*, vol. abs/1409.2, 2014. [Online]. Available: http://arxiv.org/abs/1409.2383

[72] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *J. Math. Phys.*, vol. 6, no. 1, pp. 164–189, 1927.

[73] ——, "Multiple invariants and generalized rank of a p-way matrix or tensor," *J. Math. Phys.*, vol. 7, no. 1, pp. 39–79, 1927.

[74] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, no. 1, pp. 84–84, 1970.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2015.2400415, IEEE Journal of Selected Topics in Signal Processing

[75] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart–Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.
[76] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.
[77] L. De Lathauwer, "Decompositions of a higher-order tensor in block terms — Part I: Lemmas for partitioned matrices," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1022–1032, 2008.
[78] ——, "Decompositions of a higher-order tensor in block terms — Part II: Definitions and uniqueness," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1033–1066, 2008.
[79] L. De Lathauwer and D. Nion, "Decompositions of a higher-order tensor in block terms — Part III: Alternating least squares algorithms," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1067–1083, 2008.
[80] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, Jun. 2011.
[81] L. Sorber, M. Van Barel, and L. De Lathauwer, "Unconstrained optimization of real functions in complex variables," *SIAM J. Optim.*, vol. 22, no. 3, pp. 879–898, 2012.
[82] R. Remmert, *Theory of Complex Functions*. Springer-Verlag, 1991.
[83] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., ser. Springer Series in Operations Research. Springer, 2006.
[84] L. Sorber, M. Van Barel, and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms and a new generalization," *SIAM J. Optim.*, vol. 23, no. 2, pp. 695–720, 2013.
[85] A. H. Phan, P. Tichavský, and A. Cichocki, "Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations," *IEEE Trans. Sig. Proc.*, vol. 61, no. 19, pp. 4834–4846, 2013.
[86] N. Vannieuwenhoven, N. Vanbaelen, K. Meerbergen, and R. Vandebril, "The dense multiple-vector tensor-vector product: An initial study," KU Leuven, Technical Report TW635, 2013.
[87] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

Laurent Sorber was born on August 11, 1987 in Wilrijk, Belgium. In secondary school, he studied mathematics and sciences at the Onze-Lieve-Vrouw-van-Lourdescollege (OLVE), Edegem-Mortsel. He received a Bachelor of Science (B.Sc.) degree in Electrical Engineering and Computer Science from KU Leuven, Leuven in 2008, and a Master of Science (M.Sc.) degree in Mathematical Engineering from the same university in 2010. In May 2015 he obtained a Ph.D. in Engineering, summa cum laude, at the Department of Electrical Engineering and the Department of Computer Science under the supervision of prof. dr. ir. Lieven De Lathauwer and prof. dr. ir. Marc Van Barel, supported by a doctoral fellowship of the Flanders agency for Innovation by Science and Technology (IWT).

Marc Van Barel was born in Mechelen, Belgium on 29 November 1960. He is married and has two children. He received the M.Sc. Degree in Computer Science in 1983 and the Ph.D. Degree in Computer Science (numerical analysis and applied mathematics) in 1989, both from the KU Leuven, Belgium. He is a Full Professor with the Department of Computer Science at this same university since 1998. Currently he is head of NALAG (Numerical Approximation and Linear Algebra Group), one of the three research groups of the Division Numerical Analysis and Applied Mathematics within the Department of Computer Science of the KU Leuven. He is co-editor of 6 special issues of journals devoted to numerical linear algebra. He is also associate editor of "SIAM Journal on Matrix Analysis and Applications", and member of the editorial board of the journals "Linear and Multilinear Algebra", "Numerical Algorithms", and "Calcolo". He co-authored 4 books, more than 120 papers in internationally established scientific journals, more than 30 papers in conference proceedings, more than 160 talks at international conferences, and more than 120 internal reports. He is a member of ACM, IEEE, and SIAM.

Lieven De Lathauwer received the Ph.D. degree from the Faculty of Engineering, KU Leuven, Belgium, in 1997. From 2000 to 2007 he was Research Associate with the Centre National de la Recherche Scientifique, France. He is currently Professor with KU Leuven. He is affiliated with both the Group Science, Engineering and Technology of Kulak, with the Stadius Center for Dynamical Systems, Signal Processing and Data Analytics of the Electrical Engineering Department (ESAT) and with iMinds Future Health Department. He is Associate Editor of the SIAM Journal on Matrix Analysis and Applications and has served as Associate Editor for the IEEE Transactions on Signal Processing. He is Fellow of the IEEE. His research concerns the development of tensor tools for engineering applications.