Multiresolution Tensor Decompositions with Mode Hierarchies

CLAUDIO SCHIFANELLA, University of Torino
K. SELÇUK CANDAN, Arizona State University
MARIA LUISA SAPINO, University of Torino

Tensors (multidimensional arrays) are widely used for representing high-order dimensional data, in applications ranging from social networks, sensor data, and Internet traffic. Multiway data analysis techniques, in particular tensor decompositions, allow extraction of hidden correlations among multiway data and thus are key components of many data analysis frameworks. Intuitively, these algorithms can be thought of as *multiway clustering* schemes, which consider multiple facets of the data in identifying clusters, their weights, and contributions of each data element. Unfortunately, algorithms for fitting multiway models are, in general, iterative and very time consuming. In this article, we observe that, in many applications, there is a priori background knowledge (or metadata) about one or more domain dimensions. This metadata is often in the form of a hierarchy that clusters the elements of a given data facet (or mode). We investigate whether such single-mode data hierarchies can be used to boost the efficiency of tensor decomposition process, without significant impact on the final decomposition quality. We consider each domain hierarchy as a guide to help provide higher- or lower-resolution views of the data in the tensor on demand and we rely on these metadata-induced multiresolution tensor representations to develop a multiresolution approach to tensor decomposition. In this article, we focus on an alternating least squares (ALS)–based implementation of the two most important decomposition models such as the PARAllel FACtors (PARAFAC, which decomposes a tensor into a diagonal tensor and a set of factor matrices) and the Tucker (which produces as result a core tensor and a set of dimension-subspaces matrices). Experiment results show that, when the available metadata is used as a rough guide, the proposed multiresolution method helps fit both PARAFAC and Tucker models with consistent (under different parameters settings) savings in execution time and memory consumption, while preserving the quality of the decomposition.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]: Clustering

General Terms: Theory, Algorithms

Additional Key Words and Phrases: Tensor decomposition, multiresolution, PARAFAC, Tucker

ACM Reference Format:
Claudio Schifanella, K. Selçuk Candan, and Maria Luisa Sapino. 2014. Multiresolution tensor decompositions with mode hierarchies. ACM Trans. Knowl. Discov. Data 8, 2, Article 10 (May 2014), 38 pages.
DOI: http://dx.doi.org/10.1145/2532169

10

This work is partially funded by NSF grants #116394, "RanKloud: Data Partitioning and Resource Allocation Strategies for Scalable Multimedia and Social Media Analysis" and #1016921, "One Size Does Not Fit All: Empowering the User with User-Driven Integration". This work was also supported by the National Research Foundation Grant funded by the Korean Government (MEST) (KRF-2008-220-D00113). A preliminary version of this paper has appeared as a conference paper: Claudio Schifanella, K. Selçuk Candan, and Maria Luisa Sapino. "Fast metadata-driven multiresolution tensor decomposition." CIKM'11, pp. 1275–1284, 2011.
Author's addresses: C. Schifanella and M. L. Sapino, University of Torino; emails: claunet@gmail.com; marialuisa.sapino@unito.it; K. S. Candan, Arizona State University; email: candan@asu.edu.

DOI: http://dx.doi.org/10.1145/2532169

1. INTRODUCTION

Multidimensional arrays (i.e., tensors) are increasingly used for multiway data representation in diverse applications [Acar and Yener 2009; Kolda and Bader 2009]. Starting from their early use in psychometrics [Tucker 1964], multiway analysis techniques, and in particular tensor decompositions, have been widely used in extraction of hidden correlations among multiway data, in fields, such as information retrieval [Chew et al. 2007], sensor networks analysis [Sun et al. 2007], and web ranking and analysis [Sun et al. 2005; Kolda and Bader 2006].

Intuitively, tensor decomposition algorithms can often be thought of as *multiway clustering* schemes, which consider, simultaneously, multiple facets of the data for identifying clusters, their weights, and contributions of each data element. Tensor decomposition techniques are often placed into three categories: the first group includes the Parallel Factors (PARAFAC) decomposition [Harshman 1970] (also called CANDECOMP [Carroll and Chang 1970]) and its extensions like PARAFAC2 [Harshman 1972], S-PARAFAC [Harshman et al. 2003] and PARALIND [Bro et al. 2009] that decompose a tensor into a smaller diagonal tensor and a set of factor matrices, one for each dimension of the input tensor. In the second category, we find models based on Tucker decomposition [Tucker 1966] that approximate the initial tensor into a core tensor and a set of matrices that represent subspaces of each dimension. The third group contains models like Multilinear Engine [Paatero 1999] and STATIS based multiway models [Stanimirova et al. 2004]. In the literature, there are various algorithms for fitting tensor decomposition models [Sanchez and Kowalski 1990, 1986; Jiang et al. 2000]. In particular, algorithms based on *alternating least squares (ALS)* estimate the decomposition one factor matrix at a time, keeping other factors fixed; the process is repeated until the convergence condition is reached.

Unfortunately, algorithms for fitting multiway models are, in general, very time consuming. In this article, we observe that, in many applications, there is a priori background knowledge (or metadata) about one or more domain dimensions and this metadata is often in the form of a hierarchy that clusters the elements of a given data facet (or mode). We investigate whether such single-mode data hierarchies can help boost the efficiency of tensor decomposition process, without significant impact on the final decomposition quality. Intuitively, we consider each available domain hierarchy as a support that provides different views of the data, with varying resolutions along a single dimension. Using these different resolutions at different stages of the decomposition process, then, helps eliminate redundant work and save execution time.

In this article, we focus our attention on the alternating least squares (ALS)–based algorithms for fitting both PARAFAC and Tucker models. We extend our previous work [Schifanella et al. 2011] introducing a novel multiresolution approach to the decomposition of Tucker models. We also experimentally investigate performances of both PARAFAC and Tucker algorithms taking into account a comprehensive list of parameters. Both algorithms enable the user to pick and choose among the available domain hierarchies, the approximation levels, as well as to vary target *convergence condition* for different levels of the input hierarchies to support tensor decomposition. The technique we propose starts at a low resolution and identifies a decomposition for that resolution. Once the iterations for the given level is complete, the resulting decomposition is expanded, and this expanded decomposition is used as a starting point for the next, higher resolution. This process is repeated one resolution at a time until all the selected levels of the hierarchy have been considered. In this article, we also investigate the impacts of different expansion strategies and different initialization policies.

The proposed multiresolution process provides significant time savings because the initial iterations are much faster due to the lower resolution of the data. Moreover, the number of costly high-resolution iterations are significantly reduced. Experiment results show that, when the available metadata is used as a rough guide (avoiding over-fitting to the lower resolution of the tensors), the proposed multiresolution method helps fit PARAFAC models with consistent (for both dense and sparse tensor representations, under different parameters settings) savings in execution time and memory consumption, while preserving the quality of the obtained decomposition. We obtained a similar behavior for Tucker models, although the benefits are less pronounced. Experiments also show that the proposed method can leverage more than one unimodal hierarchy to further reduce tensor decomposition times.

The structure of the article is as follows: Section 2 provides an overview of the related works and introduces the tensor background knowledge and formalisms used along this article. The details of the multiresolution algorithms are presented in Section 3, while Section 4 describes the implementation and optimization details. Section 5 contains experimental evaluation. We conclude the article in Section 6.

2. BACKGROUND AND RELATED WORKS

In this section, we introduce the tensor notation, present background about operations on tensors, and present other related works.

A tensor is a multidimensional array, where the *order* (also known as the number of ways or modes) N represents the number of dimensions. Following the notation reported in Kolda and Bader [2009], vectors are represented by boldface lowercase letters (e.g., $\mathbf{a}$), matrices are denotated by boldface capital letters (e.g., $\mathbf{A}$), while tensors with order $N \geq 3$ are denoted by Euler script letters (e.g., $\mathcal{X}$). Scalars are denoted by lowercase letters (e.g., a). Thus, an N-order tensor can be formalized as

$$\mathcal{X} \in R^{I_1 \times I_2 \times \cdots \times I_N}.$$

The ith entry of a vector $\mathbf{a}$ is denoted by a_i, the element (i, j) of a matrix $\mathbf{A}$ is denoted by a_{ij}, while the element (i, j, k) of a $3-order$ tensor $\mathcal{X}$ is represented by x_{ijk}.

Subarrays are formed when a subset of the indices is fixed. For example, if we want to identify the jth column of the matrix $\mathbf{A}$, we can write $\mathbf{a}_{:j}$. *Fibers* are defined by fixing every index of a tensor, but one. In a 3-order tensor we can identify column, row and tube fibers, denoted by $\mathbf{x}_{:jk}$, $\mathbf{x}_{i:k}$, $\mathbf{x}_{ij:}$. *Slices* are two-dimensional sections of a 3-order tensor, defined by fixing one index. Horizontal, lateral, and frontal are denoted as $\mathbf{X}_{i::}$, $\mathbf{X}_{:j:}$, and $\mathbf{X}_{::k}$.

2.1. Tensor Decomposition

The two most popular tensor decompositions are the Tucker [Tucker 1966] and the PARAFAC/CANDECOMP [Harshman 1970; Carroll and Chang 1970]. Both can be considered high-order generalizations of the matrix singular value decomposition (SVD) and principal component analysis (PCA).

CANDECOMP [Carroll and Chang 1970] and PARAFAC [Harshman 1970] decompositions (together known as the CP-decomposition) decompose the input tensor into a sum of component rank-one tensors. Given an input tensor $\mathcal{X}$ and a core size r, the PARAFAC decomposition finds r rank-one tensors in the form of $\lambda_i \mathbf{u}_i^{(1)} \circ \cdots \circ \mathbf{u}_i^{(N)}$, where r is a positive integer, $\mathbf{u}_i^{(d)} \in R^{I_d}$ for $1 \leq i \leq r$ and $\circ$ represents the vector outer product. The decomposition approximates the tensor $\mathcal{X}$ by minimizing the value of

$$\left\| \mathcal{X} - \sum_{i=1}^{r} \lambda_i \mathbf{u}_i^{(1)} \circ \cdots \circ \mathbf{u}_i^{(N)} \right\|.$$

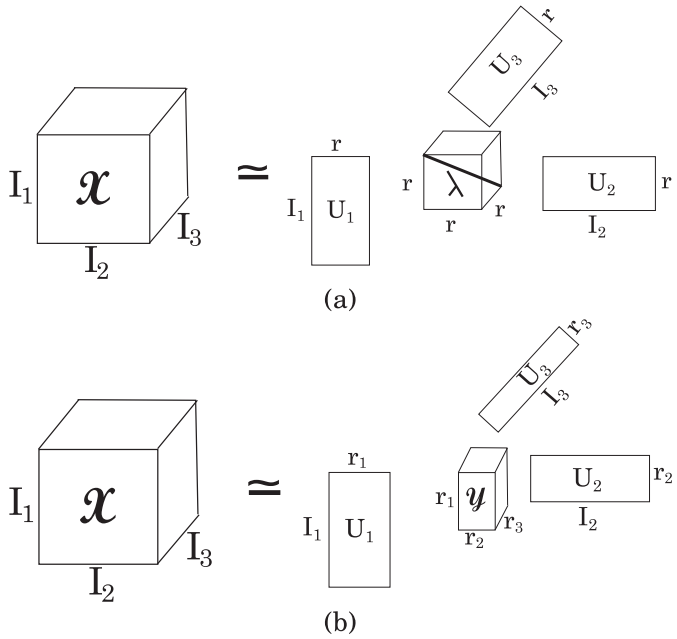


Fig. 1. (a) Parafac and (b) Tucker decompositions for a 3-way tensor.

Alternatively, the decomposition can also be rewritten in terms of factor matrices (see Figure 1(a)) as a combination of the vectors from the rank-one components as

$$\mathcal{X} \approx [\![\lambda; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!] = \sum_{i=1}^{r} \lambda_i \mathbf{u_i}^{(1)} \circ \mathbf{u_i}^{(2)} \cdots \mathbf{u_i}^{(N)},$$

where $\lambda \in R^r$ and $\mathbf{U}^{(i)} \in R^{I_i \times r}$. PARAFAC can be considered as a special case of Tucker decomposition in which the core tensor λ is superdiagonal. Intuitively, as in SVD, PARAFAC enforces that the central matrix is diagonal; however, unlike in SVD, the facet matrices are not guaranteed to be orthonormal.

The Tucker decomposition approximates a tensor using a smaller core tensor through a change of basis represented as a set of matrices, one for each tensor dimension (Figure 1(b)). Intuitively, the Tucker decomposition generalizes singular value matrix decomposition (SVD) to higher-dimensional matrices. One key difference is that Tucker may fail to guarantee an exact decomposition when it is constrained with a user provided target decomposition rank. It is possible to compute an exact Tucker decomposition only when the required number of factors is equal to the component rank, for each mode of the tensor. Another source of inexactness is due to the cost of the decomposition algorithms; in many situations, to avoid this cost, users rely on fast, but inexact approximation algorithms [Kolda and Bader 2009]. Most approaches involve searching for orthonormal facet matrices and a core tensor that collectively minimize the decomposition error. For example, the high-order SVD approach first identifies the left eigenvectors (with the highest eigenvalues) of the lateral, horizontal, and frontal slices to construct the facet matrices. Given as input a tensor $\mathcal{X}$ and a set $R = \{r_1, \ldots, r_N\}$ of core sizes, the Tucker decomposition computes a smaller core tensor and a set of factor matrices. More formally,

$$\mathcal{X} \approx [\![\mathcal{Y}; \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \ldots, \mathbf{V}^{(N)}]\!] = \mathcal{Y} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \cdots \times_N \mathbf{V}^{(N)},$$

where $\mathcal{Y} \in R^{r_1 \times \cdots \times r_N}$, $\mathbf{V}^{(i)} \in R^{I_i \times r_i}$ and $\times_i$ represents the *i-mode* tensor-matrix product. The decomposition process approximates the input tensor by minimizing the value of

$$\left\| \mathcal{X} - \mathcal{Y} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \cdots \times_N \mathbf{V}^{(N)} \right\|.$$

2.2. Obtaining Tensor Decompositions

Many of the algorithms for fitting multiway models are based on an iterative process that approximates the best solution until a convergence condition is reached.

The ALS method [Yates 1933] for PARAFAC models is relatively old and has been successfully applied to the problem of tensor decomposition by Carroll and Chang [1970] and Harshman [1970]. ALS is widely used as a building block in many applications for fitting PARAFAC and Tucker models [Kolda and Bader 2006; Sun et al. 2009; Kolda and Sun 2008]. ALS estimates, at each iteration, one factor matrix, maintaining other matrices fixed; this process is repeated for each factor matrix associated to the dimensions of the input tensor. Since our proposed multiresolution approach extends the basic ALS, in Algorithm 1 we present the pseudocode of the standard ALS fitting for PARAFAC. The algorithm takes as input the data tensor $\mathcal{X}$, the number of factors r, a tolerance value *tol* (used as a stopping condition), and an *optional* initial factorization $\mathcal{F}_{init}$. The algorithm returns the factor matrices and the elements of the resulting diagonal tensor λ. Other alternation-based algorithms for PARAFAC models include the alternating slice-wise diagonalization (ASD) [Jiang et al. 2000] and the self-weighted alternating trilinear diagonalization (SWA-TLD) [Chen et al. 2000] algorithms: they improve fitting using objective functions that are not based on least squares.

ALGORITHM 1: Alternating least squares fit algorithm for PARAFAC models

Input: $\mathcal{X}, r, tol, \mathcal{F}_{init}$
Output: $\mathcal{F} \equiv \lambda, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}$
Initialize $\mathbf{U}^{(i)} i = 1 \ldots N$ to $\mathcal{F}_{init}$ (random if $\mathcal{F}_{init}$ is not present)
do
  $fit = \|\mathcal{X} - [\![\lambda; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]\|$
  for $i = 1 \ldots N$ **do**
    $\mathbf{U}^{(i)} = \mathbf{X}^{(i)}[(\mathbf{U}^{(1)} \cdots \odot \mathbf{U}^{(i-1)} \odot \mathbf{U}^{(i+1)} \odot \cdots \odot \mathbf{U}^{(N)})^\top]^\dagger$
    normalize column of $\mathbf{U}^{(i)}$ (norms are stored in λ)
  end
  $fitchange = fit - \|\mathcal{X} - [\![\lambda; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]\|$
while $fitchange \geq tol$;
return $\mathcal{F} \equiv \lambda, \boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(N)}$

The first iterative method for computing Tucker decompositions was introduced by Tucker in 1966 [Tucker 1966]. This method was extended by De Lathauwer et al. [2000a] and is known as *high-order SVD* (HOSVD). In fact, the authors show that it is a generalization of the matrix singular value decomposition (SVD) and introduce a way to compute more efficiently the leading left singular vectors of a Tensor matricization. In 1980, Kroonenberg and De Leeuw introduced TUCKALS3 [Kroonenberg and de Leeuw 1980], an ALS algorithm that is limited to three-way tensor support that was successively extended by Kapteyn et al. [1986]. In this article, we adopt as building block for the Tucker-based multiresolution approach the ALS algorithm called *higher-order orthogonal iteration* (HOOI), introduced by De Lathauwer et al. [2000b]. The authors propose a more efficient way for calculating the factor matrices based on

ALGORITHM 2: Alternating least squares fit algorithm for Tucker models, based on the HOOI approach

Input: $\mathcal{X}, R, tol, \mathcal{F}_{init}$
Output: $\mathcal{F} \equiv \mathcal{Y} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \cdots \times_N \mathbf{V}^{(N)}$
Initialize $\mathbf{V}^{(i)} i = 1 \ldots N$ to $\mathcal{F}_{init}$ (random if $\mathcal{F}_{init}$ is not present)
do
  $fit = \|\mathcal{X} - [\![\mathcal{Y}; \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \ldots, \mathbf{V}^{(N)}]\!]\|$
  for $i = 1 \ldots N$ **do**
    $\mathcal{G} = \mathcal{X} \times_1 \mathbf{V}^{(1)\top} \cdots \times_{i-1} \mathbf{V}^{(i-1)\top} \times_{i+1} \mathbf{V}^{(i+1)\top} \cdots \times_N \mathbf{V}^{(N)\top}$
    $\mathbf{V}^{(i)} = r_i$ leading eigenvalues of $\mathbf{G}_{(i)}$
  end
  $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{V}^{(1)\top} \times_2 \mathbf{V}^{(2)\top} \cdots \times_N \mathbf{V}^{(N)\top}$
  $fitchange = fit - \|\mathcal{X} - [\![\mathcal{Y}, \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \ldots, \mathbf{V}^{(N)}]\!]\|$
while $fitchange \geq tol$;
return $\mathcal{F} \equiv \mathcal{Y} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \cdots \times_N \mathbf{V}^{(N)}$

the computation of only the dominant singular vectors of a tensor matricization. Algorithm 2 shows the pseudo-code of the standard HOOI approach: it takes as input the data tensor $\mathcal{X}$, the set R of required number of factors for each dimension, a tolerance value *tol* (used as a stopping condition), and an *optional* initial factorization $\mathcal{F}_{init}$. The algorithm returns the factor matrices and the core tensor $\mathcal{Y}$.

Closed-Form and Gradient-Based Algorithms. Noniterative approaches to PARAFAC tensor decomposition include closed form solutions, such as generalized rank annihilation method (GRAM) [Sanchez and Kowalski 1986] and direct trilinear decomposition (DTLD) [Sanchez and Kowalski 1990], which fit the model by solving a generalized eigenvalue problem. The PMF3 algorithm is based on a modified version of a Gauss-Newton method. An in-depth study and comparison of these algorithms can be found in Tomasi and Bro [2006] and Faber et al. [2003]. For what concerns Tucker models, an approach based on a Newton-Grassmann optimization problem was recently proposed [Eldén and Savas 2009].

Gradient-based methods include the Multilinear Engine [Paatero 1999]: an approach based on the conjugate gradient algorithm that can also compute different multilinear models. Acar et al. [2011] introduced a scalable gradient-based optimization approach for tensor decompositions. More recently, in Ishteva et al. [2011] the tensor approximation problem is expressed as minimization of a cost function on a product of three Grassmann manifolds. The authors apply the Riemannian trust-region scheme, using the truncated conjugate-gradient method for solving the trust-region subproblem.

Tensor Decomposition with External Knowledge. Chi and Zhu [2010] present FacetCube, a framework that allows users to incorporate prior knowledge in the nonnegative Tucker decomposition process, with the primary goal of helping users enforce facets on certain data dimensions. In that work, the metadata can be available in different forms. The first one is prior knowledge describing a subspace from which facets can be located. In the second form, the users require facets for some data dimensions be fixed. For example, in an $author \times reference \times keyword$ tensor, the paper shows how additional information on authors (i.e., the coauthor relationship) can be leveraged in a recommendation system of references. In our multiresolution approach, instead, additional metadata are used to compute lower-level resolution representations of the input tensor, in order to speed up the decomposition process.

Speeding Up the Tensor Decomposition. In the literature, we can find many approaches that aim to reduce the tensor decomposition time. Bro and Andersson [1998] propose to speed up the PARAFAC model computation using a Tucker-based compression technique. The input tensor is compressed by computing a Tucker model that produces a core tensor $\mathcal{Y}$ and a set of corresponding factor matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}$. The compressed core tensor $\mathcal{Y}$ is then used to compute an intermediate PARAFAC model $[\![\lambda; \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \ldots, \mathbf{V}^{(N)}]\!]$. The final PARAFAC model is then reconstructed by combining the factor matrices from the initial Tucker decomposition and the following PARAFAC computation $[\![\lambda; \mathbf{V}^{(1)} \times \mathbf{U}^{(1)}, \mathbf{V}^{(2)} \times \mathbf{U}^{(2)}, \ldots, \mathbf{V}^{(N)} \times \mathbf{U}^{(N)}]\!]$. The quality of the final result is ensured by the optimality theorem of the CANDELINC model [Carroll et al. 1980]. This approach is related to our proposal, and in Section 5 we experimentally study its applicability and limitations, both under dense and sparse tensor encodings.

A problem that can negatively affect the decomposition time is the so-called two factor degeneracy problem, that is, the presence of two factors that are colinear with opposite signs. This potentially causes a mutual cancellation of factors' contributions during iterative ALS, resulting in a very slow decrease of the objective function. One of the key solutions to this problem is the *line search* technique [Ross and Leurgans 1995; Bro 1998]. Line search is commonly applied for PARAFAC (but it can be used also with derivative based methods [Madsen et al. 2004]) to speed up the convergence by leveraging regression to estimate factor matrices based on a subset of past iterations instead of relying solely on the previous iteration. In this sense, it is applicable to all iterative methods and thus is orthogonal to our proposed scheme: the proposed metadata supported multiresolution scheme can also benefit from line search in cases where the factors of the data show colinearity with opposite signs. Rajih et al. [2008] propose a way to improve line search performance by solving an optimization problem over the line search parameters: they show that their proposed enhanced line search scheme allows to improve the execution time under certain conditions, such as three-way tensors with factor colinearities in one of the modes.

Finally, Kaarna et al. [2007] leverage a multiresolution approximation of the initial tensor in the ALS-based decomposition process for image analysis. Authors recognize that two contiguous pixels in an image can be approximated down to a single pixel because they are spatially correlated. In particular, Kaarna et al. [2007] relies on the hierarchical Integer Wavelet Transformations (IWT) to speed-up the decomposition process for image datasets. There are two main differences between Kaarna et al. [2007] and the approach presented in this article: First of all, unlike in Kaarna et al. [2007], where the (wavelet) hierarchy relies on the inherent spatial correlation *internal to the data*, here we investigate the applicability of external metadata in supporting the tensor decomposition process. Secondly, unlike Kaarna et al. [2007] where the data (i.e., image) tensor is inherently dense, we aim to also tackle sparse tensors (which are common in many applications, including social networks and graph structured data).

3. A MULTIRESOLUTION APPROACH FOR FITTING PARAFAC AND TUCKER MODELS

As we pointed out in the Introduction, in this article we observe that in general there may be external knowledge regarding how the domains represented by the individual tensor dimensions are hierarchically clustered (according to different clustering criteria). In this section, we show how such background knowledge can be leveraged as a guide to help provide higher- or lower-resolution views of the data in the tensor on demand and develop a multiresolution approach to tensor decomposition (Figure 2).

3.1. Multiresolution Tensor Representation

Let D_i be the domain associated to the $i-th$ dimension of a tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \cdots \times I_N}$. Here, I_i is the cardinality of D_i, that is, $D_i = \{d_{i,1}, d_{i,2}, \ldots, d_{i,I_i}\}$.

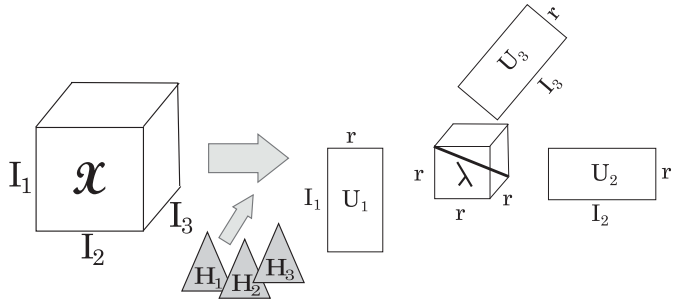


Fig. 2. Leveraging metadata during tensor decomposition.

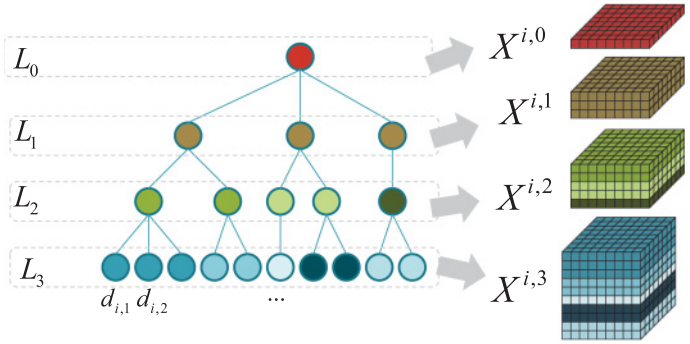


Fig. 3. Hierarchy-based multiresolution representation.

Clustering hierarchies capture different strategies along with the domain elements associated to the corresponding dimension can be grouped together. For example, if one dimension of the tensor is associated to the *users* accessing a given service, the corresponding domain elements may be clustered according to their age, according to the city/province/region/state where they live, or according to the preferences listed in their profiles. Let the tree H_i, of depth $depth_i$, be a hierarchical clustering defined on D_i (representing some semantic properties of the values associated to the corresponding tensor dimension). Let $nodes(H_i, l)$ be the set of nodes appearing at level l ($0 \leq l \leq depth_i$) in H_i. Note that the following holds:

—$|nodes(H_i, l)| < |nodes(H_i, j)|$ if $l < j$;
—let $n_{i,l,k}$ be the k^{th} node at level l in the hierarchy H_i with $k = 1, \ldots, |nodes(H_i, l)|$ and $coverage(n_{i,l,k})$ be the set of leaves of the subtree rooted at $n_{i,l,k}$. Then, $coverage(n_{i,0,1}) = D_i$, that is, the root of the hierarchy is a single cluster that covers the entire domain D_i, and $coverage(n_{i,l,k}) > coverage(n_{i,j,m})$ if $n_{i,l,k}$ is an ancestor of $n_{i,j,m}$.

We leverage the given domain hierarchical clustering strategy to define multiresolution representations of a given tensor (Figure 3). Intuitively, this representation leads to compact representations of the tensor, in which (multiple) tensor values of domain elements which are clustered together according to the given hierarchy are collapsed in a single value, which is seen as the representative of the entire cluster.

Given a domain D_i and a corresponding hierarchy H_i, the *representative selection function* is the function $repr_i : 2^R \rightarrow R$, which maps a set of real values (in the tensor, corresponding to the positions indexed by the elements clustered in H_i) to a single

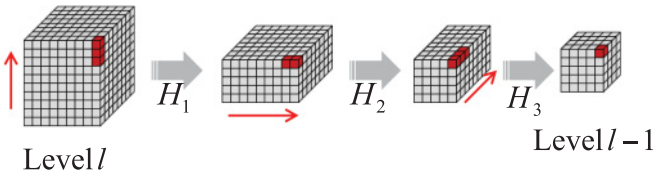


Fig. 4. The input tensor is compressed one mode at a time with respect to a given order defined in the clustering embedding strategy.

value, chosen as the representative of the cluster. Examples of representative selection functions include min, max, and average. Notice that for any given domain D_i and any clustering defined over it, multiple representative selection functions can be defined. When a dimension does not have an associated hierarchy, we denote this using $H_i = \perp$ and $repr_i = self$.

Definition 3.1 (*Clustering Embedding Strategy and Mode Hierarchy*). Given a tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \cdots \times I_N}$, a clustering embedding strategy over $\mathcal{X}$ is a pair $S = \langle \pi, \{(H_1, repr_1, l_1), \ldots, (H_N, repr_N, l_N)\}\rangle$. Here, π is a permutation over the indices of the tensor modes (describing the order in which clusterings are considered among different modes, Figure 4), each H_i is a clustering hierarchy defined on D_i, $repr_i$ is a corresponding representative selection function, and l_i is the clustering level adopted for the i^{th} mode. We refer to a hierarchy associated to a mode as the *mode hierarchy*.

Given a tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \cdots \times I_N}$, and a set $\{H_1, \ldots, H_N\}$ of mode hierarchies defined on the N domains associated to the tensor dimensions we define the corresponding multiresolution representation of $\mathcal{X}$ with respect to the set $\{H_1, \ldots, H_N\}$ as follows. Note that we first define unimodal representation with respect to a single hierarchy and, then, extend this to multimodal representation, which considers multiple hierarchies simultaneously.

Definition 3.2 (*Unimodal Multiresolution Representation*). The multiresolution representation at level l_i with respect to the clustering hierarchy H_i and the representative selection function $repr_i$ of a tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \cdots \times I_N}$, is a function $\mathcal{U}_{H_i, repr_i, l_i}(\mathcal{X})$:

$$\mathcal{U}_{H_i, repr_i, l_i}(\mathcal{X}) : R^{I_1 \times \cdots \times I_N} \rightarrow R^{I_1 \times \ldots \times I_{i-1} \times I_i' \times I_{i+1} \cdots \times I_N},$$

where $I_i' = |nodes(H_i, l)|$. We also refer to $\mathcal{U}_{H_i, repr_i, l_i}(\mathcal{X})$ as $\mathcal{X}^{i,l_i}$ in shorthand:

—If $l_i \geq depth_i$, then $\mathcal{X}^{i,l_i} = \mathcal{X}$.
—Otherwise, let $C = \{C_1, \ldots, C_{|nodes(H_i, l_i)|}\}$ be the clustering induced by H_i at level l_i on the domain elements D_i. Let also the leaf-cardinality of the cluster C_h be denoted as $coverage(n_{i,l_i,h})$. Then, for $h = 1, \ldots, |nodes(H_i, l_i)|$, we have

$$\mathcal{X}^{i,l_i}_{j_1 \ldots j_{i-1} h j_{i+1} \ldots j_N} = repr_i(\{\mathcal{X}_{j_1 \ldots j_{i-1} k j_{i+1} \ldots j_N} \mid d_k \in C_h\}).$$

Intuitively, for any cluster $C_h \in C$ along the i^{th} dimension, for all indices along the other modes, $|C_h|$ elements of the given tensor are collapsed and a single representative value, resulting from the application of the function $repr_i$ on the collapsed elements, appears in the l_i-resolution tensor (Figure 5).

Figure 3 is a visual representation of a unimodal multiresolution representation of a 3-mode tensor. For example, if the domain associated to the considered hierarchy is

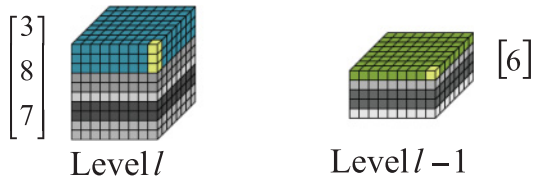


Fig. 5. The use of the representative selection function (*average* in this case).

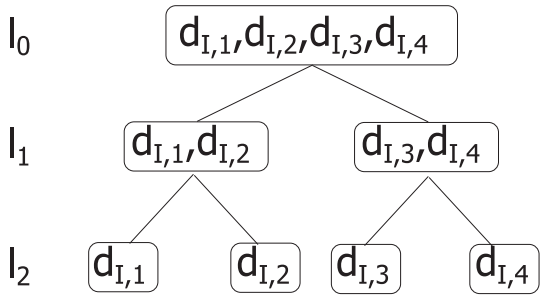


Fig. 6. An example hierarchy H_I encoding additional metadata.

a set of users, the multiresolution approach can be seen as hierarchically partitioning them in communities linked by a parent-child relationship.

Example 3.3. Let us consider the 3-mode I, J, K tensor $\mathcal{X} \in R^{4\times2\times3}$:

$$\mathbf{X}_{1::} = \begin{bmatrix} 0 & 5 & 2 \\ 1 & 0 & 9 \end{bmatrix} \qquad \mathbf{X}_{2::} = \begin{bmatrix} 2 & 7 & 0 \\ 3 & 0 & 9 \end{bmatrix}$$

$$\mathbf{X}_{3::} = \begin{bmatrix} 0 & 2 & 3 \\ 1 & 0 & 4 \end{bmatrix} \qquad \mathbf{X}_{4::} = \begin{bmatrix} 2 & 0 & 3 \\ 3 & 0 & 6 \end{bmatrix}$$

Moreover, consider the hierarchy H_I depicted in Figure 6 that clusters elements of the dimension I and the *average* function as representative selection function. The tensor $\mathcal{X}^{I,1} \in R^{2\times2\times3}$

$$\mathbf{X}^{I,1}_{1::} = \begin{bmatrix} 1 & 6 & 1 \\ 2 & 0 & 9 \end{bmatrix} \qquad \mathbf{X}^{I,1}_{2::} = \begin{bmatrix} 1 & 1 & 3 \\ 2 & 0 & 5 \end{bmatrix}$$

represents the lower-resolution representation at level 1 of the original tensor $\mathcal{X} = \mathcal{X}^{I,2}$.

Given the definition of unimodal multiresolution representation, we can also define the multimodal multiresolution representation as follows:

Definition 3.4 (*N-modal Multiresolution Representation*). The multiresolution representation with respect to the clustering embedding strategy $S = \langle \pi, \{(H_1, repr_1, l_1), \ldots, (H_N, repr_N, l_N)\}\rangle$ of the tensor $\mathcal{X} \in R^{I_1\times I_2\times\cdots\times I_N}$, is the function $\mathcal{M}_S(\mathcal{X})$:

$$\mathcal{M}_S(\mathcal{X}) : R^{I_1\times I_2\times\cdots\times I_N} \rightarrow R^{I'_1\times I'_2\times\cdots\times I'_N},$$

where $I'_i = |nodes(H_i, l_i)|$. We also refer to $\mathcal{M}_S(\mathcal{X})$ as $\mathcal{X}^S$ in shorthand. $\mathcal{X}^S$ is inductively defined as follows: Let $\mathcal{X}^S_1 = \mathcal{X}^{\pi[1],l_{\pi[1]}}$, be the unimodal multiresolution representation based on the first mode to be considered according to the permutation π specified in S. Then, we have $\mathcal{X}^S = \mathcal{X}^S_N$, where $\forall\ 1 < s \leq N$, the multimodal representation is defined,

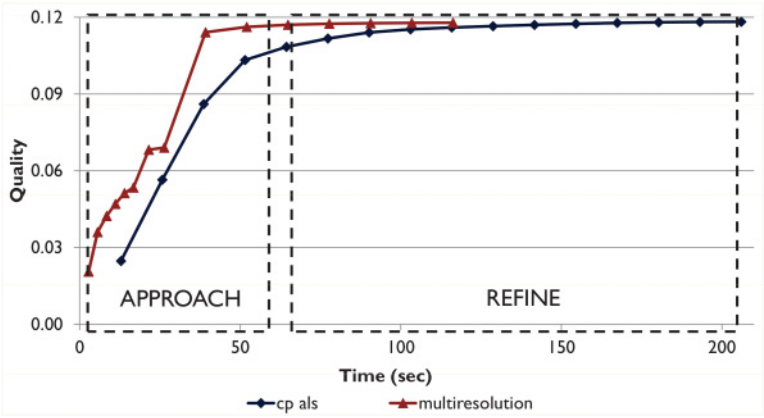


Fig. 7. A sample execution trace of a standard ALS PARAFAC decomposition compared with our multiresolution-based proposal.

relying on the unimodal definition, as follows:

$$\mathcal{X}_s^S = \left(\mathcal{X}_{s-1}^S\right)^{\pi[s],I_{\pi[s]}}.$$

3.2. Multiresolution Tensor Decomposition

In this section, we discuss how we leverage the multiresolution tensor representation to boost the efficiency of tensor decomposition process, without any significant impact on the final decomposition quality. The basic idea relies on the observation that the ALS algorithms [Yates 1933] both for PARAFAC and Tucker tensor decompositions are iterative processes that start from an initial randomly generated solution, on which each iterative step improves, until a satisfactory decomposition (i.e., an approximated decomposition differing not more than a predetermined threshold from an exact decomposition) of the given tensor is found. Figure 7 presents a sample execution trace of a standard ALS PARAFAC decomposition (the blue line). As can be seen, during this process we can identify two main phases: in the *approach* phase, the algorithm roughly goes closer to the final solution in few steps, while in the *refine* phase, the quality value slowly approaches the final result until the stopping condition is verified. In this article, we propose to use the approximated multiresolution representations during the early stages of the decomposition process (i.e., in the *approach* phase, the red line) in order to reduce the execution time and to use the less approximated tensor representations in the *refinement* phase, where the value of the quality is more important.

Based on this, we expect that an "informed" initialization of the decomposition process could significantly reduce the computational cost of the iterative process. In particular, we choose to initialize the decomposition process with the approximate decomposition resulting from the ALS decomposition of a "simpler" (i.e., lower resolution) version of the given tensor. We show in Section 5 that the computational cost of this guided decomposition (including the cost of obtaining the lower-resolution decomposition plus the cost of the iterative process starting from the initialization) is lower than the cost in the global decomposition from a randomly initialized iterative process.

Algorithm 3 contains the pseudocode of the PARAFAC multiresolution-based algorithm. The algorithm takes as input the tensor, $\mathcal{X}$, the set of hierarchies, $\mathcal{H}$, the set of representative selection functions, $\mathcal{R}$, the approximation levels to consider, ρ, the number of required factors r, and the convergence threshold (or tolerance value), *tol*, used as stopping condition.

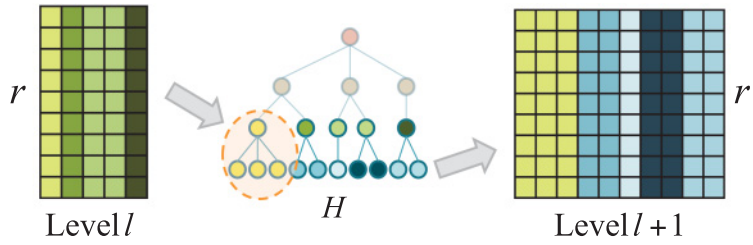


Fig. 8. Expansion of the factor matrices based on the $expand\_factor()$ function.

ALGORITHM 3: Multiresolution alternating least squares fit for PARAFAC models.

Input: $\mathcal{X}, \mathcal{H}, \mathcal{R}, \rho, r, tol$
Output: $\mathcal{F}^{[0]} \equiv \lambda, \mathbf{U}^{(1)[0]}, \ldots, \mathbf{U}^{(N)[0]}$
1 $S_\rho = clustering\_strategy(\mathcal{X}, \mathcal{H}, \mathcal{R}, \pi, \rho)$
2 $\mathcal{F}^{[\rho]} = parafac\_als(\mathcal{X}^{S_\rho}, r, tol)$
3 $resol = \rho$
4 **while** $resol > 0$ **do**
5 **foreach** $\mathbf{U}^{(i)[resol]}$ *not related to a compressed dimension* **do**
6 $\mathbf{U}^{(i)[resol-1]} = \mathbf{U}^{(i)[resol]}$
7 **end**
8 **foreach** $\mathbf{U}^{(i)[resol]}$ *related to a compressed dimension* **do**
9 $\mathbf{U}^{(i)[resol-1]} = expand\_factor(\mathbf{U}^{(i)[resol]}, \mathcal{H})$
10 **end**
11 $\mathcal{F}^{[resol-1]} \equiv \lambda, \mathbf{U}^{(1)[resol-1]}, \ldots, \mathbf{U}^{(N[resol-1]}$
12 load $\mathcal{X}^{[resol-1]}$
13 $\mathcal{F}^{[resol-1]} = parafac\_als(\mathcal{X}^{[resol-1]}, r, tol, \mathcal{F}^{[resol-1]})$
14 $resol = resol - 1$
15 **end**
16 **return** $\mathcal{F}^{[0]} \equiv \lambda, \mathbf{U}^{(1)[0]}, \ldots, \mathbf{U}^{(N)[0]}$

The first step of the algorithm constructs a clustering strategy, $S_\rho = \langle \pi, \{(H_1, repr_1, depth_1 - \rho), \ldots, (H_N, repr_N, depth_N - \rho)\}\rangle$, for the lowest resolution level.[1] Here, $H_i \in \mathcal{H}$, $repr_i \in \mathcal{R}$, and π is a permutation of the indices. Let tensor $\mathcal{X}^{S_\rho}$ correspond to the representation of $\mathcal{X}$ at the lowest resolution based on this strategy.

Starting from this lowest-resolution tensor $\mathcal{X}^{S_\rho}$, the algorithm first computes a lowest-resolution factorization $\mathcal{F}^{[\rho]}$ by using the standard ALS fitting technique (line 2). This produces the (lowest-resolution) set of factor matrices $\{\mathbf{U}^{(i)[\rho]} 1 \leq i \leq N\}$, in which the matrix $\mathbf{U}^{(i)[\rho]}$ belongs to the tensor dimension I_i. The output factorization $\mathcal{F}^{[\rho]}$ will be used as the starting point of the next invocation of the ALS tensor decomposition, with $(\rho - 1)^{th}$ approximation.

In order to obtain the more detailed factorization corresponding to the $(resol - 1)^{th}$ approximation from $\mathcal{F}^{[resol]}$, we need to transform the set of factor matrices by *expanding* the domains of the relevant dimensions (Figure 8)—the number of elements of dimension that were not compressed by the strategy S are kept the same (lines 5–7). The $expand\_factor()$ function (line 9) expands the given factor matrix $\mathbf{U}^{(i)[resol]}$ to the initial factorization of the corresponding higher-level factor matrix $\mathbf{U}^{(i)[resol-1]}$. Once the

[1]Note that ρ represents the number of considered levels of resolution. Given the hierarchy H_i, the algorithm considers the levels from $depth_i - \rho$ to $depth_i$.

decomposition $\mathcal{F}_{init}^{[resol-1]}$ is computed using the $expand\_factor()$ function (more details can be found in the last part of this section), the algorithm loads the tensor $\mathcal{X}^{[resol-1]}$ which represents the $(resol-1)^{th}$ approximate resolution view of the original tensor and computes the ALS fitting starting from the initial factorization $\mathcal{F}_{init}^{[resol-1]}$ to obtain $\mathcal{F}^{[resol-1]}$. The algorithm ends when the highest level tensor factorization is computed and returns the corresponding final factorization $\mathcal{F}^{[0]}$ (line 16).

In Algorithm 4, we propose the pseudocode for our multiresolution approach to Tucker decomposition, based on the higher-order orthogonal iteration (HOOI) algorithm [Lathauwer et al. 2000b]. As can be seen, the process shares the same rationale with the algorithm for PARAFAC introduced earlier: it takes as input the tensor, $\mathcal{X}$, the set of hierarchies, $\mathcal{H}$, the set of representative selection functions, $\mathcal{R}$, the approximation levels to consider, ρ, the set R of the number of required factors $r_1, \ldots, r_N$ for each dimension, and the convergence threshold (or tolerance value), tol, used as stopping condition. After the iterative process based on subsequent tensor decompositions at different levels of resolution, it returns the final factorization $\mathcal{F}^{[0]} \equiv \mathcal{Y}, \mathbf{V}^{(1)[0]}, \ldots, \mathbf{V}^{(N)[0]}$.

ALGORITHM 4: Multiresolution alternating least squares fit for Tucker models.

Input: $\mathcal{X}, \mathcal{H}, \mathcal{R}, \rho, R, tol$
Output: $\mathcal{F}^{[0]} \equiv \mathcal{Y}, \mathbf{V}^{(1)[0]}, \ldots, \mathbf{V}^{(N)[0]}$
1 $S_\rho = clustering\_strategy(\mathcal{X}, \mathcal{H}, \mathcal{R}, \pi, \rho)$
2 $\mathcal{F}^{[\rho]} = tucker\_hooi(\mathcal{X}^{S_\rho}, R, tol)$
3 $resol = \rho$
4 **while** $resol > 0$ **do**
5 **foreach** $\mathbf{V}^{(i)[resol]}$ *not related to a compressed dimension* **do**
6 $\mathbf{V}^{(i)[resol-1]} = \mathbf{V}^{(i)[resol]}$
7 **end**
8 **foreach** $\mathbf{V}^{(i)[resol]}$ *related to a compressed dimension* **do**
9 $\mathbf{V}^{(i)[resol-1]} = expand\_factor(\mathbf{V}^{(i)[resol]}, \mathcal{H})$
10 **end**
11 $\mathcal{F}^{[resol-1]} \equiv \mathcal{Y}, \mathbf{V}^{(1)[resol-1]}, \ldots, \mathbf{V}^{(N[resol-1]}$
12 load $\mathcal{X}^{[resol-1]}$
13 $\mathcal{F}^{[resol-1]} = tucker\_hooi(\mathcal{X}^{[resol-1]}, R, tol, \mathcal{F}^{[resol-1]})$
14 $resol = resol - 1$
15 **end**
16 **return** $\mathcal{F}^{[0]} \equiv \mathcal{Y}, \mathbf{V}^{(1)[0]}, \ldots, \mathbf{V}^{(N)[0]}$

As mentioned earlier, in both algorithms proposed in this work, the input of a tensor decomposition at level $resol$ is represented by the output of the previous decomposition process at the lower level of resolution $resol+1$. During this step, we need to *adapt* the tensor size of $\mathcal{F}^{[resol+1]}$ according to those required as input at level $resol$ by means of the $expand\_factor()$ function, described in the following. Let $N_{i,resol} = nodes(H_i, depth_i - resol)$ be the set of nodes at the $resol^{th}$ approximation level for the hierarchy H_i; similarly, let $N_{i,resol-1}$ be the set of nodes at the $(resol-1)^{th}$ approximation level. For a given node, $n_h \in N_{i,resol}$ let $children(n_h) \subseteq N_{i,resol-1}$ be the corresponding set of children at the $(resol-1)^{th}$ approximation level.

The $expand\_factor()$ function considers the dimensions of the tensor one at a time; that is, for all $1 \leq i \leq N$, we have

$$\forall n_h \in N_{i,resol} \quad \forall n_k \in children(n_h) \quad \left(\mathbf{U_{init}}^{(i)[resol-1]}\right)_{j_1 \ldots j_{i-1} k j_{i+1} \ldots j_N} = expand_i\left(\left(\mathbf{U}^{(i)[resol]}\right)_{j_1 \ldots j_{i-1} h j_{i+1} \ldots j_N}\right).$$

Note that both of the multiresolution algorithms share with the standard ALS approach both time complexity class and memory consumption upper bound.

4. IMPLEMENTATION ALTERNATIVES AND OPTIMIZATIONS

In this section, we describe additional implementation and optimization alternatives to the proposed multiresolution approaches.

4.1. Initialization Methods

As shown in Algorithm 2, at each iteration each factor matrix $\mathbf{V}^{(i)}$ is updated starting from all the remaining factor matrices $\mathbf{V}^{(j)}$ $(i \neq j)$. For this reason, if at each iteration the algorithm starts from the tensor dimension i, the corresponding initial value of $\mathbf{V}^{(i)}$ can be considered irrelevant. In our experiments, we consider three different initialization methods of factor matrices:

—pure random (denoted as r);
—eigenvectors-based (denoted as v): let r_i be the number of factors related to the dimension I, this method initializes the factor matrix $\mathbf{V}^{(i)}$ by computing the r_i leading eigenvalues of $\mathcal{X}_{(i)} \times \mathcal{X}'_{(i)}$, where $\mathcal{X}_{(i)}$ is the mode-n matricization of the input tensor $\mathcal{X}$ [Bader and Kolda 2007];
—hierarchy-based (denoted as h): we propose a metadata-driven initialization method based on hierarchy relationships. Let $r_i \in \{nodes(H_i, l), 0 \leq l \leq depth_i\}$ be the required target number of factors for the dimension i, and let $n_{i,l,k}$ be the k^{th} node at level l. For each $1 \leq x \leq |nodes(H_i, l)|$, $1 \leq y \leq |nodes(H_i, depth_i|$:

$$\mathbf{V}^{(i)}(x, y) = \begin{cases} 1 & n_{i,depth_i,y} \in coverage(n_{i,l,x}) \\ 0 & otherwise \end{cases}$$

In Section 5.4, we evaluate the impacts of the different initialization strategies for the factor matrices.

4.2. Optimizations in Intermediate Levels

As shown in Figure 7, the proposed multiresolution scheme uses the intermediate levels of the input hierarchies for quickly approaching the final result and relies on the final refinement level for seeking the result.

4.2.1. Relaxing the Convergence Thresholds in Intermediate Levels. One observation is that, since the results of the intermediate levels are rough approximations of the final result anyhow, less strict convergence thresholds might be sufficient in these intermediate *approaching* stages of the process. These less strict convergence thresholds would prevent the algorithm from spending too much time unnecessarily refining the results of the intermediate levels.

Note that, in the extreme case, the intermediate levels can be executed only once each for very quickly approaching the final refinement stage which uses the leaf levels with tight convergence threshold.

4.2.2. Nonzero Removal in Intermediate Levels. If, as we hypothesize, relaxing the precision at the lower resolutions by using a more relaxed convergence threshold could provide boosts in execution times, without negatively affecting the final quality, this may mean that we might be able to also save time (especially in sparse representations) by reducing the number of nonzero entries considered in the lower-resolution steps. This optimization can be achieved by introducing nonzero cutoffs, θ_{nz}, for non-leaf levels and eliminating all entries smaller than θ_{nz} from the tensors, creating less precise approximations of the low-resolution tensors.

In Section 5, we evaluate the impacts of these two optimization schemes along with the other decision parameters.

4.3. Expansion Alternatives

Remember from the previous section that the $expand\_factor()$ function considers the dimensions of the tensor one at a time and uses the $expand_i$ function to expand the factor matrices. While the $expand_i$ function can be chosen in a domain specific manner, in the absence of additional domain information, it is possible to use the identity function:

$$\forall n_h \in N_{i,resol} \quad \forall n_k \in children(n_h) \quad \left(\mathbf{U_{init}}^{(i)[resol-1]}\right)_{j_1 \ldots j_{i-1} k j_{i+1} \ldots j_N} = \left(\mathbf{U}^{(i)[resol]}\right)_{j_1 \ldots j_{i-1} h j_{i+1} \ldots j_N}.$$

Another strategy that we tested in the following experimental evaluation section is the *proportional* approach, where the number of children for each node at level *resol* is considered during the expansion phase:

$$\forall n_h \in N_{i,resol} \quad \forall n_k \in children(n_h) \quad \left(\mathbf{U_{init}}^{(i)[resol-1]}\right)_{j_1 \ldots j_{i-1} k j_{i+1} \ldots j_N} = \frac{\left(\mathbf{U}^{(i)[resol]}\right)_{j_1 \ldots j_{i-1} h j_{i+1} \ldots j_N}}{|children(n_h)|}.$$

5. EXPERIMENTAL EVALUATION

In this section, we compare our metadadata driven, multiresolution tensor decomposition method with the standard ALS-based decomposition, both for PARAFAC and Tucker models.

5.1. Setup

We consider matrices with both dense and sparse representations. As reference software library, we used the Matlab Tensor Toolbox [Bader and Kolda 2007] by Kolda. All experiments are performed on a desktop with a dual core 2.5GHz processor and 4GB RAM. We repeated all experiments 20 times: all reported values represent the average value of all runs.

5.1.1. Experiment Parameters. In this section, we consider the effects of various problem and system parameters. These include

—size of the input tensor;
—tolerance (or convergence) value used to complete the overall decomposition iterative process;
—tolerance values used in the lower levels of resolution;
—number of resolution levels leveraged for the multiresolution approach;
—rank, r, of the decomposition;
—number of nonzeros in lower resolution tensors;
—number modes for which there are external clustering encoded as hierarchies;
—different permutation orders in multiresolution tensor encoding phase;
—different factor matrices expansion functions.

We evaluate the decomposition schemes against two key performance parameters: CPU time and quality. For what concerns the PARAFAC models, we define the quality as [Bader et al. 2012]:

$$1 - \frac{\sqrt{\|\mathcal{X}\|^2 + \|\hat{\mathcal{X}}\|^2 - 2innerproduct(\mathcal{X}, \hat{\mathcal{X}})}}{\|\mathcal{X}\|},$$

Table I. Properties of the Datasets

Name	Type	Auth.	Conf.	Keyw.	Entries	Dens.
dblp128	dense	128	500	500	59K	0.186%
dblp256	dense	256	500	500	101K	0.158%
dblp512	dense	512	500	500	166K	0.130%
dblp2200	sparse	2.2K	1K	14K	927K	0.003%
dblp6600	sparse	6.6K	1K	14K	1.9M	0.002%
dblp20000	sparse	20K	1K	14K	3.5M	0.001%

where $\hat{\mathcal{X}}$ is the tensor obtained by recomposing $\mathcal{F}$. The quality values with Tucker decompositions are measured as [Bader et al. 2012]:

$$1 - \frac{\sqrt{\|\mathcal{X}\|^2 - \|\mathcal{Y}\|^2}}{\|\mathcal{X}\|},$$

where $\mathcal{Y}$ is the core tensor. The CPU time does not include the time to obtain different resolution versions of the input tensor. This aspect will be discussed in Section 5.5.

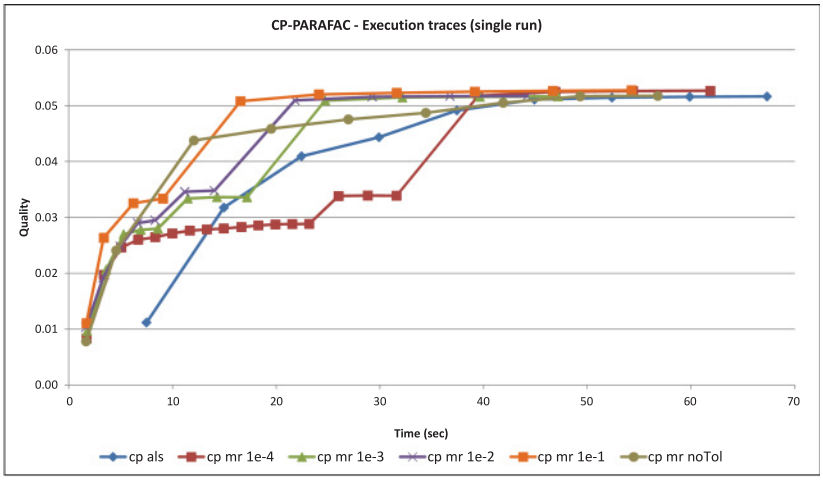
5.1.2. Datasets. In the experiments reported in this section, we used a set of 3-mode tensors of the form, *author* × *conference* × *keyword*, extracted from the publicly available DBLP data [Ley 2009].[2] In particular, we selected the most prolific authors, the most used keywords, and the most popular conferences to construct data tensors with different properties. As mentioned earlier, we have considered both dense and sparse tensors. Table I presents the relevant properties of these datasets.

5.1.3. Metadata. For the three modes of the data tensor, we have considered the following hierarchies:

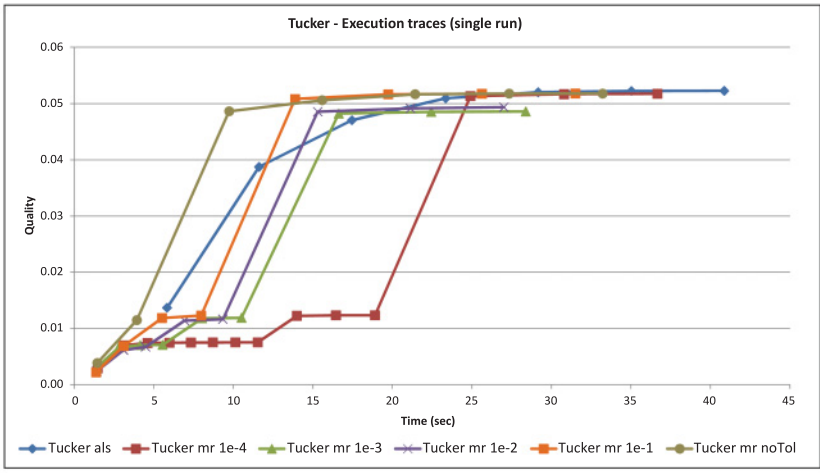
—*Conference Hierarchy*: We have obtained the hierarchy of conferences from http://en.wikipedia.org/wiki/List_of_computer_science_conferences. Conferences are first grouped in 14 main areas and then in 27 subcategories; finally, conferences are represented by the leaves of the hierarchy.
—*Authors Hierarchy*: The hierarchy of authors is obtained by leveraging the co-author relationships among the authors (which is also available at DBLP). In particular, the hierarchy is obtained by iteratively partitioning the coauthors graph using the Metis [Karypis and Kumar 1998] graph partitioning algorithm. We computed seven levels of resolution for the dblp128 and dblp2200 datasets, eight levels for dblp256 and dblp660 datasets, and nine levels for dblp512 and dblp20000 datasets (to obtain these levels, we used an average branching factor of 2 for dense tensors, while for sparse datasets the average branching factor is set to 3).
—*Keyword Hierarchy*: The keyword hierarchy is obtained by first creating a similarity graph based on the set of articles containing at least one occurrence of a given pair of keywords. The keywords hierarchy is then obtained by iteratively partitioning the graph using Metis (we computed nine levels of resolution, with a branching factor of 2).

We note that these hierarchies provide only unimodal information. Note also that the domain hierarchies are *chosen intentionally* rough and noisy. Furthermore, they reflect different degrees of externality: the conference hierarchy is completely external to the DBLP dataset, the author hierarchy is from DBLP, but not directly related to the data in the tensor, whereas the keyword hierarchy is created using document information,

[2]The DBLP Computer Science Bibliography. Available at http://www.informatik.uni-trier.de/~ley/db/.



(a) PARAFAC trace



(b) Tucker trace

Fig. 9. Different execution traces for `dblp512` dataset (dense representation) for both (a) PARAFAC and (b) Tucker decomposition models, where the number of factors is set to 5. Each plot includes standard ALS (with threshold 10^{-4}) and multiresolution approaches (using the last three levels of the authors hierarchy) for different *intermediate* tolerance values (10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, and *noTol*)—the final tolerance value for the highest resolution level is 10^{-4} for all cases.

which (while not being directly represented in the tensor) relates the authors and the keywords in the DBLP dataset.

5.2. Analysis of Sample Execution Traces

Before we provide a detailed analysis of the different problem and system parameters, we first provide sample execution traces to help observe the key advantages of the multiresolution methods against the traditional ALS-based tensor decompositions. We provide comparisons for both Parafac and Tucker models.

The plots in Figure 9 show sample quality-time execution traces for the standard ALS (`cp-als`) and the proposed multiresolution approaches (using the deepest three levels of the authors hierarchy). Each plot includes five different multiresolution plots:

—mr1e-4 corresponds to the case where the convergence threshold 10^{-4} is used for all datasets,
—mr1e-3 is the case where a less precise, 10^{-3}, threshold is used as a stopping condition in the intermediary resolutions, but 10^{-4} is used in the final step, corresponding to the highest resolution,
—mr1e-2 represents a rougher, 10^{-2} threshold, in the intermediary steps and 10^{-4} in the final step.
—mr1e-1 represents a rougher, 10^{-1} threshold, in the intermediary steps and 10^{-4} in the final step, and
—mr noTol is the case where the algorithm performs only one iteration in the intermediary steps and uses a convergence threshold of 10^{-4} in the final step.

For the ALS approach, the convergence threshold is 10^{-4} (i.e., the same as the final step of the multiresolution cases).

PARAFAC traces. Let us first consider Figure 9(a), which presents the Parafac decomposition for the dblp512 dataset (dense representation). Each marker on the curves corresponds to the end of an iteration and the beginning of the next one:

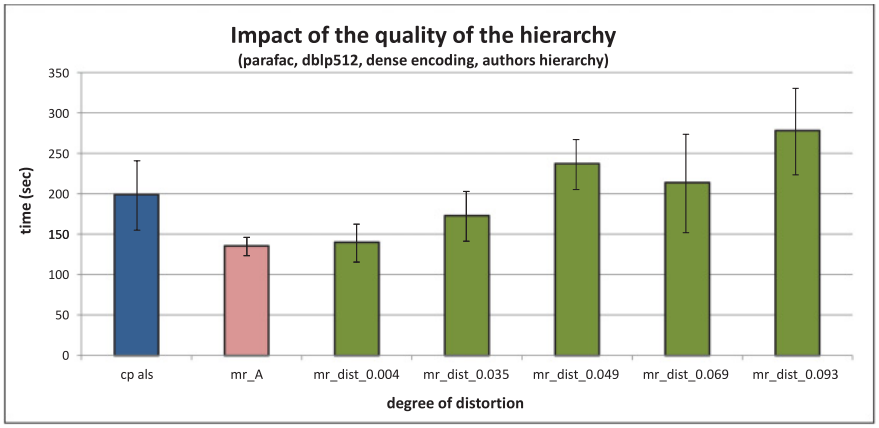
—As can be seen here, ALS PARAFAC provides most of its improvements in its first few iterations and a significant portion of the overall time is spent seeking the target per-iteration quality difference.
—In the case of multiresolution traces, there are more iterations: the algorithm performs a separate sequence of iterations for each selected level of the hierarchy, but each iteration is faster. Moreover, in deeper levels (with higher resolution, thus costlier individual steps) much fewer iterations are necessary to reach the convergence threshold.

Note that convergence thresholds at the intermediary resolutions have significant impacts on the shape of the execution trace of the multiresolution approaches. In particular, requiring the same threshold 10^{-4} as the high-resolution step at each level of the hierarchy may cause the multiresolution approach to spend (unnecessarily) more time for obtaining lower-resolution decompositions. Relaxing the threshold down reduces the time spent for decomposing the lower resolutions and may also help reduce overall decomposition time. In the given example, the mr1e-2 case, with the relaxed intermediary convergence thresholds of 10^{-2} provides the quickest convergence. Further relaxing the intermediary convergence thresholds may help reduce the time taken during the approach phase of the algorithm but may also negatively impact the overall convergence time.

Naturally, the benefits of the proposed scheme depends on the quality of the input hierarchy used as metadata to support the multiresolution approach.

In Figure 10, we study the impact of the quality of the hierarchy on the performance of the multiresolution approach. Here, we have injected different degrees of distortions in the authors hierarchy by swapping hierarchy nodes (for each configuration, we present the average of 30 random distortions). As we can see here, the general trend is that the higher the amount of distortion (measured in terms of the discordant hierarchy branches) the worse the multiresolution performance. This confirms that the proposed metadata supported tensor decomposition scheme indeed leverages the information provided by the given hierarchy: if the hierarchy is informative, then the multiresolution scheme is effective.

Tucker traces. Figure 9(b) reports sample execution traces comparison between the standard Tucker decomposition and the proposed multiresolution approach: the Tucker als curve depicts the quality-time behavior of the standard ALS HOOI algorithm. When



Quality			
$cpals$ 0.118	mr_A 0.116	$mr\_dist_{0.004}$ 0.115	$mr\_dist_{0.035}$ 0.112
$mr\_dist_{0.049}$ 0.116	$mr\_dist_{0.069}$ 0.112	$mr\_dist_{0.093}$ 0.111	

Fig. 10. Impact of the quality of the hierarchy used in the multiresolution process (parafac, dense matrices, `dblp512` data, $r_i = 20$, 10^{-4} tolerance for all levels, three levels of resolution used, authors hierarchy, the degree of the distortion is expressed as the ratio of number of discordant branches with respect to the total number of branches).

comparing with the PARAFAC traces in Figure 9(a), we see that, in the case of Tucker decomposition, the contribution to the quality of the intermediate levels is less than that obtained in the PARAFAC scenario (in fact, the *noTol* approach which does not perform iterations in the intermediary levels improves the quality of the approximation the fastest). Yet, the multiresolution approach still provides overall execution time gains. This indicates that multiresolution approach is able to leverage available external unimodel hierarchy to speed up the decomposition process, without negatively affecting the final quality. It also hints to the fact that the intermediary decomposition steps need not to be perfect to have a good overall decomposition performance.

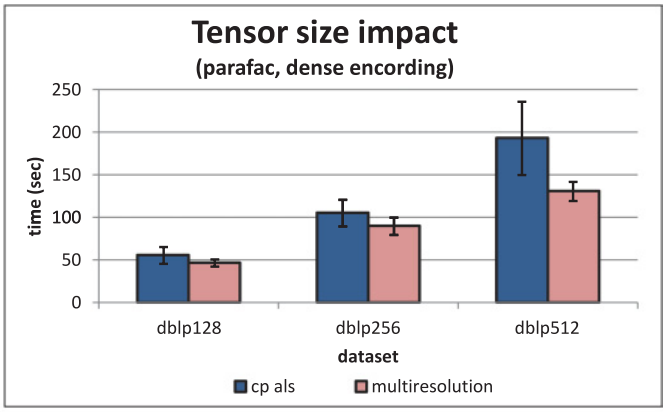
Note that the traces in Figure 9 correspond to only a single dataset. Therefore, while they provide rough hints about the impact of multiresolution approach and thresholds, we need more detailed analysis, including multiple datasets and configurations, for more concrete conclusions. We present these detailed analyses next.

5.3. Detailed Analysis: PARAFAC

In this section, we study the performance of the proposed scheme for PARAFAC decomposition under different problem and system parameters to assess whether the trends observed in the previous sample scenario extend to different situations.

5.3.1. Experiments with Tensors in Dense Form. We first consider decomposition of tensor matrices represented in dense form.

Tensor size. Figure 11 presents comparisons of the multiresolution algorithm with standard ALS Parafac for different tensor sizes. In these experiments, the tolerance is set to 10^{-4} in ALS as well as for multiresolution scheme (at intermediate levels as well as for the highest resolution).



Quality			
	dblp128	dblp256	dblp512
cp als	0.180	0.143	0.119
multires	0.181	0.143	0.117

Fig. 11. Impact of the tensor size (PARAFAC, dense matrices, $r = 20$, only author hierarchy, 10^{-4} threshold; also for multiresolution only the deepest three levels used; tolerance is set to 10^{-4} for all resolutions).

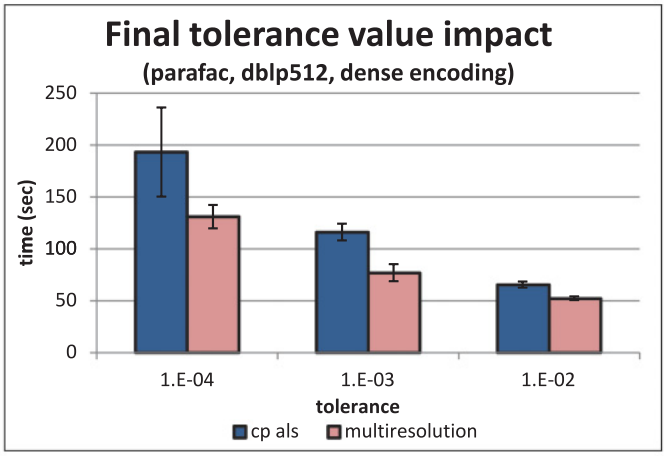
—As the figure shows, the multiresolution approach provides gains in execution time for all sizes and the gains are especially larger for larger tensors.
—The table in the figure also shows that the final quality of the multiresolution scheme matches the quality of the more traditional ALS scheme for all tensor sizes.

Final convergence threshold (i.e., tolerance). In the previous experiment, the stopping condition (tolerance) was set to 10^{-4} for ALS as well as for the multiresolution scheme. Figure 12 reports the results for the scenarios where the overall convergence tolerances range from 10^{-2} to 10^{-4}.

—As the figure shows, the gain in execution time is especially pronounced when the stopping condition is tighter; this is because ALS needs to spend a significant amount of time slowly approaching the convergence condition whereas the multiresolution approach converges faster.
—It is important to note, from the quality table, that the traditional ALS quality drops significantly (from 0.119 to 0.108) when the convergence threshold is relaxed, whereas it is much more stable if the proposed multiresolution scheme is used (from 0.117 to 0.115).

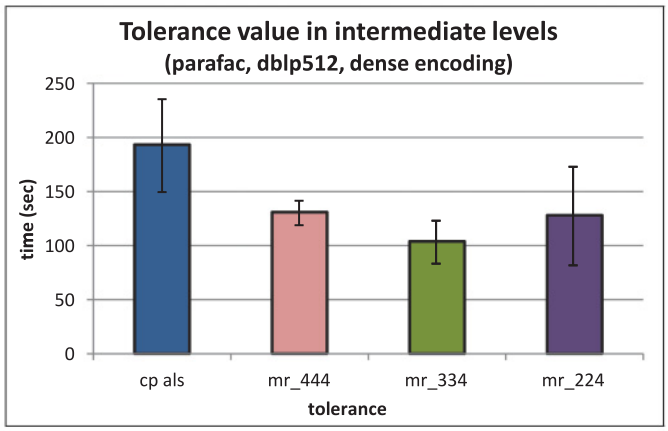
Convergence thresholds (i.e., tolerance) for intermediate levels. Figure 13 depicts the impact of using different convergence tolerances at intermediate levels of the multiresolution scheme (this figure provides a summary of the traces shown in Figure 9). Here, for multiresolution approaches, the final tolerance value is set to 10^{-4}, whereas the tolerance values at the intermediate levels are varied.

—We observe that the fastest execution is obtained when the thresholds at the intermediate resolutions are neither too tight, nor too lax.
—In terms of quality, as we have also seen in Figure 9, a relatively more lax intermediary threshold in fact helps the multiresolution scheme avoid getting stuck at local optima, leading to better overall decomposition quality.



Quality			
	10^{-4}	10^{-3}	10^{-2}
cp als	0.119	0.118	0.108
multires	0.117	0.118	0.115

Fig. 12. Impact of the final tolerance; that is, stopping condition (PARAFAC, dense matrices, $r = 20$, only author hierarchy, `dblp512`, deepest three levels of hierarchy).

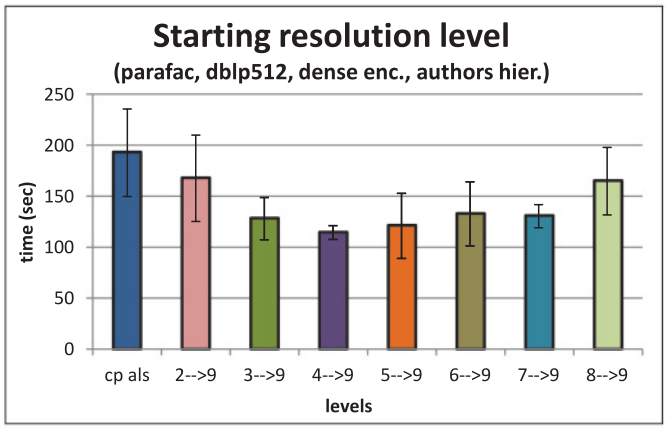


Quality			
cp als	mr 10^{-4}	mr 10^{-3}	mr 10^{-2}
0.119	0.117	0.116	0.118

Fig. 13. Impact of tolerance values in the intermediate levels of resolution (PARAFAC, dense matrices, $r = 20$, only author hierarchy, `dblp512` data, deepest three levels of hierarchy).

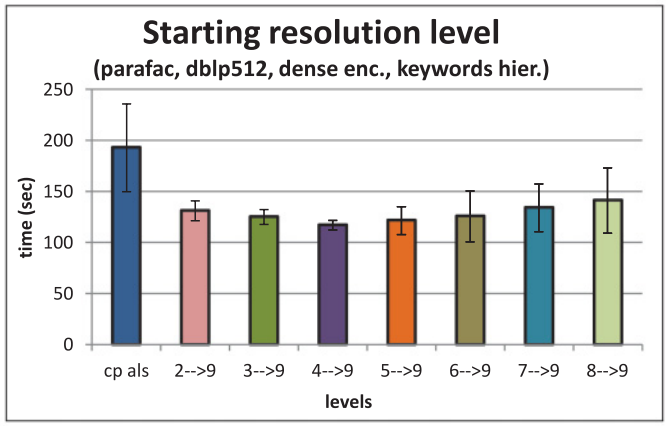
Resolution levels. Figures 14 and 15 show the impact of the number of resolution levels used in the multiresolution approach with the authors and keywords hierarchies.

—As shown in the figures, the number of hierarchy levels used impacts the execution times and the largest gains in execution time are obtained when the process starts at mid-height of the hierarchy: starting at levels closer to root (e.g., level 2 in the figures) causes the multiresolution algorithm to spend time unnecessary searching



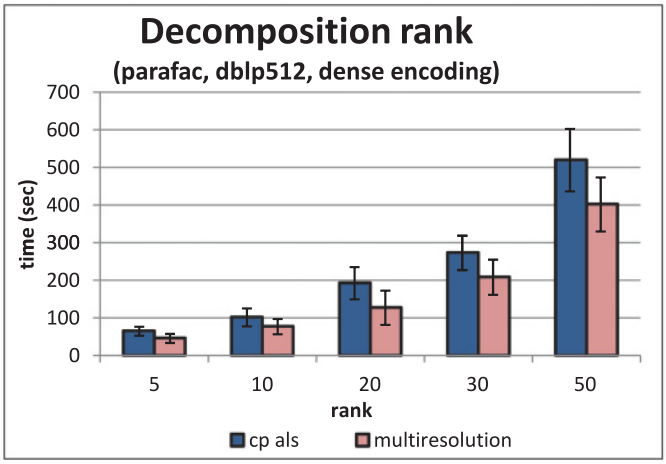
Quality				
cp als	$2 \to 9$	$3 \to 9$	$4 \to 9$	$5 \to 9$
0.119	0.118	0.118	0.117	0.118
	$6 \to 9$	$7 \to 9$	$8 \to 9$	
	0.117	0.117	0.119	

Fig. 14. Impact of the starting resolution level (PARAFAC, dense matrices, $r = 20$, only author hierarchy, `dblp512` data, tolerance 10^{-4} at all levels).



Quality				
cp als	$2 \to 9$	$3 \to 9$	$4 \to 9$	$5 \to 9$
0.119	0.120	0.121	0.121	0.120
	$6 \to 9$	$7 \to 9$	$8 \to 9$	
	0.120	0.120	0.119	

Fig. 15. Impact of the starting resolution level (PARAFAC, dense matrices, $r = 20$, only keyword hierarchy, `dblp512` data, tolerance 10^{-4} at all levels).



Quality					
	5	10	20	30	50
cp als	0.052	0.081	0.119	0.144	0.179
multires	0.050	0.079	0.118	0.141	0.178

Fig. 16. Impact of the rank, r, of the decomposition (PARAFAC, dense matrices, $r = 5, 10, 20, 30, 50$, only author hierarchy, `dblp512` data, tolerance 10^{-4} at all levels, deepest three levels of hierarchy).

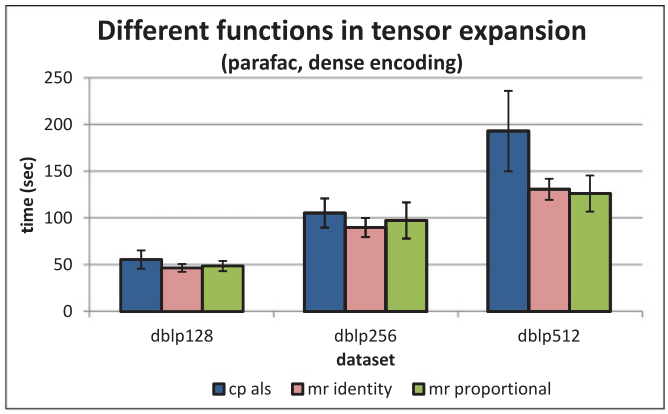
for decompositions that are too imprecise to help effectively bootstrap lower levels; starting at levels too close to the leaves (e.g., 8 in the examples), on the other hand, prevents the multiresolution scheme to properly leverage the available metadata.

—Note that, while the absolute values of the gains differ, both author and keyword hierarchies show the same overall behavior with respect to the starting resolution level, confirming the previous observation.

Target rank of the decomposition. Figure 16 shows that the proposed multiresolution approach consistently outperforms the traditional ALS-based decomposition for all target decomposition ranks. The time gain ranges from $\sim$20% to $\sim$30%. The multiresolution approach also closely follows the quality of ALS at all ranks.

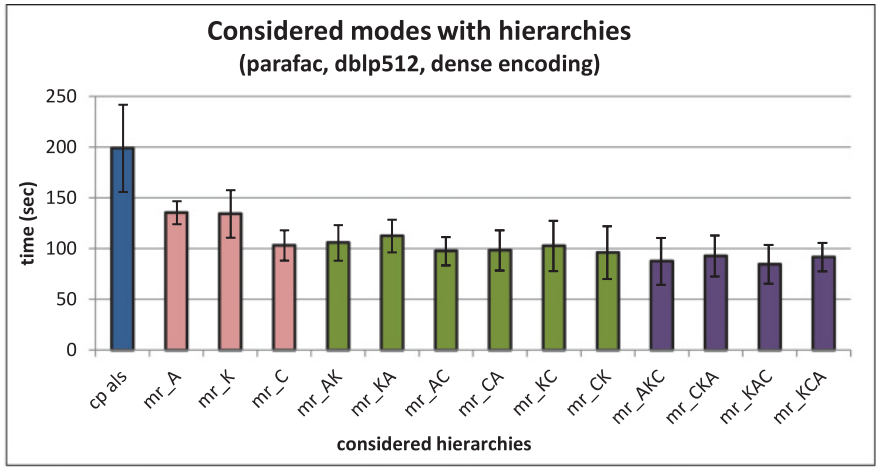
Tensor expansion strategy. Remember from Section 3 that the $expand\_factor()$ function allows to expand the factor matrices from the lower level of resolution $resol - 1$ to the next level $resol$ and that we introduced two different expansion functions: the (intuitive) identity function and an expansion function based on a *proportional* strategy. All previously reported experiments are based on the identity function. In Figure 17, however, we compare the two expansion strategies. As can be seen here, the two approaches provide similar behaviors in terms of execution time and final decomposition quality. Therefore, in the rest of the article, we continue using the identity function as the default expansion function (unless otherwise specified).

Number of hierarchies used for multiresolution process. Figure 18 shows that different domain hierarchies impact the multiresolution process differently. We measured the influence of both number of hierarchies and their order used in the encoding phase. As can be seen, except for the use of the conference hierarchy, the more hierarchies are used, the faster the overall decomposition becomes. If we consider the experiments with only one hierarchy (the pink bars), it is interesting to note that, in



Quality			
	dblp128	dblp256	dblp512
cp als	0.180	0.143	0.119
mr identity	0.181	0.143	0.117
mr proportional	0.179	0.141	0.118

Fig. 17. Comparison of different strategies in tensor expansion (PARAFAC, dense matrices, $r = 20$, only author hierarchy, 10^{-4} threshold; deepest three levels used).



Quality						
cp_als	mr_A	mr_K	mr_C	mr_{AK}	mr_{KA}	mr_{AC}
0.118	0.116	0.120	0.116	0.120	0.120	0.116
mr_{CA}	mr_{KC}	mr_{CK}	mr_{AKC}	mr_{CKA}	mr_{KAC}	mr_{KCA}
0.115	0.116	0.117	0.116	0.116	0.116	0.115

Fig. 18. Impact of the number and orders of hierarchies considered in multiresolution process (PARAFAC, dense matrices, $r = 20$, `dblp512` data, tolerance 10^{-4} at all levels, deepest three levels of hierarchy). The ordering of the considered hierarchies associated to tensor modes is represented by the sequence of capital letters in the chart and in the table: (A)uthors, (C)onferences, and (K)eywords.

Table II. Tolerance Values Used in Experiments on Sparse Tensors
In *mr_noTol* there is not a stopping condition in the two lowest levels of resolution; only one iteration is computed.

	Tolerance values (i.e., stopping conditions)		
	leaf - 2	leaf - 1	leaf (highest res.)
*mr*444	10^{-4}	10^{-4}	10^{-4}
*mr*334	10^{-3}	10^{-3}	10^{-4}
*mr*224	10^{-2}	10^{-2}	10^{-4}
*mr*114	10^{-1}	10^{-1}	10^{-4}
mr_noTol	Only one iteration	Only one iteration	10^{-4}

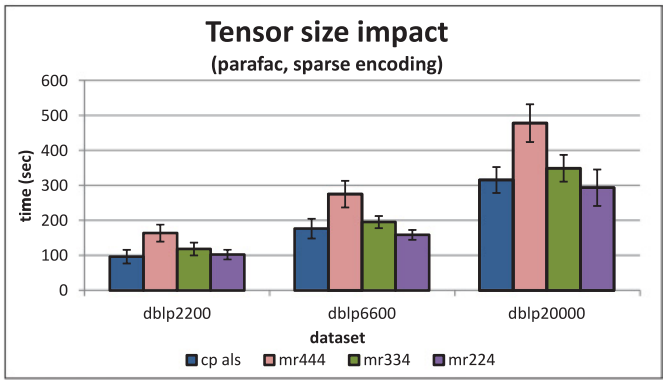
this case, with the keyword hierarchy, the multiresolution approach outperforms the basic ALS algorithm both in overall quality and execution time, while the use of the conference hierarchy ensures the best execution time. The green and violet bars report the impact of using two and three hierarchies: as can be seen here, the execution time decreases when the number of hierarchies increases. Moreover, the decomposition quality is minimally affected by the number of considered hierarchies: the combined use of the author hierarchy and the keyword hierarchy provides the best overall quality.

Finally, remember from Section 3.1 that a clustering embedding strategy is defined through π, a permutation over the indices of the tensor modes describing the order in which clusterings are considered among different modes. Figure 18 also compares the impact of different considered permutation orders. As can be seen, different permutation orders do not have any significant impact on the execution times or the final quality values. This is intuitive in that in these experiments we have used the *average* function, which is associative, as the representative selection function during multiresolution tensor encoding.

5.3.2. Experiments with Tensors in Sparse Form. So far, we have reported experiment results on tensor represented in dense form. However, in many datasets of interest (such as social media graphs) tensors can be sparse and these can leverage tensor decomposition tools specifically designed to work for tensors with sparse encodings. Therefore, in this section, we also evaluate the multiresolution approach on datasets with sparse representations, as described in Table I. In these experiments, we set the target rank r to 20 and the tolerance value for high-resolution decomposition is set to 10^{-4}. For all datasets, we have considered the deepest three levels of the hierarchy and varied the thresholds for the different levels as reported in Table II.

First, the bad news. As shown in Figure 19, when the tensor is sparse enough to rely on sparse tensor representations, the multiresolution approach does not necessarily provide time gains. Time gains are observed only when relatively lax tolerances (10^{-2}) are used at intermediate resolutions and, even then, gains are modest. In fact, the execution time performance of ALS is matched only when the intermediary levels are not iterated until a tolerance level is reached, but executed only once (*mr_noTol*). This is because, when the tensors are represented in sparse form, decomposition cost tends to be a function of the number of nonzero entries [Bader and Kolda 2007]. Unfortunately, as reported in Table III, drops in tensor resolutions do not necessarily imply significant reductions in the number of tensor nonzero entries and, therefore, time gains in the high-resolution level do not always amortize the overhead incurred by at the lower resolutions. As Figure 20 shows, the performance degradation is especially significant for the conference hierarchy, which is completely external to the dataset.

Then, the good news. While this initially looks very disappointing, we remember from the earlier experiments that *relaxing the precision at the lower resolutions by using a more relaxed convergence threshold could provide big boosts in execution time, without*



Quality			
	dblp2200	dblp6600	dblp20000
cp als	0.055	0.042	0.035
mr444	0.053	0.042	0.034
mr334	0.053	0.042	0.034
mr224	0.053	0.042	0.034

Fig. 19. Impact of the tensor size (PARAFAC, sparse matrices, $r = 20$, only author hierarchy, 10^{-4} final tolerance; deepest three levels used).

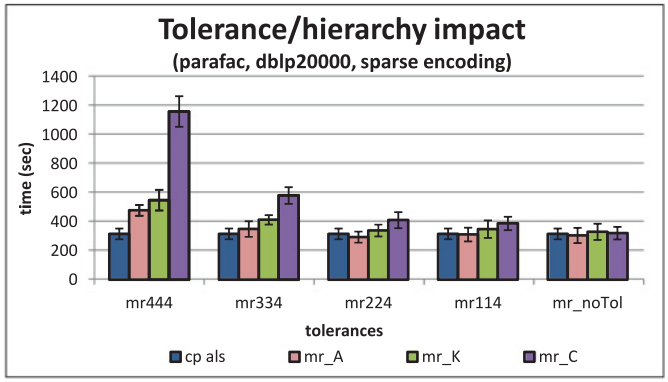
Table III. Number of Nonzero Entries at Different Levels of Resolution

Number of nonzero entries (for dblp2200)		
Leaf-2 (lowest res.)	Leaf-1	Leaf (highest res.)
802,032	853,138	927,663
Number of nonzero entries (for dblp6600)		
Leaf-2	Leaf-1	Leaf
1,572,440	1,685,556	1,862,769
Number of nonzero entries (for dblp20000)		
Leaf-2	Leaf-1	Leaf
2,819,233	3,060,544	3,463,212

negatively affecting the final quality. This means that we might be able to eliminate the bottleneck by relaxing the number of nonzero entries considered in the lower-resolution steps. To achieve this effect, we introduced cutoffs, θ_{nz}, for nonleaf levels and eliminated all entries smaller than θ_{nz} from the tensors, creating less precise approximations of the original tensor at low resolutions. Table III shows the parameters used in the experiments. Note that we have experimented with very significant reductions in the number of nonzero entries. Figure 21 shows the performance results for these θ-cutoffs:

—As expected the more nonzero entries are removed in the low-resolution steps, the faster becomes the decomposition, providing up to ∼40% gains in execution time.
—Most interestingly, the θ_{nz}-cutoff based nonzero removal has close to zero impact on the quality of the final decomposition, rendering the proposed multiresolution approach very applicable for sparse tensors as well.

Note that, as discussed in Section 5.5, the proposed metadata supported multi-resolution scheme has an additional tensor encoding cost, which may become a bottleneck for sparse tensors if it is carried out during the decomposition step. As we discuss in Section 5.5, the multiresolution tensor encoding of sparse tensors is advantageous



Quality			
$cpals$	0.035		
	Authors	Keywords	Conferences
$mr444$	0.034	0.035	0.030
$mr334$	0.034	0.035	0.033
$mr224$	0.034	0.035	0.034
$mr114$	0.034	0.035	0.034
mr_noTol	0.034	0.035	0.034

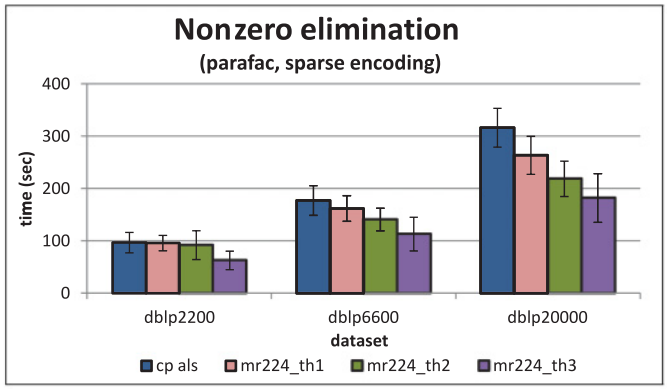
Fig. 20. Impact of the tolerance value and tensor modes with hierarchies (PARAFAC, sparse matrices, $dblp20000$ data, $r = 20$, 10^{-4} final tolerance; deepest three levels used).

Table IV. Number of Nonzero Entries, with θ-Cutoff, at Different Levels of Resolution (Sparse Tensors)

Number of nonzero entries with θ-cutoff (for dblp2200)						
Leaf-2			Leaf-1			Leaf
Initial	θ_1	Final	Initial	θ_1	Final	No cutoff
802,032	0.096	613,319	853,138	0.230	731,600	927,663
	0.109	540,300		0.229	507,519	
	0.132	226,074		0.238	186,379	
Number of nonzero entries with θ-cutoff (for dblp6600)						
Leaf-2			Leaf-1			Leaf
Initial	θ_2	Final	Initial	θ_2	Final	No cutoff
1,572,440	0.096	1,237,419	1,685,556	0.230	1,406,544	1,862,769
	0.109	1,083,064		0.310	1,001,005	
	0.132	460,251		0.450	340,733	
Number of nonzero entries with θ-cutoff (for dblp20000)						
Leaf-2			Leaf-1			Leaf
Initial	θ_3	Final	Initial	θ_3	Final	No cutoff
2,819,233	0.096	2,156,319	3,060,544	0.230	2,461,299	3,463,212
	0.109	1,852,138		0.310	1,697,666	
	0.162	827,287		0.380	705,163	

only if it can be amortized over multiple decomposition tasks (different subsets of modes and ranks).

Can nonzero removal also be used for dense tensors? Since we have seen that nonzero removal can be used to reduce the cost of multiresolution tensor decoding under sparse representations, we further investigate whether a similar strategy could



Quality				
	cutoff	dblp2200	dblp6600	dblp20000
cp als		0.055	0.042	0.035
$mr224$	θ_1	0.053	0.042	0.034
$mr224$	θ_2	0.053	0.042	0.034
$mr224$	θ_3	0.053	0.042	0.034

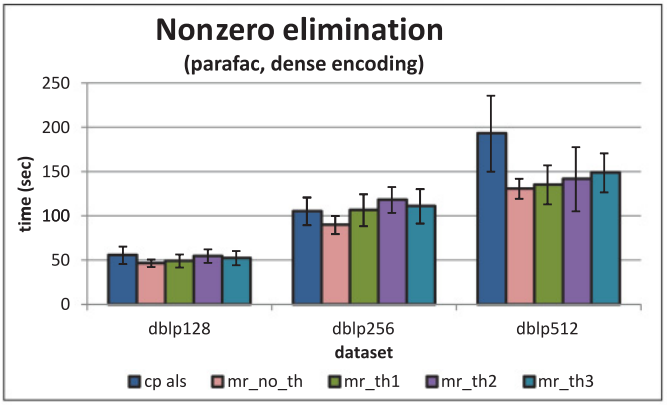
Fig. 21. Impact of the tensor size with θ-cutoff elimination of nonzero entries (PARAFAC, sparse tensors, $r = 20$, only author hierarchy, 10^{-4} tolerance; deepest three levels used). See Table IV for details of θ-cutoff values.

Table V. Number of Nonzero Entries, with θ-Cutoff, at Different Levels of Resolution (Dense Tensors)

Number of nonzero entries with θ-cutoff (for dblp128)						
Leaf-2			Leaf-1			Leaf
Initial	θ_1	Final	Initial	θ_1	Final	No cutoff
53,899	0.170	45,399	55,160	0.260	43,531	59,602
	0.260	16,999		0.340	34,951	
	0.420	11,302		0.510	11,469	
Number of nonzero entries with θ-cutoff (for dblp256)						
Leaf-2			Leaf-1			Leaf
Initial	θ_2	Final	Initial	θ_2	Final	No cutoff
91,642	0.170	65,813	93,998	0.260	73,574	101,398
	0.300	28,452		0.380	45,910	
	0.460	19,359		0.510	18,490	
Number of nonzero entries with θ-cutoff (for dblp512)						
Leaf-2			Leaf-1			Leaf
Initial	θ_3	Final	Initial	θ_3	Final	No cutoff
151,012	0.170	118,319	154,135	0.260	128,209	166,678
	0.260	53,395		0.410	78,474	
	0.460	30,575		0.610	29,564	

also be useful in the case of tensors with dense representations. Table V reports the cutoff values used in this set of experiments, while Figure 22 shows the corresponding time and quality results for dense tensors:

—As can be seen here, in the case of dense tensors, removal of nonzero values from low-level tensors results in, not a decrease in execution time, but a slight increase (up to ∼8% for *dblp*512) with respect to the *pure* multiresolution approach (without



Quality				
	cutoff	dblp128	dblp256	dblp512
cp als		0.180	0.143	0.119
$mr444$	no threshold	0.181	0.143	0.117
$mr444$	θ_1	0.180	0.141	0.117
$mr444$	θ_2	0.179	0.139	0.117
$mr444$	θ_3	0.176	0.138	0.115

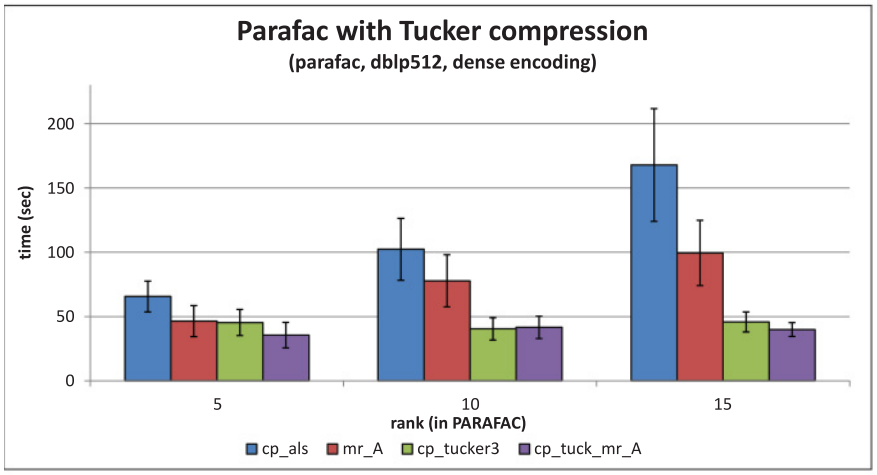
Fig. 22. Impact of the tensor size with θ-cutoff elimination of nonzero entries (PARAFAC, dense tensors, $r = 20$, only author hierarchy, 10^{-4} tolerance; deepest three levels used). See Table V for details of θ-cutoff values.

nonzero removal). This is because, in the case of tensors with dense representation, the execution time is not a function of the nonzero values, but the overall tensor dimensions and, thus, decomposer cannot leverage the increase in the number of zeros to reduce the execution time.

—Instead, it appears that an increase in the number of zeros renders the search algorithm take longer time in approaching to a sufficiently close approximation: removal of nonzero values at lower levels causes poor tensor approximations that negatively influence the internal decomposition steps.

Thus, we recommend against using nonzero removal for dense tensors and suggest that it is an effective tool only for tensors that are sparse.

5.3.3. PARAFAC with Tucker Support. As discussed in Section 2.2, Bro and Andersson [1998] proposed using Tucker decomposition as a first step toward PARAFAC decomposition. In this scheme, first a Tucker decomposition is obtained and then this decomposition is used to bootstrap the PARAFAC iterations. The initial Tucker decomposition can be obtained either using an eigen-decomposition of a single mode (referred to as Tucker1) or through a full Tucker decomposition across all available modes (referred to as Tucker3). This approach to PARAFAC decomposition is applicable when the input tensors are dense, as the initial Tucker decomposition of the tensors with sparse representations would result in dense structures in intermediary steps of Tucker [Kolda and Sun 2008] causing memory blowup problems. We see that our proposed multiresolution approach to PARAFAC avoids this memory problem of [Kolda and Sun 2008] when decomposing sparse tensors, as it does not involve Tucker. On the other hand, when the input tensors are dense, using Tucker decomposition as an initial step to PARAFAC can provide significant time gains. We note, however,



Quality			
	5	10	15
cp_als	0.052	0.081	0.103
mr_a	0.050	0.079	0.101
cp_tucker	0.045	0.077	0.100
cp_tuck_mr_a	0.046	0.072	0.097

Fig. 23. Impact of the Tucker compression in PARAFAC fitting models (PARAFAC, dense tensors, only author hierarchy, 10^{-4} final tolerance; no tolerance values in the intermediate levels; deepest three levels used; the number of factors in Tucker compression is set as the rank in PARAFAC, for all tensor modes).

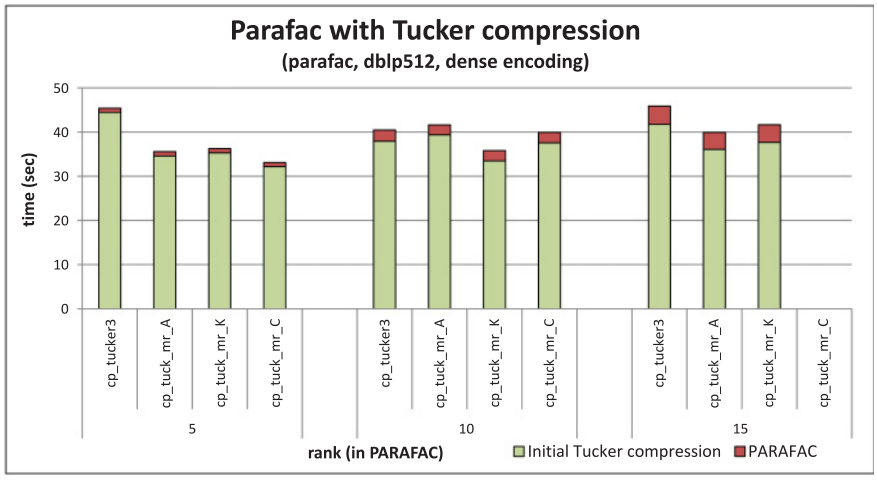
that our proposed metadata supported multiresolution approach can also be used for improving the execution time of this initial Tucker decomposition step.

This is confirmed in Figure 23. Here, we see that Tucker3 compression (*cp_tucker*3) provides significant execution time savings relative to basic ALS approach (*cp_als*) and it performs faster than the basic multiresolution technique (*mr_A*). Moreover, when we replace the initial Tucker decomposition step, with our multiresolution Tucker algorithm (*cp_tuck_mr_A*), we obtain the best or very close to best results in terms of the execution time. Figure 24 further confirms that the gains in the proposed solution are primarily due to the execution time reductions in the Tucker decomposition step that precedes the PARAFAC operation.

5.4. Detailed Analysis: Tucker

In this section, we report results obtained with the proposed multiresolution approach in the case of fitting Tucker models. Note that we consider Tucker decompositions of dense tensors: the reason for this is that decomposition of a tensor with sparse representation produces dense structures in the intermediate steps, resulting in the well-known memory blowup problem [Kolda and Sun 2008]. For the datasets we consider, this implied that the Tucker decomposition under sparse representation was not feasible with the available memory (4GB).

Initialization methods. To ensure a high degree of independence of the decomposition process with respect to the factor matrices initial guess, in all previous experiments, we used a random process as the default initialization method. We now show how different initialization techniques of the factor matrices $\mathbf{V}^{(i)}$ introduced in Section 4.1 can influence the final results of the standard ALS HOOIn algorithm. Table VI compares



Quality			
	5	10	15
cp_tucker	0.045	0.077	0.100
cp_tuck_mr_a	0.046	0.072	0.097
cp_tuck_mr_k	0.046	0.076	0.098
cp_tuck_mr_c	0.044	0.072	n.a.

Fig. 24. Comparison of different Tucker compression techniques (standard HOOI and multiresolution) in PARAFAC fitting models (PARAFAC, dense tensors, 10^{-4} final tolerance; no tolerance values in the intermediate levels; deepest three levels used). Note that *cp_tuck_mr_c* is not feasible when the target rank is 15 as the member of conferences since the number of elements in the conference hierarchy at the deepest level used (i.e., 14) is less than the number of required factors in the initial Tucker compression (i.e., 15).

Table VI. Comparison of Different Initialization Methods (Random (r), Hierarchy-based (h), Eigenvector-based (v)) for Standard Tucker HOOI Algorithm (Dense Matrices, `dblp512` Data, Factors (r_A, r_C, r_K): {(4, 27, 4), (8, 27, 8), (16, 27, 16), (32, 27, 32), (64, 27, 64), (128, 27, 126), (199, 27, 186)}; 10^{-3} Final Tolerance Value)

		Execution time		Quality	
Initialization method		Average	Stand. deviation	Average	Stand. deviation
Pure	rrR	48.7	0.440	0.1428	6.59E-04
rendom	rRr	41.9	5.070	0.1432	6.39E-04
	Rrr	47.6	2.698	0.1433	4.99E-04
Hierarchy	hRr	47.7	4.400	0.1428	5.88E-04
based	Rhr	47.0	1.919	0.1433	3.33E-04
	Rrh	45.4	2.426	0.1433	9.31E-05
	hhR	46.7	0.128	0.1427	2.02E-08
	hRh	41.0	0.121	0.1424	1.97E-07
	Rhh	41.6	0.127	0.1435	8.06E-06
Eigenvectors	vRr	49.8	3.181	0.1434	4.87E-04
based	Rvr	58.0	2.562	0.1431	4.08E-04
	Rrv	51.9	0.740	0.1432	1.12E-04
	vvR	55.3	0.098	0.1441	2.09E-08
	vRv	53.2	0.128	0.1439	2.48E-07
	Rvv	58.4	0.165	0.1440	2.27E-08

the average execution times and qualities for different initialization strategies. The averages and standard deviations are obtained by varying the number of factors for each tensor dimension. In this table, an experiment represented by *abc* indicates that the first factor matrix (authors) is initialized with the method *a*, the second factor matrix (conferences) is initially set with the method *b*, and the third factor matrix (keywords) is initialized with method *c*. The mode denoted with the capital letter is the dimension that is considered as the starting dimension in the iterative process. As described in Section 4.1, we have considered three initialization methods: random (*r*), eigenvectors based (*v*), and hierarchy based (*h*). Note that the first mode considered in the iterative process is always initialized using a random method, because at the first iteration the corresponding factor matrix is updated starting from all the other factor matrices. As can be seen, as expected, the highest quality is provided by the eigenvector based initialization strategy, which has a significant execution time drawback due to the time cost of eigenvector computation. Random initialization strategies result in significant standard deviations in execution time and quality, which makes them undesirable. Hierarchy-based initializations, in general, provide better quality than random initialization approaches, with lower execution times with respect to eigenvectors-based approaches; moreover, the extreme performance variations when using the random initialization strategy are avoided.

In the rest of the experiments, we use the pure random initialization strategy as default.

Convergence thresholds (i.e., tolerance) for intermediate levels. Figure 26 shows the impact of using different tolerance values in the intermediate levels. We vary these thresholds from 10^{-4} to 10^{-1}, and include an additional modality, called *noTol*, in which the intermediary levels are executed only once. The results show that we can reduce the execution times of decomposition by using relatively lax convergence thresholds in the intermediate levels without significantly impacting the overall decomposition quality. While it is not the best one among the considered convergence thresholds, in the rest of this section we will use the *noTol* as the default convergence strategy.

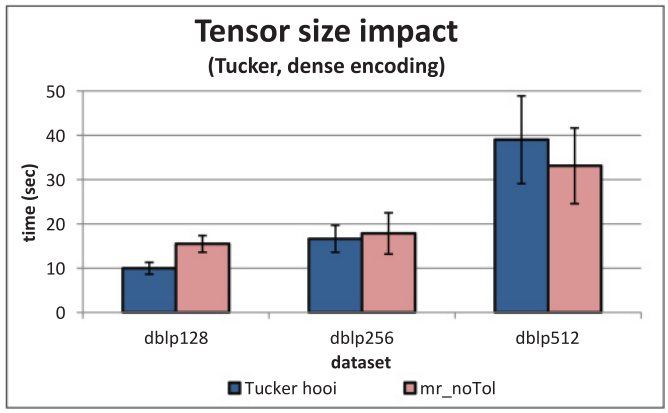
Tensor size. Figure 25 compares the standard Tucker HOOI algorithm with the multiresolution approach proposed in Algorithm 4 for varying tensor sizes:

—As the figure shows, for very small tensors, the basic Tucker HOOI method is more efficient, but the execution time benefits of the multiresolution approach become apparent as the size of the tensor increases.
—Similarly, as the size of the tensor becomes larger, the quality of the multiresolution approach gets closer to the quality provided by the Tucker HOOI method.

Domain hierarchy used for multiresolution process. Figure 26 also shows the impact of the use of different hierarchies in the decomposition process. Note that the impact of the use of a single hierarchy on Tucker decomposition is similar to the case of dense tensor decomposition using PARAFAC, reported in Figure 18; both of them contrast significantly with the sparse tensor decomposition reported in Figure 20. From these results, it appears that the conference hierarchy (which is completely external to the dataset) negatively affects the performance only for sparse tensors.

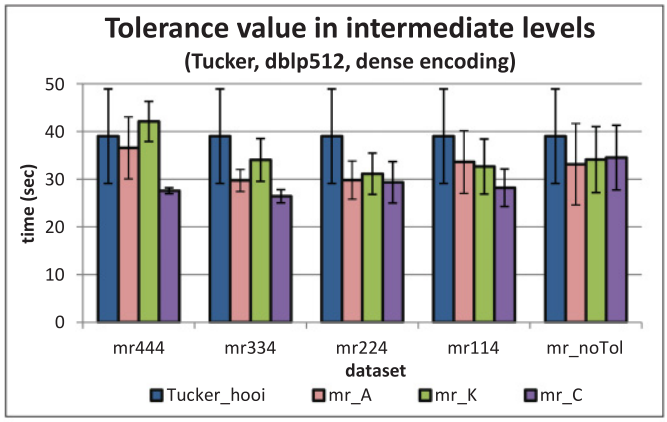
Target rank of the decomposition. Figure 27 shows that the proposed approach scales well with the target rank and is competitive against standard Tucker for a wide range of target ranks.

Number of hierarchies. Figure 28 shows the impact of using more than one hierarchy to support decomposition. It also reports the results about the order of hierarchies in



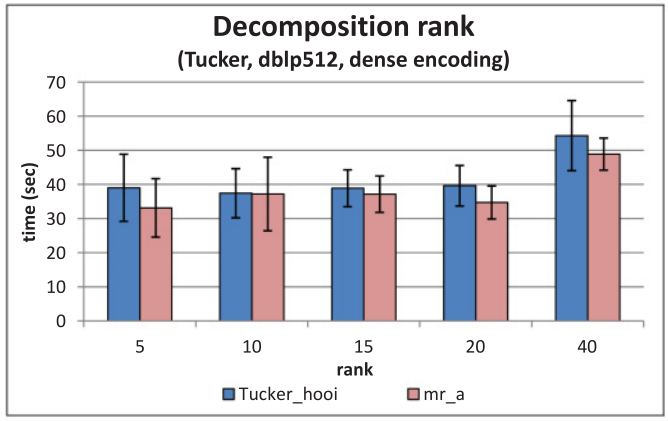
Quality			
	dblp128	dblp256	dblp512
Tucker hooi	0.091	0.068	0.053
mr noTol	0.086	0.064	0.051

Fig. 25. Impact of the tensor size (Tucker, dense matrices, $r = 5$, only author hierarchy, only the deepest three levels used; tolerance is set to 10^{-4} for the highest level, no tolerance value is used in the intermediate levels).



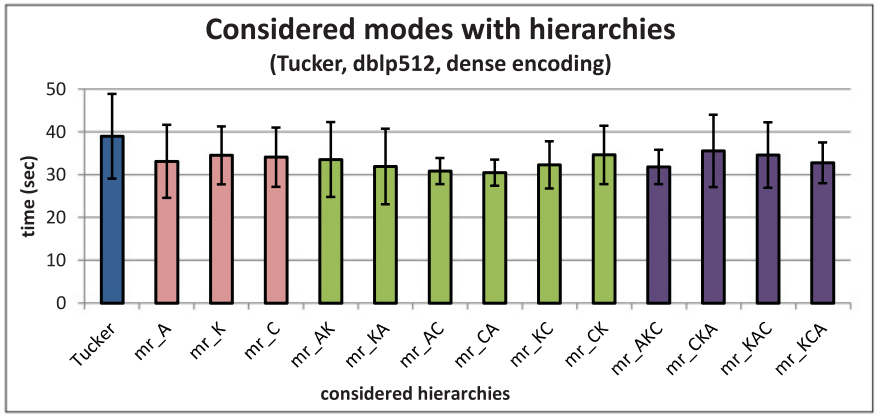
Quality			
$Tuckerhooi$	0.053		
	Authors	Keywords	Conferences
$mr444$	0.052	0.053	0.053
$mr334$	0.051	0.053	0.053
$mr224$	0.051	0.052	0.052
$mr114$	0.051	0.053	0.052
mr_noTol	0.051	0.053	0.052

Fig. 26. Impact of different tensor modes with hierarchies and tolerance values in the intermediate levels for Tucker decomposition (dense matrices; `dblp512` data; $r_i = 5$; deepest three levels used).



Quality					
	5	10	15	20	40
$Tucker\_hooi$	0.053	0.085	0.104	0.122	0.171
mr_a	0.051	0.081	0.105	0.120	0.171

Fig. 27. Impact of the rank in Tucker models (dense matrices,`dblp512` data, 10^{-4} tolerance for the highest level, no tolerance value for the intermediate levels, only authors hierarchy).



Quality				
$Tucker$ 0.053	mr_noTol_A 0.051	mr_noTol_K 0.052	mr_noTol_C 0.053	mr_noTol_{AK} 0.051
mr_noTol_{KA} 0.051	mr_noTol_{AC} 0.052	mr_noTol_{CA} 0.052	mr_noTol_{KC} 0.052	mr_noTol_{CK} 0.052
mr_noTol_{AKC} 0.051	mr_noTol_{CKA} 0.052	mr_noTol_{KAC} 0.051	mr_noTol_{KCA} 0.051	

Fig. 28. Impact of the number of hierarchies used in the multiresolution process (Tucker, dense matrices, `dblp512` data, $r_i = 5$, 10^{-4} tolerance for the highest level, no tolerance value for the intermediate levels).

Table VII. Multiresolution Tensor Encoding Costs for Different Numbers of Approximation Levels for the Dense dblp512 Dataset

dblp512, dense, $166K$ entries			
number of approx. levels(ρ)	authors hierarchy	authors, keyw. hierarchies	authors, keyw., conf. hierarchies
1	9.2 sec	9.5 sec	9.9 sec
2	13.8 sec	12.9 sec	12.6 sec
3	16.8 sec	14.8 sec	14.2 sec
4	18.8 sec	16.0 sec	15.3 sec
5	20.3 sec	17.1 sec	16.3 sec
6	21.3 sec	17.8 sec	17.0 sec
7	22.2 sec	18.5 sec	17.7 sec

Table VIII. Multiresolution Tensor Encoding Costs for Different Numbers of Approximation Levels for the Sparse dblp20000 Dataset

dblp20000, sparse, $3.5M$ entries			
number of approx. levels(ρ)	authors hierarchy	authors, keyw. hierarchies	authors, keyw., conf. hierarchies
1	32.1 sec	32.1 sec	31.9 sec
2	61.2 sec	59.4 sec	55.9 sec
3	87.7 sec	82.2 sec	70.4 sec
4	111.9 sec	100.1 sec	76.4 sec
5	133.7 sec	114.1 sec	77.5 sec
6	152.6 sec	123.9 sec	77.6 sec
7	168.7 sec	130.2 sec	77.7 sec

the encoding phase. As can be seen, the combined use of the keyword hierarchy with the author or conference provides an execution time gain up to ∼22%, while the use of all hierarchies does not show any improvement with respect to the use of a single hierarchy. Moreover, different permutation orders do not have any significant impact on the execution times or the final quality values. Finally, the decomposition quality does not vary significantly with the use of different hierarchies.

5.5. Multiresolution Tensor Encoding Costs

So far, we have focused on the impact of available multiresolution tensor encodings on the cost and quality of tensor decomposition process. In this section, we investigate the cost of obtaining multiresolution tensors in the first place. Note that multiresolution tensors are encoded offline and the multiresolution encoding costs are often amortized over multiple decompositions of the same tensor for different subsets of modes, for different numbers of factors, and so on.

Multiresolution encoding involves the creation of multiple resolutions of the input tensor and thus depends on both the underlying representative selection function as well as the number, ρ, of approximation levels to be considered. The encoding cost also depends on the number of nonzeros in the data as well as how the tensor is organized (i.e., dense vs. sparse representations). Tables VII and VIII report, respectively, the data preparation costs for dense dblp512 (with $166K$ entries) and sparse dblp20000 tensors (with $3.5M$ entries) for different numbers of approximation levels and different numbers of modes involved in multiresolution representation. In all cases, the representative selection function is *average*.

Encoding of dense tensors. As shown in Table VII, if we consider the dense representation, the multiresolution encoding is cheap (especially relative to the cost of the

decomposition step—reported in Section 5.3.1). It is interesting to note that, while the cost increases more or less linearly with the number of approximation levels created, the cost is inversely proportional with the number of modes (except when a single approximation level is created). This is because, when more modes are involved in multiresolution representation, the intermediary encoding steps need to operate in less entries and this reduces the overall encoding time.

Encoding of sparse tensors. As Table VIII illustrates, encoding of sparse tensors is relatively faster: ∼22 seconds for 166K entries (0.13ms per entry) for dense tensors using authors hierarchy versus ∼169 seconds for 3.5M entries (0.05ms per entry) for sparse tensors for the same configuration. However, since decomposition of tensors in sparse form is relatively cheaper (especially when nonzero removal is utilized to reduce the number of close-to-zero entries—see Section 5.3.2), the multiresolution tensor encoding would constitute a larger overhead. Therefore, in the case of sparse tensors, multiresolution representation is most advantageous only if the initial encoding can be amortized over multiple decompositions of the same tensor.

6. CONCLUSIONS

In this article, we presented a metadata driven tensor decomposition approach which leverages unimodal clustering hierarchies available a priori to improve the execution times of iterative alternating squares based PARAFAC decomposition and for HOOI-based Tucker decomposition. The experiment results showed that the time gains obtained by using external metadata do not come with undesirable reductions in the decomposition quality, even when the available metadata is imperfect and not directly representative of the tensor data. Benefits of the proposed approach are especially pronounced for dense representations, while sparse PARAFAC decompositions can also benefit from multiresolution process with a new nonzero removal strategy. Tucker decompositions benefit from the multiresolution approach especially for large input tensors and small target decomposition ranks.

REFERENCES

E. Acar, D. M. Dunlavy, and T. G. Kolda. 2011. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics* 25, 2, 67–86.

E. Acar and B. Yener. 2009. Unsupervised multiway data analysis: A literature survey. *IEEE Transactions on Knowledge and Data Engineering* 21, 6–20.

B. W. Bader and T. G. Kolda. 2007. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing* 30, 1, 205–231.

B. W. Bader, T. G. Kolda, J. Sung, et al. 2012. Matlab tensor toolbox version 2.5. Available at http://www.sandia.gov/tgkolda/TensorToolbox/index-2.5.html.

R. Bro. 1998. Multi-way analysis in the food industry: models, algorithms, and applications. Ph.D. thesis, KøbenhavnsUniversitetKøbenhavns Universitet, Det Biovidenskabelige Fakultet for Fødevarer, VeteFaculty of Life Sciences, Institut for FødevarevidenskabDepartment of Food Science, Kvalitet og TeknologiQuality & Technology.

R. Bro and C. A. Andersson. 1998. Improving the speed of multiway algorithms: Part II: Compression. *Chemometrics and Intelligent Laboratory Systems* 42, 12, 105–113.

R. Bro, R. A. Harshman, N. D. Sidiropoulos, and M. E. Lundy. 2009. Modeling multi-way data with linearly dependent loadings. *Journal of Chemometrics* 23, 7–8, 324–340.

J. Carroll and J.-J. Chang. 1970. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika* 35, 283–319.

J. D. Carroll, S. Pruzansky, and J. B. Kruskal. 1980. CANDELINC: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. *Psychometrika* 45, 3–24.

Z.-P. Chen, H.-L. Wu, J.-H. Jiang, Y. Li, and R.-Q. Yu. 2000. A novel trilinear decomposition algorithm for second-order linear calibration. *Chemometrics and Intelligent Laboratory Systems* 52, 1, 75–86.

P. A. Chew, B. W. Bader, T. G. Kolda, and A. Abdelali, 2007. Cross-language information retrieval using parafac2. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*. ACM, New York, NY, 143–152.

Y. Chi and S. Zhu. 2010. Facetcube: A framework of incorporating prior knowledge into non-negative tensor factorization. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management* (*CIKM'10)*. ACM, New York, NY, 569–578.

L. De Lathauwer, B. D. Moor, and J. Vandewalle. 2000a. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications* 21, 4, 1253–1278.

L. De Lathauwer, B. D. Moor, and J. Vandewalle. 2000b. On the best Rank-1 and Rank-(R1,R2,. . .,RN) approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications* 21, 4, 1324–1342.

L. Eldén and B. Savas. 2009. A Newton–Grassmann method for computing the best multilinear rank-(r_1, r_2, r_3) approximation of a tensor. *SIAM Journal on Matrix Analysis and Applications* 31, 2, 248–271.

N. K. M. Faber, R. Bro, and P. K. Hopke. 2003. Recent developments in candecomp/parafac algorithms: a critical review. *Chemometrics and Intelligent Laboratory Systems* 65, 1, 119–137.

R. Harshman. 1970. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics* 16.

R. A. Harshman. 1972. PARAFAC2: Mathematical and technical notes. *UCLA Working Papers in Phonetics* 22, 30–44.

R. A. Harshman, S. Hong, and M. E. Lundy. 2003. Shifted factor analysis—Part I: Models and properties. *Journal of Chemometrics* 17, 7, 363–378.

M. Ishteva, P. Absil, S. Van Huffel, and L. De Lathauwer. 2011. Best low multilinear rank approximation of higher-order tensors, based on the riemannian trust-region scheme. *SIAM Journal on Matrix Analysis and Applications* 32, 1, 115–135.

J.-H. Jiang, H.-L. Wu, Y. Li, and R.-Q. Yu. 2000. Three-way data resolution by alternating slice-wise diagonalization (ASD) method. *Journal of Chemometrics* 14, 1, 15–36.

A. Kaarna, A. Andriyashin, S. Nakauchi, and J. Parkkinen. 2007. Multiresolution approach in computing ntf. In *Proceedings of the 15th Scandinavian Conference on Image Analysis (SCIA'07).*Springer-Verlag, Berlin, 334–343.

A. Kapteyn, H. Neudecker, and T. Wansbeek. 1986. An approach to n-mode components analysis. *Psychometrika* 51, 269–275. 10.1007/BF02293984.

G. Karypis and V. Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 359–392.

T. Kolda and B. Bader. 2006. The TOPHITS model for higher-order web link analysis. In *Proceedings of the SIAM Data Mining Conference Workshop on Link Analysis, Counterterrorism and Security*.

T. G. Kolda and B. W. Bader. 2009. Tensor decompositions and applications. *SIAM Review* 51, 3, 455–500.

T. G. Kolda and J. Sun. 2008. Scalable tensor decompositions for multi-aspect data mining. In *Proceedings of the 2008 8th IEEE International Conference on Data Mining*. IEEE Computer Society, Washington, DC, 363–372.

P. Kroonenberg and J. de Leeuw. 1980. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika* 45, 69–97.

M. Ley. 2009. DBLP: Some Lessons Learned. *Proceedings of the VLDB Endowment* 2, 2, 1493–1500.

K. Madsen, H. B. Nielsen, and O. Tingleff. 2004. Methods for non-linear least squares problems (2nd ed.). Available at www2.imm.dtu.dk/pubdb/views/publication_details.php?id=3215.

P. Paatero. 1999. The multilinear engine: A table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *Journal of Computational and Graphical Statistics* 8, 4, 854–888.

M. Rajih, P. Comon, and R. A. Harshman. 2008. Enhanced line search: A novel method to accelerate parafac. *SIAM Journal Matrix Analysis and Applications* 30, 3, 1128–1147.

R. T. Ross and S. Leurgans. 1995. Component resolution using multilinear models. In *Biochemical Spectroscopy*, K. Sauer, Ed. Methods in Enzymology Series, vol. 246. Academic Press, 679–700.

E. Sanchez and B. R. Kowalski. 1990. Tensorial resolution: A direct trilinear decomposition. *Journal of Chemometrics* 4, 1, 29–45.

E. Sanchez and R. Kowalski. 1986. Generalized rank annihilation factor analysis. *Analytical Chemistry* 58, 2, 496–499.

C. Schifanella., K. S. Candan, and M. L. Sapino. 2011. Fast metadata-driven multiresolution tensor decomposition. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM'11).*ACM, New York, NY, 1275–1284.

I. Stanimirova, B. Walczak, D. L. Massart, V. Simeonov, C. A. Saby, and E. D. Crescenzo. 2004. Statis, a three-way method for data analysis. application to environmental data. *Chemometrics and Intelligent Laboratory Systems* 73, 2, 219–233.

J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. 2007. Graphscope: Parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*. ACM, New York, NY, 687–696.

J. Sun, S. Papadimitriou, C. Yung Lin, N. Cao, S. Liu, and W. Qian. 2009. Multivis: Content-based social network exploration through multi-way visual analysis. In *SIAM International Conference on Data Mining*. 1063–1074.

J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen. 2005. CubeSVD: A novel approach to personalized web search. In *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*. ACM, New York, NY, 382–390.

G. Tomasi and R. Bro. 2006. A comparison of algorithms for fitting the PARAFAC model. *Computational Statistics and Data Analysis* 50, 1700–1734.

L. Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31, 3, 279–311.

L. R. Tucker. 1964. The extension of factor analysis to three-dimensional matrices. In *Contributions to Mathematical Psychology*, H. Gulliksen and N. Frederiksen, Eds. Holt, Rinehart and Winston, New York, 110–127.

F. Yates. 1933. The analysis of replicated experiments when the field results are incomplete. *Journal of Experimental Agriculture* 1, 129.

Received June 2012; revised July 2013; accepted August 2013