Sparse Hierarchical Tucker Factorization and its Application to Healthcare

Ioakeim Perros, Robert Chen, Richard Vuduc, Jimeng Sun
School of Computational Science & Engineering
Georgia Institute of Technology
Atlanta, GA, U.S.A.
{perros, rchen87}@gatech.edu, {richie, jsun}@cc.gatech.edu

Abstract—We propose a new tensor factorization method, called the Sparse Hierarchical-Tucker (Sparse H-Tucker), for sparse and high-order data tensors. Sparse H-Tucker is inspired by its namesake, the classical Hierarchical Tucker method, which aims to compute a tree-structured factorization of an input data set that may be readily interpreted by a domain expert. However, Sparse H-Tucker uses a nested sampling technique to overcome a key scalability problem in Hierarchical Tucker, which is the creation of an unwieldy intermediate dense core tensor; the result of our approach is a faster, more space-efficient, and more accurate method.

We test our method on a real healthcare dataset, which is collected from 30K patients and results in an 18th order sparse data tensor. Unlike competing methods, Sparse H-Tucker can analyze the full data set on a single multi-threaded machine. It can also do so more accurately and in less time than the state-of-the-art: on a 12th order subset of the input data, Sparse H-Tucker is $18\times$ more accurate and $7.5\times$ faster than a previously state-of-the-art method. Moreover, we observe that Sparse H-Tucker scales nearly linearly in the number of non-zero tensor elements. The resulting model also provides an interpretable disease hierarchy, which is confirmed by a clinical expert.

I. INTRODUCTION

This paper proposes a new tensor factorization method, when the number of modes is high and the input data are sparse. Tensor methods are recognized as one of the most promising approaches for mining multi-modal data, with proof-of-concept demonstrations in many application domains, such as network intrusion detection [1], and healthcare analytics [2], [3]. However, tensors also pose numerous computational scalability challenges, in all the senses of time, storage, and accuracy. This paper addresses them.

A tensor generalizes the concept of a matrix to more than two dimensions. In data analysis, each dimension is referred to as a mode, order, or way [4]. For example, a 10th order disease tensor might be constructed so as to capture interactions across 10 different disease groups. Examples of well-known tensor decomposition methods include CP (CANDECOMP-PARAFAC) and Tucker methods [5], [6]. However, despite their known value to data analysis problems, these methods have been largely limited to the analysis of data sets with a relatively small number of modes, typically 3 to 5, and so would not apply to our hypothetical 10th order example. There are two principal challenges:

1) Modeling assumptions. Traditional tensor models like CP or Tucker approximate the input data tensor as a sum of low-rank components, each of which involves all modes. Consequently, one cannot learn a model where only a subset of modes are relevant. For instance, the 10 different disease groups in our hypothetical example might have natural subgroups, or even hierarchical structure.

2) Exponential computational cost. With respect to the order of the tensor, there may be exponential costs in space and time. In the case of the Tucker method, the cause is the need to store a fully dense core tensor $C$ as output, even if the input tensor is sparse. Consider an order $d = 50$ input tensor for which we wish to compute just a very low-rank approximation of, say, $r = 2$. Then, the dense core has size $r^d$, which in this case is nearly 9 Petabytes.

Thus, the extensive literature on scaling-up tensor factorizations, which mostly targets improvements in the efficiency of CP [7], [8] or Tucker [9], are not suitable to the case of high-order tensors. Instead, we propose a scalable hierarchical tensor factorization for sparse high-order tensors, which we call the Sparse Hierarchical Tucker (or Sparse H-Tucker) method. Sparse H-Tucker expresses mode interactions as a binary tree, which is further parameterized in order to allow the approximation accuracy and cost to be tuned. It respects sparsity in the input, achieving a near-linear scale-up in time and space with respect to the non-zeros of the input tensor. This level of performance is not achieved at the cost of accuracy; on the contrary, as we verify experimentally, Sparse H-Tucker achieves remarkable gains in accuracy as well, particularly as the tensor density and order increase.

Our contributions can be summarized as follows:

- A sparse high-order tensor factorization method: We propose Sparse H-Tucker, a scalable tensor factorization method tailored to sparse and high-order tensors, which enjoys a near-linear scale-up to the number of non-zero tensor elements in both time and space.

- Demonstration on a real-world healthcare application: We present a case study in which we apply Sparse H-Tucker to disease phenotyping using electronic health records (EHR). The interpretability of the resulting hierarchical disease model is confirmed by a domain expert.
II. BACKGROUND

We denote a tensor as $\mathcal{A}$, a matrix as $\mathbf{A}$, the vector outer product as $\bullet$, the dimension tree as $T_d$, the leaf tree nodes as $L(T_d)$, the interior tree nodes as $I(T_d)$, the set of successors of parent node associated with $t$ as $s(t)$, the root node of the dimension tree as $t_r$, the index set of subset $t$ of modes as $I_t$, the cardinality of set $I_t$ as $|I_t|$.

A. Matrix factorizations

The Singular Value Decomposition (SVD) $[10]$ for $\mathbf{A} \in \mathbb{R}^{m \times n}$ is defined as $\text{svd}(\mathbf{A}) = \mathbf{U} \Sigma \mathbf{V}$ and is well-known for providing the optimal low-rank approximation in terms of the Frobenius norm. Instead of using the singular vectors in SVD, the CUR decomposition $[11]$ uses representative columns $\mathbf{C}$ and rows $\mathbf{R}$ to approximate the input matrix. The relative-error guarantees of this method $[12]$ depend on the notion of leverage score sampling. The leverage scores $\pi_j$ for each $j = 1, \ldots, n$ column of $\mathbf{A}$ are: $\pi_j = 1/k \sum_{\ell=1}^{k} (v_j(\xi))^2$, where $v_j$ is the j-th right singular vector of $\mathbf{A}$ (out of $k$ computed in total). Symmetrically, row sampling for matrix $\mathbf{R}$ is achieved by applying the above process on $\mathbf{A}^T$.

B. Tensor operations and factorizations

Tensors are high-order generalizations of matrices. A fiber is a vector extracted from a tensor by fixing all modes but one and a slice is a matrix extracted from a tensor by fixing all modes but two $[4]$. Let a d-order tensor $\mathcal{A} \in \mathbb{R}^I$. The index set over which the tensor is defined is: $I := I_1 \times \cdots \times I_d$ and the index set of each individual mode is $I_{\mu} := \{1, \ldots, n_{\mu}\}, \mu \in \{1, \ldots, d\}$. Matricization (or reshaping, unfolding) logically reorganizes tensors into other forms, without changing the values themselves. Let the index set $I^{(\mu)} := I_1 \times \cdots \times I_{\mu-1} \times I_{\mu+1} \times \cdots \times I_d$. Then, the $\mu$-mode matricization is a mapping: $\mathcal{A}^{(\mu)} : \mathbb{R}^I \rightarrow \mathbb{R}^{I_{\mu} \times I^{(\mu)}}$. As a result, the mode-$\mu$ fibers of the tensor become columns of a matrix. Given $\mathbf{U}_{\mu} \in \mathbb{R}^{J_\mu \times I_{\mu}}$, the $\mu$-mode multiplication is defined by $(\mathbf{U}_{\mu} \circ_{\mu} \mathcal{A})^{(\mu)} := \mathbf{U}_{\mu} \mathcal{A}^{(\mu)} \in \mathbb{R}^{J_\mu \times I^{(\mu)}}$. Given matrices $\mathbf{U}_{\nu} \in \mathbb{R}^{J_\nu \times I_{\nu}}$ with $\nu = 1, \ldots, d$ the multilinear multiplication is defined as: $(\mathbf{U}_1, \ldots, \mathbf{U}_d) \circ \mathcal{A} := \mathbf{U}_1 \circ_1 \cdots \mathbf{U}_d \circ_d \mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$.

The factorization of a tensor into a sum of component rank-one tensors is called the CP/PARAFAC $[5]$ factorization. If the rank of a d-order tensor $\mathcal{A}$ is equal to $R$, then: $\mathcal{A} = \sum_{r=1}^R \lambda_r a_r^{(1)} \bullet a_r^{(2)} \bullet \cdots \bullet a_r^{(d)}$. The most popular factorization approach approximating the above model is the CP Alternating Least Squares (ALS) $[4]$. The Tucker format is given by the following form $[6, 13]$: $\mathcal{A} = (\mathbf{U}_1, \ldots, \mathbf{U}_d) \circ \mathbf{C}$, where $\mathbf{U}_\mu \in \mathbb{R}^{I_{\mu} \times k_{\mu}}$ are (columnwise) orthonormal matrices and $\mathbf{C} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$ is a core tensor. For fixed $U_{\mu}$ matrices, the unique core tensor minimizing the approximation error is given by: $\mathbf{C} = (U_1^T, \ldots, U_d^T) \circ \mathcal{A}$. If the core tensor is computed in the above way and each $U_{\mu}$ contains the leading $k_{\mu}$ left singular vectors of $\mathcal{A}^{(\mu)}$, the factorization of tensor $\mathcal{A}$ is called the higher-order SVD (HOSVD) $[13, 4]$. HOSVD is considered as a good initialization to the higher-order orthogonal iteration (HOOI) $[14]$, which is an ALS-type algorithm to approximate the Tucker format.

Hierarchical Tucker and its Limitations One popular model that shares structural similarities with our proposed model is the Hierarchical Tucker (H-Tucker in short) presented in $[15]$. Intuitively, the H-Tucker factorization algorithm proposed in $[15]$ first decomposes the input tensor into the Tucker format through the HOSVD and then recursively factorizes the output tensor of this process. Such a strategy though suffers from severe scalability issues as the tensor order d increases. Despite the fact that the final form of H-Tucker requires linear storage to $d$, the size of the intermediate core tensor computed increases exponentially to $d$; and this core tensor is dense. As a result, this method faces a potential memory blow-up as it requires further decomposing an intermediate result that may not even fit into memory. Another factorization scheme that is based on H-Tucker and is similar to ours was proposed in the tensor community by Ballani et al $[16]$. However, that work exclusively targets dense tensors (does not work for sparse input), while ours focuses on sparse ones and data mining applications.

III. SPARSE HIERARCHICAL TUCKER

A. Model

Our proposed target model is called the Sparse Hierarchical Tucker (Sparse H-Tucker). In Sparse H-Tucker, the tensor modes are split recursively, resulting in a binary tree that we call the dimension tree and denote by $T_d$. Each node of this tree contains a subset $t \subset \{1, \ldots, d\}$ of the modes and is either a leaf and singleton $t = \mu$ or the union of its two disjoint successors $t_1, t_2 : t = t_1 \cup t_2$. Each tree node is associated with an output factor of the model. We denote these output factors by:

$$(B_t)_{t \in \mathcal{I}(T_d)} \in \mathbb{R}^{k_{t_1} \times k_{t_2} \times k_t}, (\mathbf{U}_t)_{t \in \mathcal{I}(T_d)} \in \mathbb{R}^{I_t \times k_t}.$$  

The tensors $B_t$ are called the transfer tensors, which correspond to the interior nodes, $\mathcal{I}(T_d)$; the matrices $\mathbf{U}_t$ correspond to the leaves of the tree, $L(T_d)$, where $s(t) = \{t_1, t_2\}$ denotes the set of successors of node $t$. By definition, the matrices $\mathbf{U}_t$ associated with the leaves of this tree structure are sparse. The tensor associated with the root node $t_0$ is a degenerate one (i.e., it is a matrix since $k_{t_0} = 1$), because unlike other interior nodes, only the root node connects to 2 nodes instead of 3.

Our proposed model’s tree structure is like that of the H-Tucker model $[15]$. However, Sparse H-Tucker preserves sparsity. By contrast, in H-Tucker, the matrices corresponding to the leaf nodes are dense, which fundamentally limits the scalability of any algorithms operating on it.
B. Sparse H-Tucker factorization algorithm

The rationale behind these two phases is first to conduct all 

- **Phase 1** computes a sampling-based low-rank approximation of all $A^{(t)}$ associated with each tree node except for the root. Notice that $A^{(t)}$ combines all modes contained in $t$ as row indices and the rest of the modes into column indices.
- **Phase 2** uses the output of Phase 1, in order to assemble the final Sparse H-Tucker model in parallel.

The rationale behind these two phases is first to conduct all the preparation work in Phase 1 and then to compute the expensive steps fully in parallel in Phase 2.

**Algorithm 1** Sparse Hierarchical Tucker factorization

**Input:** Input tensor $A \in \mathbb{R}^t$, tree $T_2$, accuracy parameter $\epsilon$

**Output:** $(B_1)_{i \in \mathcal{L}(T_2)}, (U_i)_{i \in \mathcal{L}(T_2)}$

```
1: {P_t, Q_t, M_t} = TreeParameterization(A, t, \epsilon)  // Phase 1
2: for each $t \in T_2$ do  // Phase 2: fully-parallelizable loop
3: if $t \in T(T_2)$ then
4: Compute $B_t$ through Equation 3
5: else
6: Compute sparse matrix $U_t$ through Equation 4
7: end if
8: end for
```

**Algorithm 2** TreeParameterization

**Input:** Tensor $A$, tree node $t$, sampled column indices $Q_t$, accuracy parameter $\epsilon$

**Output:** $\{P_t, Q_t, M_t\}_{t \in T_2 \setminus t_r}$

```
1: $\{t_1, t_2\} = s(t)\quad$ // Algorithm $[$
2: $[P_{t_1}, Q_{t_1}, M_{t_1}, A_{t_1}] = NestedSampling(A, t_1, Q_{t_1}, \epsilon)$
3: $[P_{t_2}, Q_{t_2}, M_{t_2}, A_{t_2}] = NestedSampling(A, t_2, Q_{t_2}, \epsilon)$
4: if $t_1 \in T(T_2)$ then
5: TreeParameterization(A, t_1, Q_{t_1}, \epsilon)
6: end if
7: if $t_2 \in T(T_2)$ then
8: TreeParameterization(A, t_2, Q_{t_2}, \epsilon)
9: end if
```

Algorithm $[^1]$ is our top-level procedure to compute the Sparse H-Tucker form. It takes as input the original tensor $A$, the dimension tree structure $T_2$ and a parameter $\epsilon$ which governs the accuracy of low-rank approximations. In Line $[^1]$ of Algorithm $[^1]$ we invoke Algorithm $[^2]$ by starting the recursion from the root node of the tree ($t_r$) to parameterize the dimension tree.

Within Algorithm $[^2]$ Lines $[^2]$ and $[^3]$ call the function NestedSampling to compute the factors for the approximation of each $A^{(t)}$. If $C_t$ and $R_t$ contain column and row samples from $A^{(t)}$, respectively, and $M_t$ is a small matrix minimizing the error of approximation, then the product $C_t M_t R_t$ is an approximation of $A^{(t)}$. To avoid the materialization of $C_t$ and $R_t$, we maintain the index sets $P_t, Q_t$ denoting the row and column indices sampled from $A^{(t)}$ respectively. The challenges emerging so as to execute the NestedSampling function and its exact operation will be explained in Section III.C. The recursive procedure TreeParameterization is continued until we reach the leaf nodes.

In Phase 2 of Algorithm $[^1]$ we construct the output factors of the Sparse H-Tucker model, by exploiting the sampling results from Phase 1. Since the construction over a single node is completely independent to others, we can fully parallelize this step.

To assemble the matrices $U_t$ corresponding to the leaf nodes, we directly sample from the column fibers of $A^{(t)}$:

$$((U_t)_{i \in \mathcal{L}(T_2)}) = A^{(t)}(:, q_i), q_i \in Q_t. \quad (1)$$

Since we are sampling directly from the sparse input tensor for the construction of the $(U_t)_{i \in \mathcal{L}(T_2)}$ matrices, our leaf output factors maintain the sparsity of the input tensor. Thus, the requirement of our model for sparsity on matrices associated with leaf nodes is satisfied.

A great advantage of the model is that the transfer tensors are directly assembled without the need of computing a huge, dense intermediate result (as in the case of the H-Tucker model). Below, we provide the equation for computing the factors $(B_{i})_{i \in \mathcal{L}(T_2)}$ for the interior tree nodes. The proof of its correctness will be given in an extended version of this paper. Given nodes $t, t_1, t_2$ where $\{t_1, t_2\} = s(t)$ and $q_i \in Q_{t_1}, q_j \in Q_{t_2}, q_i \in Q_{t_1}$:

$$(B_t)_{i,j,l} = \sum_{p \in P_{t_1}} \sum_{q \in P_{t_2}} (M_{t_1})_{q_i, p} A^{(t)}(p, q_j, (M_{t_2})_{q_j, q_i}. \quad (2)$$

C. Nested sampling

Below, we describe the NestedSampling function that is called within Algorithm $[^2]$. Its role is to compute the factors required to approximate the matricizations $A^{(t)}$ for each subset of modes $t$ associated with each tree node. Our approach is to form the factors approximating $A^{(t)}$ through the CUR decomposition based on leverage score sampling $[11]$. The biased samples from CUR decomposition help to boost the accuracy of our approach. More specifically, we follow the same sampling strategy as in $[11]$, by retrieving $O(k \log k/\epsilon^2)$ rows or columns for each required approximation, where $k$ is the rank of SVD, which is set to a small value ($k = 5$) (we detected no significant change in the accuracy of the final approximation by tuning $k$, hence we keep it fixed).

However, a simple application of the CUR decomposition within our factorization framework would completely fail, due to challenges related to the consistency of each $A^{(t)}$ approximation with the whole framework and the scalability of each decomposition even for low-order tensors. Next we will present the way we tackle the aforementioned challenges.

**Consistency:** Assume calling the NestedSampling function with arguments $A, t_1, Q_{t_1}, \epsilon$ (as happens in Line $[^2]$ of
Algorithm [2]. Before even attempting to execute a nested sampling of $A^{(t)}$, we have to ensure that the available set of rows and columns is consistent across the entire dimension tree. In other words, we have to ensure that the way we extract our model’s transfer tensors (Equation 2) is consistent to each individual $A^{(t)}$ approximation. To do so, we have to guarantee that:

$$Q_{t_1} \subseteq I_{t_2} \times Q_t$$

where $\{t_1, t_2\} = s(t)$ and the notation $\times$ denotes the cartesian set product. Relation 3 implies that the columns sampled ($Q_{t_1}$) by the successor node associated with $t_1$ will be a subset of all possible combinations formed between the unrestricted index set $I_{t_2}$ and the fixed index set $Q_t$ (which had been previously defined by the parent node). By symmetry, $Q_{t_2} \subseteq I_{t_1} \times Q_t$. The theoretical reasoning behind the above index restriction will be included in the paper’s extended version.

Tensors $A_1, A_2$ are passed in each TreeParameterization call so as to avoid starting the index dropping from scratch at each NestedSampling call. Those tensors are obtained by finding the subset of tensor entries of $A$ that obey to the rule of Relation 3.

scalability: To calculate the leverage scores, we need to compute $\text{SVD}(A^{(t)})$ for all tree nodes. However, the size of both row and column index sets for nodes close to the root may be huge, even if the original tensor is of moderate order and individual mode size. This results in severe memory blowup issues.

For example, consider an 8th order tensor ($d = 8$) with individual size of each mode equal to $n = 1000$. In the simplest case of a balanced tree, we would have to matrixize the sparse input tensor into a sparse matrix of size $n^{d/2} \times n^{d/2}$ and perform SVD. Even in the unrealistic case of a completely empty sparse matrix, the space required for its construction is $\approx 7.4$ TeraBytes (this stems from the underlying sparse matrix storage requirements). In order to cope with this limitation, we exploit our following remark:

Remark: Let $I_{n_{zr}} \subseteq I_1$ be the set of indices of non-zero rows of $A^{(t)}$ following their original ordering. Let $I_{n_{zc}} \subseteq I(t)$ be the set of indices of non-zero columns of $A^{(t)}$ following their original ordering. Let $|I_{n_{zr}}| = m_i |I_{n_{zc}}| = n$ and a matrix $T \in \mathbb{R}^{m \times n}$. Then, every non-zero $(i, j)$ entry of $A^{(t)}$ will be placed in a position of $T$ following the map ($I_{n_{zr}}, I_{n_{zc}}$) $\rightarrow$ ($\{1, \ldots, m\}, \{1, \ldots, n\}$). We execute the SVD(T) and each $j$-th column leverage score corresponds to the original space following the backward mapping: $\{1, \ldots, n\} \rightarrow I_{n_{zr}}$. This applies similarly for the row leverage scores, using the left singular vectors and the mapping between $\{1, \ldots, m\}$ and $I_{n_{zc}}$.

This observation enables us to avoid materializing sparse matrices with huge row and column index sets and eventually enables the execution of CUR factorization via the leverage-score sampling strategy.

### IV. Experiments

#### A. Setup

Our experiments were conducted on a server running the Red Hat Enterprise 6.6 OS with 64 AMD Opteron processors (1.4 GHz) and 512 GB of RAM. We used Matlab R2015a as the programming framework as well as Matlab Tensor Toolbox v2.6 [18] in order to support tensor operations. Our source code is open-sourced and publicly available [1].

The methods under comparison are the following: Sparse H-Tucker (Sequential): Sparse H-Tucker implementation with sequential execution in Phase 2; Sparse H-Tucker (Parallel): Sparse H-Tucker implementation with parallel execution in Phase 2; H-Tucker: Hierarchical Tucker implementation provided in htucker toolbox [17]; CP-ALS: Tensor Toolbox [18] implementation; and Tucker-ALS: Tensor Toolbox [18] implementation of HOOI.

#### B. Experiments on real healthcare data

Dataset and task description We used publicly available healthcare data for our experimental evaluation. The dataset is called MIMIC-II and can be found in [19] [1]. It contains disease history of 29,802 patients where an overall of 314,647 diagnostic events are recorded over time. The task is about extracting co-occurring patterns of different diagnoses in patient records, in order to better understand the complex interactions of disease diagnoses. To acquire accurate phenotyping, we exploit the domain knowledge provided by the International Classification of Diseases (ICD) hierarchy [20] and guide the tensor construction with it. The ICD hierarchy consists of a collection of trees representing hierarchical relationships between diagnoses. As such, diagnoses belonging to the same diagnostic family reside under the same sub-tree. We map each tensor node to a node of the top-level hierarchy. Thus, the order of our tensor will be equal to the number of top-level nodes. Furthermore, the lower-level diagnoses that are contained in each top-level node will be the elements of a tensor mode.

Input tensor construction In order to end up with the previously described tensor, we sum over the number of co-occurrences for each multi-index of diagnoses. This means

![Table I Description of Tensors Used in Our Experiments](http://www.cc.gatech.edu/~iperros3/sec/sp_hucker.zip)

In order to enable sparse tensor input, we modified the computation of the left leading singular vectors by re-directing to the “invec” function of the “sptensor” Tensor Toolbox class.

1http://www.cc.gatech.edu/~iperros3/sec/sp_hucker.zip
2In order to enable sparse tensor input, we modified the computation of the left leading singular vectors by re-directing to the “invec” function of the “sptensor” Tensor Toolbox class.
3http://physionet.org/mimic2/
that each one of the input tensor cells contains the sum of the corresponding diagnoses found for all patients. For example, consider the case of a 3-order tensor \( T \) where each one of the 3 modes corresponds to a group of similar diseases. If a certain combination of diseases \((i, j, k)\) is co-occurring in 3 patients out of the whole dataset, then \( T(i, j, k) = 3 \). Since the top-level of the ICD hierarchy contains 18 nodes, our complete tensor input is an 18-order tensor. For the purposes of our experimental evaluation, we constructed tensors from the same dataset with varying tensor order (less than 18) by limiting the subset of disease groups. Also, to each one of the modes, we added an additional element corresponding to "no disease", so that we model the case when a mode does not participate in a certain co-occurrence at all.

**Scalability** We conducted experiments in order to assess the scalability behavior of the methods under comparison, with respect to both the time and space required for increasing tensor order and number of non-zero elements. The input tensors having different density and order were constructed as explained above. For Sparse H-Tucker (parallel), we set the number of Matlab workers to 16, so as to exploit the full parallelism potential of our method for higher orders. We were not able to reliably measure the memory overhead for this version of our approach. Still, we empirically remark that the memory required for parallel Sparse H-Tucker shares the same scalability properties as the sequential version. The results are presented in Figure 1. It is remarkable that the H-Tucker factorization could not run for none but the 4-order tensor case. For the 6-order case and beyond, the memory it required exceeded the available memory of our server. The same behavior is observed in the case of Tucker-ALS. On the other hand, despite having comparable scalability behavior to Sparse H-Tucker, the CP method could not factorize the highest order tensors (16, 18) due to numerical issues (matrix being close to singular, in a sub-problem assuming a full-rank matrix). Our proposed Sparse H-Tucker enjoys near-linear scalability properties with respect to increasing the non-zero elements or tensor orders for both time and space requirements.

**Cost-Accuracy Trade-off** We would also like to evaluate the time-error trade-off as the tensor order increases. It was intractable for any method to re-construct the full (dense, due to approximation errors) tensor for any order but the 4th; as such, we evaluated a random sample of 50K out of all the non-zero tensor elements, for each one of the methods (element-wise evaluation). Then, we measured the approximation error of those re-constructed entries with the real ones from the original input tensor. Since the 4th order tensor contained less than 50K non-zero values, we measured the error for the whole tensor. In Figure 1(c), we present the results of this experiment. We would like to highlight the fact that as the tensor order increases, our method achieves increasingly beneficial cost-error trade-offs over the CP-ALS method. In particular, for the 12-order tensor, Sparse H-Tucker achieves 18x reduction of the reconstruction error in 7.5x less time.

C. Disease phenotyping case study

In this section, we apply Sparse H-Tucker method to disease phenotyping. The qualitative analysis refers to the results of factorizing the full 18-order disease co-occurrence tensor.

The factors of the Sparse H-Tucker model are fit according to a certain tree structure. Such a tree can be obtained directly from existing knowledge such as a domain medical ontology or derived from data. In this case study, we build this dimension tree in a completely data-driven fashion using hierarchical clustering. For each one of the \( m \) non-zero values of the input tensor, we create a binary vector of size \( d \), the order of the input tensor. This vector contains ones in the non-null positions of each specific entry. The columns of the \( m \times d \) matrix formed are considered as data points into a \( m \)-dimensional space and are hierarchically clustered according to the Jaccard coefficient. The tree construction
for the H-Tucker model is attempted by the recent work in [21]. However, the cost of their algorithm is prohibitive.

**Interpretation of output factors** We propose the following interpretation of the output factors: the non-zero elements that correspond to each one of the column vectors of the matrices $((U_t)_{t=1}^{T})$ form a concept for each individual mode $t$ of the input tensor. Also, the numerical values of those elements are clear indicators of their “contribution” to each concept, since these are actual fibers containing co-occurrence counts from the input tensor.

As concerns the interpretation of transfer tensors $B_t$ with $\{t_1, t_2\} = s(t)$, they should be considered as reflecting the interactions between the concepts of the successor nodes $t_1, t_2$. Thus, the $(B_t)_{(i,j,v)}$ elements having the largest absolute value within each $v$-th slice reflect a joint concept formed through the $j$-th concept of $t_1$ and the $v$-th concept of $t_2$.

**Qualitative analysis** The most significant concepts grouped together as the result of applying our tensor factorization method, are shown in Table [II] Those results were examined by a domain expert who verified their clinical value and meaningfulness. A detailed discussion on the phenotypes formed will be provided in the paper’s extended version.

### V. CONCLUSION

In this work, we propose a scalable high-order tensor factorization method specifically designed for data analytics. Our experiments on real data established its accuracy, scalability, and usefulness for healthcare analytics. Sparse H-Tucker is not limited though to healthcare applications; this is just the focus of the current work and the application on more datasets and domains is left as a future work. Future work will also focus on further examining the underlying tree’s construction and the method’s theoretical properties.

**ACKNOWLEDGMENT**

This work was supported by the U.S. National Science Foundation (NSF), Award Numbers IIS-1418511, CCF-1533768, 1339745, 1337177, and 1533768, Children’s Healthcare of Atlanta, CDC I-SMILE project, Google Faculty Award, AWS Research Award, Microsoft Azure Research Award and UCB. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding parties.

We would like to thank the anonymous reviewers for their constructive comments which helped us improve this work. We would also like to thank Evangelos E. Papalexakis for his valuable feedback which helped us improve the paper’s clarity.

Table II

<table>
<thead>
<tr>
<th>Diagnostic family</th>
<th>Grouped clinical concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endocrine, Nutritional, Metabolic Diseases, Immunity Disorders</td>
<td>Pure hypercholesterolemia, Type II diabetes mellitus, Other and unspecified hyperlipidemia, Coronary atherosclerosis of native coronary artery, Hyper tension, Atrial fib rillation, Congestive heart failure</td>
</tr>
<tr>
<td>Diseases of the Circulatory System</td>
<td>Anemia unspecified, Acute posthemorrhagic anemia, Thrombocytopenia, Secondary thrombocytopenia, Chronic airway obstruction, Asthma unspecified type without mention of status asthmaticus, Iatrogenic pneumothorax, Pulmonary collapse, Pleural effusion, Pneumonia organism unspecified</td>
</tr>
<tr>
<td>Diseases of the Blood and Blood-Forming Organs</td>
<td>Undiagnosed cardiac murmurs</td>
</tr>
<tr>
<td>Diseases of the Respiratory System</td>
<td>Other Staphylococcus infection in conditions classified elsewhere and of unspecified site, Septicemia</td>
</tr>
</tbody>
</table>

### REFERENCES


