Computing Dense Tensor Decompositions with Optimal Dimension Trees

Oguz Kaya, Yves Robert

Algorithmica manuscript No.
(will be inserted by the editor)

Computing Dense Tensor Decompositions with Optimal Dimension Trees

Oguz Kaya · Yves Robert

Received: date / Accepted: date

Abstract Dense tensor decompositions have been widely used in many signal processing problems including analyzing speech signals, identifying the localization of signal sources, and many other communication applications. Computing these decompositions poses major computational challenges for big datasets emerging in these domains. CANDECOMP/PARAFAC (CP) and Tucker formulations are the prominent tensor decomposition schemes heavily used in these fields, and the algorithms for computing them involve applying two core operations, namely tensor-times-matrix (TTM) and tensor-times-vector (TTV) multiplication, which are executed repetitively within an iterative framework. In the recent past, efficient computational schemes using a data structure called dimension tree, are employed to significantly reduce the cost of these two operations, through storing and reusing partial results that are commonly used across different iterations of these algorithms. This framework has been introduced for sparse CP and Tucker decompositions in the literature, and a recent work investigates using an optimal binary dimension tree structure in computing dense Tucker decompositions. In this paper, we investigate finding an optimal dimension tree for both CP and Tucker decompositions. We show that finding an optimal dimension tree for an N-dimensional tensor is NP-hard for both decompositions, provide faster exact algorithms for finding an optimal dimension tree in $O(3^N)$ time using $O(2^N)$ space for the Tucker case, and extend the algorithm to the case of CP decomposition with the same time and space complexities.

Oguz Kaya
INRIA Bordeaux, 200 Avenue de la Vieille Tour 33400 Talence, FRANCE
Tel.: +33-689985143
E-mail: oguz.kaya@inria.fr

Yves Robert
ENS Lyon, 46 Allée d'Italie 69007 Lyon, FRANCE
University Tennessee Knoxville, USA
E-mail: yves.robert@inria.fr

Keywords tensor computations, CP decomposition, Tucker decomposition, dimension tree

1 Introduction

In parallel with the increasing data size in big data applications, the number of data features also escalates, which in turn augments the dimensionality of data. As a result, tensors, or multi-dimensional arrays, have increasingly been used in the recent past due to their ability to naturally model such data in many application domains including the analysis of Web graphs [19], knowledge bases [6], recommender systems [29,30,34], signal processing [21,25,26,31], computer vision [35], health care [27], and many others [20]. In these applications, tensor decomposition algorithms are used as an effective tool for analyzing data in order to extract latent information within the data, or predict missing data elements. There have been considerable efforts in designing numerical algorithms for different tensor decomposition problems (see the survey [20]) and algorithmic and software contributions go hand in hand with these efforts [2, 4, 9, 13, 16–18, 32, 33]. In particular, dense tensor decompositions have proven to be among the most powerful tools in many signal processing applications [25, 26, 31]. Among these applications are analyzing speech signals for source separation [25], finding the localization of the signal source from radar signals [26], and other communication applications [31].

The two prominent tensor decomposition approaches employed in these applications are CANDECOMP/PARAFAC (CP) and Tucker formulations. CP decomposition approximates a given tensor as a sum of rank-one tensors. The standard algorithm for computing CP decomposition is CP-ALS [7, 11], which is based on the alternating least squares (ALS) method, though other variants also exist [1]. The computational core of these algorithms involve a special operation called the matricized tensor-times Khatri-Rao product (MTTKRP). The standard method for computing Tucker decomposition is Higher Order Orthogonal Iteration algorithm (HOOI), whose computational cost is dominated by an operation called tensor-times-matrix multiplication (TTM). All these algorithms are iterative, in which MTTKRP and TTM operations are performed repetitively in alternating dimensions. For an N-dimensional tensor, the traditional methods necessitate the multiplication of the input tensor with $O(N^2)$ matrices in each iteration of these algorithm, which gets very expensive as the dimensionality of tensor increases. Efficiently carrying them out for higher dimensional tensors has been the focus of recent work [8, 15, 18]. Specifically, [14] and [18] investigate the use of a data structure called *dimension tree* in order to reduce the number of such multiplications to $N \log N$ within an iteration of HOOI and CP-ALS algorithms, respectively, for sparse tensors. For dense tensors, the sheer number of multiplications is not the most precise cost metric, and for this reason Choi et al. [8] investigate the use of an optimal dimension tree structure, computed in $O(4^N)$ time using $O(3^N)$

space, that potentially performs more TTMs in total, yet yields the lowest actual operation count possible.

The main objective of this paper is to investigate the complexity of finding an optimal dimension tree for both CP and Tucker decompositions, and to design efficient algorithms for finding such trees. Here follows the list of our contributions. For Tucker decomposition, we show that finding an optimal dimension tree which is *balanced* and *binary* is NP-hard. We conjecture that finding an optimal tree in the general case stays NP-hard. We introduce a fast greedy algorithm for finding the optimal order for a series of TTMs, which in turn enables us designing an exact algorithm for finding an optimal binary dimension tree in $O(3^N)$ time using $O(2^N)$ memory using a dynamic programming formulation, a significant improvement with respect to the state of the art requiring $O(4^N)$ time and $O(3^N)$ space [8]. We show, however, with a counter-example that an optimal dimension tree is not necessarily binary in the Tucker case. For CP decomposition, we show that an optimal dimension tree must be binary, and prove that finding a such tree is NP-hard. We then extend the algorithm for the Tucker case to the CP case for similarly finding an optimal binary dimension tree in $O(3^N)$ time using $O(2^N)$ memory.

The organization of the rest of the paper is as follows. In the next section, we provide some background, including our notation and a survey involving the descriptions of CP and Tucker decompositions together with their efficient computation with the help of dimension trees. Meanwhile, we mention related work pertaining to computing these two decompositions. Next, we provide in Section 3 all our theoretical findings and algorithms regarding an optimal dimension tree structure for Tucker decomposition. Then in Section 4, we give related theorems and algorithms for finding an optimal dimension tree in computing CP decomposition. Section 5 is devoted to a survey a related work. Finally, Section 6 provides final remarks and hints for future work.

2 Notation and background

We denote the set $\{1, \ldots, M\}$ of integers as $\mathbb{N}_M$ for $M \in \mathbb{Z}^+$. For vectors, we use bold lowercase Roman letters, as in $\mathbf{x}$. For matrices, we use bold uppercase Roman letters, e.g., $\mathbf{X}$. For tensors, we generally follow the notation in Kolda and Bader's survey [20]. We represent tensors using bold calligraphic fonts, e.g., $\boldsymbol{\mathcal{X}}$. The *order* of a tensor is defined as the number of its *dimensions*, or equivalently, *modes*, which we denote by N. We use italic lowercase letters with corresponding indices to represent vector, matrix, and tensor elements, e.g., x_i for a vector x, $x_{i,j}$ for a matrix $\mathbf{X}$, and $x_{i,j,k}$ for a 3-dimensional tensor $\boldsymbol{\mathcal{X}}$. For column vectors of a matrix, we use the same letter in lowercase and with a subscript corresponding to the column index, e.g., $\mathbf{x}_i$ to denote $\mathbf{X}(:, i)$. A *slice* of a tensor in the nth mode is a set of tensor elements obtained by fixing the index only along the nth mode. We use the MATLAB notation to refer to matrix rows and columns as well as tensors slices, e.g., $\boldsymbol{\mathcal{X}}(i, :)$ and

$\boldsymbol{\mathcal{X}}(:,j)$ are the ith row and the jth column of $\boldsymbol{\mathcal{X}}$, whereas $\boldsymbol{\mathcal{X}}(:,:,k)$ represents the kth slice of $\boldsymbol{\mathcal{X}}$ in the third dimension.

The multiplication of an N-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ along a dimension $n \in \mathbb{N}_N$ with a vector $\mathbf{v} \in \mathbb{R}^{I_n}$ is a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times 1 \times I_{n+1} \times \cdots \times I_N}$ with elements

$$y_{i_1,\ldots,i_{n-1},1,i_{n+1},\ldots,i_N} = \sum_{j=1}^{I_n} v_j x_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N}. \tag{1}$$

This operation is called tensor-times-vector multiply (TTV) and is denoted by $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_n \mathbf{v}$. The cost of this operation is $O(\prod_{i \in \mathbb{N}_N} I_i)$. $\boldsymbol{\mathcal{X}}$ can also be multiplied with a matrix $\mathbf{U} \in \mathbb{R}^{K \times I_n}$ in a mode n, which results in the tensor $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_n \mathbf{U}$, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times K \times I_{n+1} \times \cdots \times I_N}$ with elements

$$y_{i_1,\ldots,i_{n-1},k,i_{n+1},\ldots,i_N} = \sum_{j=1}^{I_n} u_{k,j} x_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N}. \tag{2}$$

In other words, TTV of $\boldsymbol{\mathcal{X}}$ with the kth row vector of $\mathbf{U}$ forms the kth slice of $\boldsymbol{\mathcal{Y}}$ in the nth dimension. This operation is called tensor-times-matrix multiply (TTM) and has the cost $O(K \prod_{i \in \mathbb{N}_N} I_i)$. The order of TTVs or TTMs in a set of distinct modes is irrelevant, i.e., $\boldsymbol{\mathcal{X}} \times_i \mathbf{u} \times_j \mathbf{w} = \boldsymbol{\mathcal{X}} \times_j \mathbf{w} \times_i \mathbf{u}$ for $\mathbf{u} \in \mathbb{R}^{I_i}$, $\mathbf{w} \in \mathbb{R}^{I_j}$, $i \neq j$, and $i, j \in \mathbb{N}_N$.

A tensor $\boldsymbol{\mathcal{X}}$ can be *matricized* in some modes, meaning that a matrix $\mathbf{X}$ can be associated with $\boldsymbol{\mathcal{X}}$ by identifying a subset of its modes to correspond to the rows of $\mathbf{X}$, and the rest of the modes to correspond to the columns of $\mathbf{X}$. This involves a mapping of the elements of $\boldsymbol{\mathcal{X}}$ to those of the matricization $\mathbf{X}$ of the tensor. We will be exclusively dealing with the matricizations of tensors along a single mode, meaning that a single mode is mapped to the rows of the resulting matrix, and the rest of the modes correspond to its columns. We use $\mathbf{X}_{(d)}$ to denote matricization along a mode d, e.g., for $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, the matrix $\mathbf{X}_{(1)} \in \mathbb{R}^{I_1 \times I_2 I_3 \ldots I_N}$ denotes the mode-1 matricization of $\boldsymbol{\mathcal{X}}$. Specifically, the tensor element $x_{i_1,\ldots,i_N}$ corresponds to the element with index $\left(i_1, i_2 + \sum_{j=3}^{N} \left[(i_j - 1) \prod_{k=2}^{j-1} I_k\right]\right)$ of $\mathbf{X}_{(1)}$ in this matricization. Matricizations in other modes are defined similarly.

The *Hadamard product* of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^I$ is a vector $\mathbf{w} = \mathbf{u} * \mathbf{v}, \mathbf{w} \in \mathbb{R}^I$, where $w_i = u_i \cdot v_i$. The *outer product* of $K > 1$ vectors $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(K)}$ of corresponding sizes $I_1, \ldots, I_K$ is denoted by $\boldsymbol{\mathcal{X}} = \mathbf{u}^{(1)} \circ \cdots \circ \mathbf{u}^{(K)}$ where $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_K}$ is a K-dimensional tensor with elements $x_{i_1,\ldots,i_K} = \prod_{t \in \mathbb{N}_K} u_{i_t}^{(t)}$. The *Kronecker product* of vectors $\mathbf{u} \in \mathbb{R}^I$ and $\mathbf{v} \in \mathbb{R}^J$ results in a vector $\mathbf{w} = \mathbf{u} \otimes \mathbf{v}$, $\mathbf{w} \in \mathbb{R}^{IJ}$, which is defined as

$$\mathbf{w} = \mathbf{u} \otimes \mathbf{v} = \begin{bmatrix} u_1 \mathbf{v} \\ u_2 \mathbf{v} \\ \vdots \\ u_I \mathbf{v} \end{bmatrix}.$$

For matrices $\mathbf{U} \in \mathbb{R}^{I \times K}$ and $\mathbf{V} \in \mathbb{R}^{J \times K}$, their *Khatri-Rao product* corresponds to the Kronecker product of their corresponding columns, i.e.,

$$\mathbf{W} = \mathbf{U} \odot \mathbf{V} = [\mathbf{u}_1 \otimes \mathbf{v}_1, \ldots, \mathbf{u}_K \otimes \mathbf{v}_K] \,, \tag{3}$$

where $\mathbf{W} \in \mathbb{R}^{IJ \times K}$.

We use the shorthand notation $\diamond_{i \neq n} \mathbf{u}^{(i)}$ to denote operation $\diamond$ using a set $\{\mathbf{u}_1^{(1)}, \ldots, \mathbf{u}^{(N)}\}$ of operands, i.e., $\boldsymbol{\mathcal{X}} \times_{i \neq n} \mathbf{u}^{(i)}$ denotes $\boldsymbol{\mathcal{X}} \times_1 \mathbf{u}^{(1)} \times_2 \cdots \times_{n-1} \mathbf{u}^{(n-1)} \times_{n+1} \mathbf{u}^{(n+1)} \times_{n+2} \cdots \times_N \mathbf{u}^{(N)}$. Similarly, we employ the notation $\diamond_{i \in \mathcal{I}} \mathbf{u}^{(i)}$ to denote an operation over a subset $\mathcal{I} = \{i_1, \ldots, i_{|\mathcal{I}|}\}$ of dimensions , i.e., $\boldsymbol{\mathcal{X}} \times_{i \in \mathcal{I}} \mathbf{u}^{(i)} = \boldsymbol{\mathcal{X}} \times_{i_1} \mathbf{u}^{(i_1)} \times_{i_2} \cdots \times i_{i_{|\mathcal{I}|}} \mathbf{u}^{(i_{|\mathcal{I}|})}$

Next, we describe a data structure called *dimension tree* that has been employed in the literature for efficiently computing Tucker and CP decompositions. Afterwards, we provide the descriptions of the standard algorithms for computing Tucker and CP decompositions together with their dimension tree-based variants. We assume hereafter that the input tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ has dimensions $I_n \geq 2$ for $n \in \mathbb{N}_N$, as one can otherwise remove all dimensions of size 1 and execute the algorithms on the resulting lower dimensional tensor.

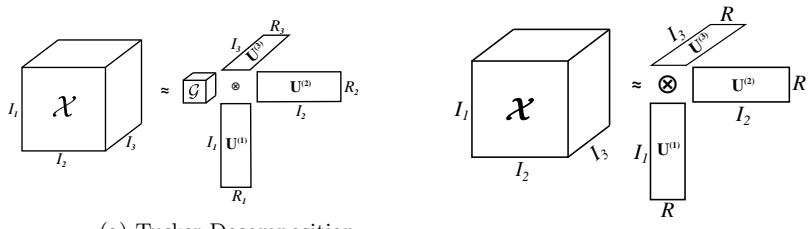
2.1 Dimension tree

A *dimension tree* partitions the mode indices of an N-dimensional tensor in a hierarchical manner for computing tensor decompositions efficiently. It was first used in the hierarchical Tucker format representing the hierarchical Tucker decomposition of a tensor [10], which was introduced as a computationally feasible alternative to the original Tucker decomposition for higher order tensors. Later, it was employed by Kaya and Uçar [14, 18] for efficiently computing standard CP and Tucker decompositions, which was made possibly computing, storing, and reusing *partial* TTM and TTV multiplications in a dimension tree structure. In the following part, we provide the formal definition of a dimension tree.

Definition 1. *A dimension tree $\mathcal{T}$ for N dimensions is a rooted tree with N leaf nodes. In a dimension tree, each non-leaf node has at least two children. The root of the tree is denoted by* $\textsc{Root}(\mathcal{T})$*, and the leaf nodes are denoted by* $\textsc{Leaves}(\mathcal{T})$*. Each tree node $t \in \mathcal{T}$ is associated with a* ***mode set*** *$\mu(t) \subseteq \mathbb{N}_N$ satisfying the following properties:*

1. *$\mu(\textsc{Root}(\mathcal{T})) = \mathbb{N}_N$.*
2. *For each non-leaf node $t \in \mathcal{T}$, the mode sets of its children partition $\mu(t)$.*
3. *The ith leaf node, denoted by $l_i \in \textsc{Leaves}(\mathcal{T})$, has $\mu(l_i) = \{i\}$.*

For the simplicity of the presentation, we assume without loss of generality that the sequence $l_1, \ldots, l_N$ corresponds to an ordering of leaves obtained from a post-order traversal of the dimension tree. If this is not the case, we can relabel the tensor modes accordingly. We define the *inverse mode set* of a node t as $\mu'(t) = \mathbb{N}_N \setminus \mu(t)$. For each node t with a parent $P(t)$, $\mu(t) \subset \mu(P(t))$ holds due to the second property, which yields $\mu'(t) \supset \mu'(P(t))$.



(a) Tucker Decomposition

(b) CP Decomposition

Fig. 1: Tucker and CP decompositions of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$

2.2 Tucker decomposition and its computation using dimension trees

Tucker decomposition expresses a given tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with a smaller core tensor $\boldsymbol{\mathcal{G}}$ multiplied by a *factor matrix* $\mathbf{U}^{(n)}$ of size $I_n \times R_n$ in each mode $n \in \mathbb{N}_N$. Here, the tuple $(R_1, \ldots, R_N)$ forms the requested rank of the decomposition across different modes. This decomposition is denoted as $[\![\boldsymbol{\mathcal{G}}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$ which expresses (or approximates) $\boldsymbol{\mathcal{X}}$ as $\boldsymbol{\mathcal{G}} \times_1 \mathbf{U}^{(1)} \times_2 \cdots \times_N \mathbf{U}^{(N)}$. For example, if $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, then in its Tucker decomposition $[\![\boldsymbol{\mathcal{G}}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]$ we have $x_{i,j,k} \approx \sum_{p=1}^{R_1} \sum_{q=1}^{R_2} \sum_{r=1}^{R_3} g_{p,q,r} u^{(1)}_{i,p} u^{(2)}_{j,q} u^{(3)}_{k,r}$. In Figure 1a, we provide an illustration of the Tucker decomposition of a three dimensional tensor.

A well-known algorithm for computing Tucker decomposition is called Higher Order Orthogonal Iteration (HOOI) [22], which is shown in Algorithm 1. In this algorithm, factor matrices are initialized first. This initialization can be done randomly or using higher-order SVD of the input tensor $\boldsymbol{\mathcal{X}}$ [22]. Then, the "repeat-until" loop applies the ALS method. Here, for each mode n, $\boldsymbol{\mathcal{X}} \times_{i \neq n} \mathbf{U}^{(i)^T}$ is computed at Line 4. This produces a tensor of size $R_1 \times R_2 \times \cdots \times R_{n-1} \times I_n \times R_{n+1} \times \cdots \times R_N$, which is then matricized along the nth mode into the matrix $\mathbf{Y}_{(n)} \in \mathbb{R}^{I_n \times \prod_{i \neq n} R_i}$. Next, the leading R_n left singular vectors of $\mathbf{Y}_{(n)}$ are computed to form the new $\mathbf{U}^{(n)}$ at Line 5. After all matrices $\mathbf{U}^{(n)}$ are updated, the core tensor $\boldsymbol{\mathcal{G}}$ is formed at Line 6, and the fit $(|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{G}}|)/|\boldsymbol{\mathcal{X}}|$ is measured to check convergence at the end of each iteration.

In computing Tucker decomposition using Algorithm 1, performing successive TTMs to compute the resulting tensor $\boldsymbol{\mathcal{Y}}$ constitutes the most expensive step in each ALS subiteration. This step involves the multiplication of $\boldsymbol{\mathcal{X}}$ with the set $\{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\} \setminus \mathbf{U}^{(n)}$ of matrices to eventually update $\mathbf{U}^{(n)}$ at the end of the subiteration for mode n. In the literature [14], an efficient algorithmic scheme was proposed to perform this step faster with the following observation. For a 4-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$, in the first two ALS subiterations, one needs to compute $\boldsymbol{\mathcal{X}} \times_2 \mathbf{U}^{(2)^T} \times_3 \mathbf{U}^{(3)^T} \times_4 \mathbf{U}^{(4)^T}$ and $\boldsymbol{\mathcal{X}} \times_1 \mathbf{U}^{(1)^T} \times_3 \mathbf{U}^{(3)^T} \times_4 \mathbf{U}^{(4)^T}$, then update the corresponding matrices $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$, respectively. Note that in doing so, $\mathbf{U}^{(3)}$ and $\mathbf{U}^{(4)}$ remains unchanged, which brings about the possibility of computing the intermediate

Algorithm 1 HOOI: ALS algorithm for computing Tucker decomposition

Input: $\mathcal{X}$: An N-mode tensor
$R_1, \ldots, R_N$: The rank of Tucker decomposition
Output: $[\![\mathcal{G}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$: A rank $(R_1, \ldots, R_N)$ Tucker decomposition of $\mathcal{X}$
1: Initialize the matrix $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n \in \mathbb{N}_N$
2: **repeat**
3: **for** $n = 1, \ldots, N$ **do**
4: $\mathcal{Y} \leftarrow \mathcal{X} \times_{i \neq n} \mathbf{U}^{(i)^T}$
5: $\mathbf{U}^{(n)} \leftarrow R_n$ leading left singular vectors of $\mathbf{Y}_{(n)}$
6: $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{U}^{(1)^T} \times_2 \cdots \times_N \mathbf{U}^{(N)^T}$
7: **until** convergence or the maximum number of iterations
8: **return** $[\![\mathcal{G}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$

tensor $\mathcal{Z} = \mathcal{X} \times_3 \mathbf{U}^{(3)^T} \times_4 \mathbf{U}^{(4)^T}$, then reusing this partial result in the first and second subiterations as $\mathcal{Z} \times_2 \mathbf{U}^{(2)^T}$ and $\mathcal{Z} \times_1 \mathbf{U}^{(1)^T}$, respectively. This computation stays valid as one can perform TTM in a set of distinct modes in any order. This way, however, the number of performed TTMs reduces significantly.

For an N-dimensional tensor, we are interested in identifying and reusing such common partial results as much as possible in a systematical way. This is achieved by using a dimension tree $\mathcal{T}$ as follows [14]. A tensor $\mathcal{X}^{(t)}$ is associated with each node $t \in \mathcal{T}$, which corresponds to the multiplication $\mathcal{X} \times_{i \in \mu'(t)} \mathbf{U}^{(i)^T}$. Note that this implies $\mathcal{X}^{(\text{ROOT}(\mathcal{T}))} = \mathcal{X}$, and $\mathcal{X}^{(l_n)} = \mathcal{X} \times_{i \neq n} \mathbf{U}^{(i)^T}$ for all $n \in \mathbb{N}_N$. With this tree structure, the tensor of any non-root tree node t can be computed from that of its parent as $\mathcal{X}^{(t)} = \mathcal{X}^{(P(t))} \times_{i \in \mu(P(t)) \setminus \mu(t)} \mathbf{U}^{(i)^T}$. Indeed, if $\mathcal{X}^{(P(t))}$ does not exist, it needs to be similarly computed from its parent's tensor. We provide the algorithm for computing the tensor of any tree node t in Algorithm 2.

Algorithm 2 DTREE-TTM: Performing TTM on a dimension tree

Input: t: A dimension tree node
Output: $\mathcal{X}^{(t)}$: The tensor of t
1: **if** EXISTS($\mathcal{X}^{(t)}$) **then**
2: **return** $\mathcal{X}^{(t)}$
3: $\mathcal{X}^{(t)} \leftarrow$ DTREE-TTM($P(t)$)
4: **for** $d \in \mu(P(t)) \setminus \mu(t)$ **do**
5: $\mathcal{X}^{(t)} \leftarrow \mathcal{X}^{(t)} \times_d \mathbf{U}^{(d)^T}$
6: **return** $\mathcal{X}^{(t)}$

In Algorithm 3, we provide the HOOI algorithm that performs TTMs using a dimension tree $\mathcal{T}$ and the TTM routine in Algorithm 2. This is the algorithm employed for computing the Tucker decomposition of sparse tensors [14], yet the framework is equally applicable to dense tensors. In the ALS subiteration for mode n, the algorithm starts at Line 6 with destroying all tensors in the tree that involves a multiplication with $\mathbf{U}^{(n)}$, as $\mathbf{U}^{(n)}$ will subsequently be

updated in that subiteration, invalidating these tensors. Note that every tree node t that does not lie in the path from l_n to $\text{ROOT}(\mathcal{T})$ has $n \in \mu'(t)$, hence their tensors are destroyed. Next, DTREE-TTM is performed for the leaf node l_n at Line 7 to compute $\mathcal{Y} = \mathcal{X} \times_{i \neq n} \mathbf{U}^{(i)^T}$. Algorithm 2 only computes tensors of nodes in the path from l_n to $\text{ROOT}(\mathcal{T})$, and all other tensors not lying on this path are already destroyed at Line 6; therefore, at any instant of Algorithm 3, only the tensors of nodes lying on a path from a leaf to the root are stored. Following the computation of TTM for l_n, $\mathbf{U}^{(n)}$ is updated by performing a truncated SVD on $\mathbf{Y}_{(n)}$ at Line 8. After all N ALS subiterations, the core tensor $\mathcal{G}$ is formed by multiplying $\mathbf{U}^{(N)}$ with the last $\mathcal{Y}$ corresponding to $\mathcal{X} \times_{i \neq N} \mathbf{U}^{(i)^T}$ at Line 9.

Algorithm 3 DTREE-HOOI: Dimension tree-based HOOI algorithm

Input: $\mathcal{X}$: An N-mode tensor
$R_1, \ldots, R_N$: The rank of Tucker decomposition
$\mathcal{T}$: A dimension tree for N dimensions with leaves $l_1, \ldots, l_N$
Output: $[\![\mathcal{G}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$: A rank $(R_1, \ldots, R_N)$ Tucker decomposition of $\mathcal{X}$

1: Initialize the matrix $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n \in \mathbb{N}_N$
2: **repeat**
3: **for** $n = 1, \ldots, N$ **do**
4: **for all** $t \in \mathcal{T}$ **do**
5: **if** $n \in \mu'(t)$ **then**
6: DESTROY$(T(t))$ ▶ Destroy all tensors that are multiplied by $\mathbf{U}^{(n)}$.
7: $\mathcal{Y} \leftarrow$ DTREE-TTM(l_n) ▶ Perform TTM for the leaf node.
8: $\mathbf{U}^{(n)} \leftarrow R_n$ leading left singular vectors of $\mathbf{Y}_{(n)}$
9: $\mathcal{G} \leftarrow \mathcal{Y} \times_N \mathbf{U}^{(N)^T}$ ▶ Form the core tensor.
10: **until** convergence or the maximum number of iterations
11: **return** $[\![\mathcal{G}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$

Since $l_1, \ldots, l_N$ corresponds to a post-order traversal of the tree, the tensor of each tree node is computed when DTREE-TTM is called for its first leaf descendant node, stays valid and reused throughout the subiterations of all its leaf descendants, and is destroyed in the subiteration following its last leaf descendant. This implies that the tensor of every tree node is computed and destroyed exactly once per HOOI iteration.

Kaya and Uçar employ dimension trees in computing the Tucker decomposition of sparse tensors [14]. Here, a balanced binary dimension tree is used in order to upper bound the number of TTMs performed in a HOOI iteration by $O(N \log N)$, and the number of allocated tree tensors at any instant of the algorithm by $O(\log N)$. The actual computational and memory requirements are difficult to determine as they depend on the sparsity of the input tensor, and this scheme provides a reliable coarse upper bound on the computational and memory utilization in computing sparse Tucker decomposition. For the dense case, however, this is no longer pertinent for two reasons. First, tensors of intermediate tree nodes are expected to get dramatically smaller as they are obtained by performing TTMs on the original tensor in dimensions with

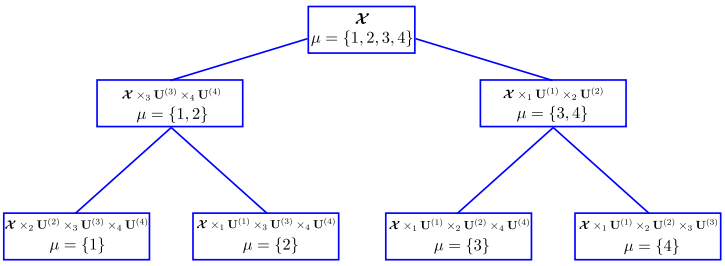


Fig. 2: A binary dimension tree for computing HOOI where each node is associated with a tensor and a mode set (indicated as μ). Each node's tensor is obtained by TTMs using $\boldsymbol{\mathcal{X}}$ and factor matrices in every dimension that are not in the mode set of the node.

$R_n \ll I_n$, rendering their memory cost negligible in practice. Second, once the structure of the dimension tree is determined, one can easily compute the exact computational cost due to all operations being dense multilinear algebra computations. For this reason, Choi et al. [8] investigate the use of an optimal dimension tree for computing dense Tucker decomposition, and propose a dynamic programming algorithm which finds an optimal binary tree in $O(4^N)$ time using $O(3^N)$ memory, which is further investigated in Section 3 where an improved algorithm with $O(3^N)$ and $O(2^N)$ time and space complexities is proposed.

2.3 CP decomposition and its computation using dimension trees

The rank-R CP-decomposition of a tensor $\boldsymbol{\mathcal{X}}$ expresses or approximates it as a sum of R rank-1 tensors. For instance, for $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, one writes $\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \mathbf{u}_r^{(3)}$ where $\mathbf{u}_r^{(1)} \in \mathbb{R}^{I_1}$, $\mathbf{u}_r^{(2)} \in \mathbb{R}^{I_2}$, and $\mathbf{u}_r^{(3)} \in \mathbb{R}^{I_3}$ respectively represent the columns of factors $\mathbf{U}^{(1)} = [\mathbf{u}_1^{(1)}, \ldots, \mathbf{u}_R^{(1)}]$, $\mathbf{U}^{(2)} = [\mathbf{u}_1^{(2)}, \ldots, \mathbf{u}_R^{(2)}]$, and $\mathbf{U}^{(3)} = [\mathbf{u}_1^{(3)}, \ldots, \mathbf{u}_R^{(3)}]$. In this case, the element-wise approximation (or equality) becomes $x_{i,j,k} \approx \sum_{r=1}^{R} u_{i,r}^{(1)} u_{j,r}^{(2)} u_{k,r}^{(3)}$. The minimum R value rendering this approximation an equality is called the *rank* (or CP-rank) of the tensor $\boldsymbol{\mathcal{X}}$, and computing this rank is NP-hard [12]. In Figure 1b, we provide an illustration of the CP decomposition of a three dimensional tensor.

The standard algorithm for computing CP decomposition is the ALS method (CP-ALS), which establishes a good trade-off between convergence rate (number of iterations) and cost per iteration [20]. It is an iterative algorithm, shown in Algorithm 4, that progressively updates the factors $\mathbf{U}^{(n)}$ in an alternating fashion staring from an initial guess. CP-ALS continues until it can no longer improve the solution, or it reaches the allowed maximum number of iterations. Similarly to the Tucker case, in CP-ALS factor matrices can be initialized randomly or using the truncated SVD of the matricizations of $\boldsymbol{\mathcal{X}}$ [20], and each

iteration consists of N *subiterations*, where in the nth subiteration $\mathbf{U}^{(n)}$ is updated using $\boldsymbol{\mathcal{X}}$ as well as the current values of all other factor matrices.

Algorithm 4 CP-ALS: ALS algorithm for computing CP decomposition

Input: $\boldsymbol{\mathcal{X}}$: An N-mode tensor
R: The rank of CP decomposition
Output: $[\![\boldsymbol{\lambda}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$: A rank-R CP decomposition of $\boldsymbol{\mathcal{X}}$
1: Initialize the matrix $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n \in \mathbb{N}_N$
2: **repeat**
3: **for** $n = 1, \ldots, N$ **do**
4: $\mathbf{M}^{(n)} \leftarrow \mathbf{X}_{(n)}(\odot_{i \neq n} \mathbf{U}^{(i)})$
5: $\mathbf{H}^{(n)} \leftarrow *_{i \neq n}({\mathbf{U}^{(i)}}^T \mathbf{U}^{(i)})$
6: $\mathbf{U}^{(n)} \leftarrow \mathbf{M}^{(n)} {\mathbf{H}^{(n)}}^\dagger$ ▶ ${\mathbf{H}^{(n)}}^\dagger$ is the pseudo-inverse of $\mathbf{H}^{(n)}$.
7: $\boldsymbol{\lambda} \leftarrow$ COLUMN-NORMALIZE($\mathbf{U}^{(n)}$) ▶ Normalize columns and store the norms in $\boldsymbol{\lambda}$.
8: **until** convergence or the maximum number of iterations
9: **return** $[\![\boldsymbol{\lambda}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$

Computing the matrix $\mathbf{M}^{(n)} \in \mathbb{R}^{I_n \times R}$ at Line 4 of Algorithm 4 is the sole part involving the tensor $\boldsymbol{\mathcal{X}}$, and is the most expensive computational step of the CP-ALS algorithm. The operation $\boldsymbol{\mathcal{X}}_{(n)}(\odot_{i \neq n} \mathbf{U}^{(i)})$ is called *matricized tensor-times Khatri-Rao product* (MTTKRP). Here, the Khatri-Rao product of the involved $\mathbf{U}^{(n)}$s defines a matrix of size $(\prod_{i \neq n} I_i) \times R$ according to Equation (3). Although one can matricize $\boldsymbol{\mathcal{X}}$ and form this Khatri-Rao product explicitly to carry out this operation, other methods are proposed in the literature that enable performing MTTKRP with asymptotically same computational cost without forming the Khatri-Rao product. One such formulation [3] expresses MTTKRP in terms of a series of TTVs and computes the resulting matrix $\mathbf{M}^{(n)}$ column by column. With this formulation, the rth column of $\mathbf{M}^{(n)}$ can be computed using $N-1$ TTVs as in

$$\mathbf{m}_r^{(n)} \leftarrow \boldsymbol{\mathcal{X}} \times_{i \neq n} \mathbf{u}_r^{(i)}, \tag{4}$$

Once $\mathbf{M}^{(n)}$ is obtained, the Hadamard product of the matrices ${\mathbf{U}^{(i)}}^T \mathbf{U}^{(i)} \in \mathbb{R}^{R \times R}$ is computed for $i \in \mathbb{N}_N \setminus \{n\}$ to form the matrix $\mathbf{H}^{(n)} \in \mathbb{R}^{R \times R}$. Note that within the subiteration n, only $\mathbf{U}^{(n)}$ is updated among all factor matrices. Therefore, for efficiency, one can precompute all matrices ${\mathbf{U}^{(i)}}^T \mathbf{U}^{(i)}$ of size $R \times R$ for $1 \leq i \leq N$, then update ${\mathbf{U}^{(n)}}^T \mathbf{U}^{(n)}$ once $\mathbf{U}^{(n)}$ changes. As the rank R of approximation is typically much smaller than the tensor dimensions I_n in practice, performing these Hadamard products to compute $\mathbf{H}^{(n)}$ and the matrix-matrix multiplication to compute ${\mathbf{U}^{(n)}}^T \mathbf{U}^{(n)}$ become relatively cheap compared with the TTV step. Once both $\mathbf{M}^{(n)}$ and $\mathbf{H}^{(n)}$ are computed, another matrix-matrix multiplication is performed using $\mathbf{M}^{(n)}$ and the pseudo-inverse of $\mathbf{H}^{(n)}$ in order to update the matrix $\mathbf{U}^{(n)}$, which is not expensive as R is small. Finally, $\mathbf{U}^{(n)}$ is normalized column-wise, and the norms of column vectors are stored in the vector $\boldsymbol{\lambda} \in \mathbb{R}^R$. The convergence is achieved when the

relative improvement in the error norm, i.e., $\boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \lambda_r (\mathbf{u}_r^{(1)} \circ \cdots \circ \mathbf{u}_r^{(N)})$, is small. The cost of this computation is insignificant hence we skip its details.

We have already shown how dimension trees reduce the computational cost by storing and reusing common partial TTM results used across different subiterations of HOOI. In (4), there is a TTV formulation of the expensive MTTKRP step in CP-ALS, which similarly enables storing and reusing partial TTV results, e.g., for a 4-dimensional tensor $\boldsymbol{\mathcal{X}}$, one can similarly compute $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} \times_3 \mathbf{u}_r^{(3)} \times_4 \mathbf{u}_r^{(4)}$, then use this partial result to obtain both $\mathbf{m}_r^{(1)} = \boldsymbol{\mathcal{Z}} \times_2 \mathbf{u}_r^{(2)}$ and $\mathbf{m}_r^{(2)} = \boldsymbol{\mathcal{Z}} \times_1 \mathbf{u}_r^{(1)}$. One difference in this case is that a series of R TTVs is needed to compute all columns of $\mathbf{M}^{(1)}$ and $\mathbf{M}^{(2)}$. For this reason, each tree node t holds tensors $\boldsymbol{\mathcal{X}}_r^{(t)}$ for each $r \in \mathbb{N}_R$ corresponding to the partial result for the rth TTV, and we denote the set of all such tensors as $\boldsymbol{\mathcal{X}}_{:}^{(t)}$. For the root node, all tensors are identical and equal to the original tensor $\boldsymbol{\mathcal{X}}$, i.e., $\boldsymbol{\mathcal{X}}_r^{(\text{ROOT}(\mathcal{T}))} = \boldsymbol{\mathcal{X}}^{(\text{ROOT}(\mathcal{T}))} = \boldsymbol{\mathcal{X}}$ for $r \in \mathbb{N}_R$, as they involve no TTVs, i.e., $\mu'(\text{ROOT}(\mathcal{T})) = \emptyset$.

This idea is used for computing the CP decomposition of sparse tensors [18]. To the best of our knowledge its application to dense tensors is considered in this paper for the first time. Though one can compute $\boldsymbol{\mathcal{X}}_r^{(t)}$ as a series of $|\mu(P(t)) \setminus \mu(t)|$ TTVs [18], for the dense case we introduce the following modification for better efficiency. TTV is a special case of tensor contractions, and the aforementioned scheme [18] effectuates $|\mu(P(t)) \setminus \mu(t)|$ tensor-vector contractions. Each contraction involves a matricization step followed by a matrix-vector multiplication step, and the cost of both these steps are linear with the size of the tensor. We instead employ the following alternative that carries out all TTVs in a single tensor-tensor contraction step. We first form the vector $\mathbf{v} = \otimes_{n \in \mu(P(t)) \setminus \mu(t)} \mathbf{u}_r^{(n)}$ corresponding to the vectorization of the rank-1 tensor $\circ_{\mu(P(t)) \setminus \mu(t)} \mathbf{u}_r^{(n)}$, then matricize the tensor $\boldsymbol{\mathcal{X}}_r^{(P(t))}$ to a matrix $\mathbf{Y} \in \mathbb{R}^{(\prod_{n \in \mu(t)} I_n) \times (\prod_{n \in \mu(P(t)) \setminus \mu(t)} I_n)}$. Finally, we perform the matrix-vector multiplication $\text{vec}(\boldsymbol{\mathcal{X}}_r^{(t)}) \leftarrow \mathbf{Yv}$ to obtain the tensor $\boldsymbol{\mathcal{X}}_r^{(t)}$. Here, the cost of matricization and matrix-vector multiplication is the same as the same costs for the *first* tensor-vector contraction in the previous scenario, and the costs for the subsequent contractions are thereby avoided. In this scenario, we do calculate in addition an outer product of vectors with a cost $O(\prod_{n \in \mu(P(t)) \setminus \mu(t)} I_n)$, yet this cost is expected to be significantly less than the cost of matricizations (which involve strided memory accesses, whereas the outer product can be realized with a contiguous access pattern) in the subsequent contractions of the previous scenario.

We provide the method for performing TTVs using a dimension tree in Algorithm 5 for a tree node t. The algorithm similarly returns the tensors $\boldsymbol{\mathcal{X}}_{:}^{(t)}$ if they are already computed. Otherwise, DTREE-TTV is called at Line 3 to obtain the parent $P(t)$'s tensors. Each $\boldsymbol{\mathcal{X}}_r^{(t)}$ is computed using the corresponding parent tensor $\boldsymbol{\mathcal{X}}_r^{(P(t))}$. Here, we matricize the tensor, compute the Kronecker product of vectors to be multiplied, and finally execute the matrix-vector multiplication. For computing each tensor $\boldsymbol{\mathcal{X}}_r^{(t)}$, this incurs $\prod_{n \in \mu(P(t))} I_n$ opera-

Algorithm 5 DTREE-TTV: Performing TTV on a dimension tree

Input: t: A dimension tree node
Output: $\mathcal{X}_{:}^{(t)}$: The tensors of t
1: **if** EXISTS($\mathcal{X}_{:}^{(t)}$) **then**
2: **return** $\mathcal{X}_{:}^{(t)}$
3: $\mathcal{X}_{:}^{(P(t))} \leftarrow$ DTREE-TTV($P(t)$)
4: **for** $r \in \mathbb{N}_R$ **do**
5: $\mathcal{Y} \leftarrow \mathcal{X}_{r}^{(P(t))}$
6: $\text{vec}(\mathcal{X}_{r}^{(t)}) \leftarrow \mathbf{Y}_{(\mu(t))}(\otimes_{i \in \mu(P(t)) \setminus \mu(t)} \mathbf{u}_r^{(i)})$
7: **return** $\mathcal{X}_{:}^{(t)}$

tions for matricizing the tensor $\mathcal{X}_{r}^{(P(t))}$, $\prod_{n \in \mu(P(t)) \setminus \mu(t)} I_n$ operations for performing the Kronecker product of vectors, and finally $\prod_{n \in \mu(P(t))} I_n$ operations for multiplying the matricized tensor with the Khatri-Rao product, resulting in the overall cost $R(2\prod_{n \in \mu(P(t))} I_n + \prod_{n \in \mu(P(t)) \setminus \mu(t)} I_n)$ for computing all R tensors. Note that this cost reduces to $(R+1)\prod_{n \in \mu(P(t))} I_n + R\prod_{n \in \mu(P(t)) \setminus \mu(t)} I_n$ when the parent node is the root, since one matricization suffices in this case due to all tensors $\mathcal{X}_{:}^{(\text{ROOT}(\mathcal{T}))}$ being the same.

Algorithm 6 DTREE-CP-ALS: Dimension tree-based CP-ALS algorithm

Input: $\mathcal{X}$: An N-mode tensor
R: The rank of CP decomposition
$\mathcal{T}$: A dimension tree for N dimensions with leaves $l_1, \ldots, l_N$
Output: $[\boldsymbol{\lambda}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]$: A rank-$R$ CP decomposition of $\mathcal{X}$
1: Initialize the matrix $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n \in \mathbb{N}_N$
2: **for** $n = 2 \ldots N$ **do**
3: $\mathbf{W}^{(n)} \leftarrow {\mathbf{U}^{(n)}}^T \mathbf{U}^{(n)}$
4: **repeat**
5: **for** $n = 1, \ldots, N$ **do**
6: **for all** $t \in \mathcal{T}$ **do**
7: **if** $n \in \mu'(t)$ **then**
8: DESTROY($\mathcal{X}_{:}^{(t)}$) ▶ Destroy all tensors that are multiplied by $\mathbf{U}^{(n)}$.
9: DTREE-TTV(l_n) ▶ Perform the TTV for the leaf node.
10: **for** $r = 1 \ldots R$ **do**
11: $\mathbf{M}^{(n)}(:, r) \leftarrow \mathcal{X}_{r}^{(l_n)}$ ▶ Form rth column by the rth tensor of the leaf.
12: $\mathbf{H}^{(n)} \leftarrow *_{i \neq n} \mathbf{W}^{(i)}$
13: $\mathbf{U}^{(n)} \leftarrow \mathbf{M}^{(n)} {\mathbf{H}^{(n)}}^{\dagger}$
14: $\boldsymbol{\lambda} \leftarrow$ COLUMN-NORMALIZE($\mathbf{U}^{(n)}$)
15: $\mathbf{W}^{(n)} \leftarrow {\mathbf{U}^{(n)}}^T \mathbf{U}^{(n)}$
16: **until** converge or the maximum number of iterations
17: **return** $[\![\boldsymbol{\lambda}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$

In Algorithm 6, we provide the complete CP-ALS algorithm using dimension trees to execute MTTKRPs, which was introduced in the literature for sparse tensors [18]. First, ${\mathbf{U}^{(n)}}^T \mathbf{U}^{(n)}$ is precomputed for all dimensions except the first at Line 3. The main subiteration for a dimension n begins with destroying all tensors in the tree that involve multiplication with $\mathbf{U}^{(n)}$ at Line 8,

as this matrix will be updated within the iteration, rendering such tensors invalid. Next, DTREE-TTV is called for the leaf node, whose tensors are then used to form the columns of the MTTKRP results $\mathbf{M}^{(n)}$ at Line 11. $\mathbf{H}^{(n)}$ is computed using the precomputed $\mathbf{W}$ matrices, and is multiplied with $\mathbf{M}^{(n)}$ to form the final factor matrix $\mathbf{U}^{(n)}$ at Lines 12 and 13. $\mathbf{U}^{(n)}$ is finalized by column-wise normalization, and the $\mathbf{W}^{(n)}$ is updated using the new $\mathbf{U}^{(n)}$ at Line 15.

Similar to the case of Tucker decomposition, Kaya and Uçar investigate the use of balanced binary dimension trees in computing the CP decomposition of sparse tensors [18]. This work retains the bounds $O(RN \log N)$ and $O(R \log N)$ (note that there are R tensors per node in the CP case) on the number of performed TTVs per iteration and intermediate tree tensors stored, respectively, and shows a dimension tree-based sparse tensor data structure to hold tensor indices using $N \log N$ index arrays to carry out all sparse operations on the tree. In the dense case, however, a balanced tree is not necessarily the best choice, and one can search for an optimal dimension tree minimizing the computational cost. Despite not using dimension trees, the idea of reusing partial TTVs was also exploited in another work [28] in which the mode set $\mathbb{N}_N$ is partitioned into two disjoint subsets $\mathcal{S}_1$ and $\mathcal{S}_2$, and two partial TTV results $\boldsymbol{\mathcal{X}} \times_{s \in \mathcal{S}_1} \mathbf{u}_r^{(s)}$ and $\boldsymbol{\mathcal{X}} \times_{s \in \mathcal{S}_2} \mathbf{u}_r^{(s)}$ are computed and reused. A full dimension tree-based computation generalizes this approach to minimize the computational cost, which is investigated in detail in Section 4 for finding an optimal scheme.

In the following sections, we provide theorems and algorithms regarding an optimal dimension tree for minimizing the TTM and MTTKRP costs in computing Tucker and CP decompositions, respectively. The next section investigates finding an optimal dimension tree for Tucker decomposition, and the subsequent section involves the same analysis for CP decomposition.

3 Finding an optimal dimension tree for computing dense Tucker decomposition

In computing Tucker decomposition using Algorithm 2, one indeterminacy is the order of TTMs performed at Line 5. The order of these multiplications can have a significant impact on the overall computational cost. For minimizing this cost, we propose the following optimal greedy ordering algorithm that minimizes the cost of a series of TTMs in distinct modes, which is demonstrated in the following theorem.

Theorem 1 (Optimal TTM order)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ be a tensor and $D = \{d_1, \ldots, d_{|D|}\}, D \subseteq \mathbb{N}_N$ represent the set of dimensions in which $\boldsymbol{\mathcal{X}}$ is to be multiplied with corresponding matrices in the set $\{\mathbf{M}_1, \ldots, \mathbf{M}_{|D|}\}$ having corresponding sizes, i.e., $\mathbf{M}_i \in \mathbb{R}^{I_{d_i} \times R_i}$. Let $\beta_i = R_i/(1 - R_i/I_{d_i})$, and suppose w.l.g that $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_{|D|}$. Then, the total cost of TTM is minimized with the multiplication order $\boldsymbol{\mathcal{X}} \times_{d_1} \mathbf{M}_1 \times_{d_2} \mathbf{M}_2 \times_{d_3} \cdots \times_{d_{|D|}} \mathbf{M}_{|D|}$.*

Proof. Let $\alpha_i = R_i/I_{d_i}$ for $i \in \mathbb{N}_{|D|}$ and $\pi = \prod_{n\in\mathbb{N}_N} I_n$ be the number of elements in $\boldsymbol{\mathcal{X}}$. Then, the cost of the multiplication in this order becomes $\pi(R_1 + \alpha_1 R_2 + \alpha_1\alpha_2 R_3 + \cdots + \alpha_1 \dots \alpha_{N-1} R_N)$ according to the formulation in (2).

Consider the permutation σ of the list $(d_1, \dots, d_{|D|})$ of dimensions that minimizes the cost of the multiplication $\boldsymbol{\mathcal{X}} \times_{\sigma(1)} \mathbf{M}_{\sigma(1)} \times_{\sigma(2)} \cdots \times_{\sigma(N)} \mathbf{M}_{\sigma(N)}$. This cost is

$$C_\sigma = \pi(R_{\sigma(1)} + \alpha_{\sigma(1)} R_{\sigma(2)} + \alpha_{\sigma(1)}\alpha_{\sigma(2)} R_{\sigma(3)} + \cdots + \alpha_{\sigma(1)} \cdots \alpha_{\sigma(N-1)} R_{\sigma(N)}).$$

Assume by contradiction that $\sigma \neq (d_1, \dots, d_{|D|})$, and let i be the first index such that $\beta_{\sigma(i)} > \beta_{\sigma(i+1)}$. We rewrite the cost C_σ as $C_\sigma = C_1 + C_2 + C_3$ where

- $C_1 = \sum_{j=1}^{i-1} (\prod_{k=1}^{j-1} \alpha_{\sigma(k)}) R_{\sigma(j)}$ corresponds to the first $i-1$ terms;
- $C_2 = (\prod_{k=1}^{i-1})(R_{\sigma(i)} + \alpha_{\sigma(k)} R_{\sigma(i+1)})$ corresponds to terms i and $i+1$;
- $C_3 = \sum_{j=i+1}^{n} (\prod_{k=1}^{j-1} \alpha_{\sigma(k)}) R_{\sigma(j)}$ corresponds to the last $n-i-1$ terms.

Now let us permute dimensions $\sigma(i)$ and $\sigma(i+1)$. Formally, this amounts to replacing σ by $\sigma^* = \tau_{i,i+1} \circ \sigma$, where $\tau_{i,i+1}$ is the transposition that exchanges elements in position i and $i+1$. The costs C_1 and C_3 are not modified, so that $C_{\sigma^*} = C_1 + C_2^* + C_3$, with

$$C_2^* = (\prod_{k=1}^{i-1})(R_{\sigma(i+1)} + \alpha_{\sigma(i+1)} R_{\sigma(i)}).$$

But

$$\begin{aligned} C_2^* < C_2 &\Leftrightarrow R_{\sigma(i+1)} + \alpha_{\sigma(i+1)} R_{\sigma(i)} < R_{\sigma(i)} + \alpha_{\sigma(i)} R_{\sigma(i+1)} \\ &\Leftrightarrow R_{\sigma(i+1)}(1 - \alpha_{\sigma(i)}) < R_{\sigma(i)}(1 - \alpha_{\sigma(i+1)}) \\ &\Leftrightarrow \beta_{\sigma(i+1)} < \beta_{\sigma(i)}. \end{aligned}$$

Hence, permuting dimensions $\sigma(i)$ and $\sigma(i+1)$ does decrease the optimal cost, the desired contradiction. This concludes the proof. □

Note that as a result, we can precompute β values in all $|D|$ dimensions and sort these dimensions accordingly to obtain the optimal TTM order in $O(|D| \log |D|)$ time.

An important point in Algorithm 3 is that the dimension tree $\mathcal{T}$ can be of any shape (balanced, unbalanced, k-ary), and can employ arbitrary partitions of dimensions at each level. Indeed, each tree configuration yields a different computational cost for TTMs; hence, we are interested in finding a tree topology that minimizes this cost. Intuitively, we argue that using a binary tree is a good choice due to the following observation. For a non-leaf node $t \in \mathcal{T}$ with children $t_1, \dots, t_k$ ($k \geq 2$), $\mu(t)$ is partitioned into k disjoint sets, namely $\mu(t_1), \dots, \mu(t_k)$. Performing Dtree-Ttm on each child t_l for $l \in \mathbb{N}_k$ requires TTM in dimensions $\mu'(t_l) = \mu(t) \setminus \mu(t_l)$. Overall, calling Dtree-Ttm on all k children requires $k-1$ TTMs for each dimension in $\mu(t)$, which is minimized for $k = 2$ using a binary tree. Nevertheless, in some extreme cases, using a binary tree may not be optimal in terms of actual computational cost, despite avoiding such extra TTMs, and we provide such a counter-example using a

6-dimensional tensor in Appendix A. In most practical scenarios, however, binary dimension tree (BDT) is expected to provide optimal or close to optimal results; hence, we conduct the analysis using BDT in the rest of the discussion of this section.

We now investigate the computational complexity of finding an optimal BDT that minimizes the TTM cost in Dtree-Hooi. We will show that finding an optimal "balanced" binary dimension tree (BBDT) in which the dimensions $\mu(t)$ of each tree node t are equitably partitioned to its children t_1 and t_2, i.e., $|\mu(t_1)|, |\mu(t_2)| \geq \lfloor |\mu(t)|/2 \rfloor$, is NP-hard, and conjecture that the problem using arbitrary dimension trees still remains NP-hard. In doing so, we consider the Product-Partition problem, which is NP-hard [24]: Given a set $S = \{s_1, \ldots, s_K\}$ of positive integers, find a subset S' of S such that $\max(\prod_{i \in S'} i, \prod_{i \in S \setminus S'} i)$ is minimized. We use a variant of this problem called Balanced-Product-Partition where $|S| = 2K$ is a multiset for $K > 0$, and $|S'| = K$ is another multiset containing exactly a half of the elements of S. This problem still remains NP-hard as one can trivially solve any instance of Product-Partition using an instance of Balanced-Product-Partition with the following polynomial-time reduction. For the given set S of size K for Product-Partition, we create a multiset Q of size $2K$ having all elements of S, and K 1s. Let Q' be the optimal solution for Balanced-Product-Partition of Q, and $S' = Q' \cap S$ be the set of elements of the optimal solution Q' belonging to S. Since the multiset $Q' \setminus S$ can only contain 1s, we get $\prod_{i \in S'} i = \prod_{i \in Q'} i$ as well as $\prod_{i \in S \setminus S'} i = \prod_{i \in Q \setminus Q'}$. Therefore, $\max(\prod_{i \in Q'} i, \prod_{i \in Q \setminus Q'} i)$ is minimized if and only if $\max(\prod_{i \in S'} i, \prod_{i \in S \setminus S'} i)$ is minimized, which makes S' an optimal solution for Product-Partition of S. Note that the objective of Balanced-Product-Partition is also equivalent to minimizing the sum $\prod_{i \in Q'} i + \prod_{i \in Q \setminus Q'} i$, and the same for Product-Partition.

Theorem 2 (Optimal BBDT for Tucker decomposition)**.** *Finding a BBDT that minimizes the cost of TTMs in computing Tucker decomposition is NP-hard.*

Proof. The decision problem corresponding to the BBDT optimization problem obviously belongs to NP. We show the completeness by performing a reduction from Balanced-Product-Partition using a set $S = \{s_1, \ldots, s_{2N}\}$. We start by giving the intuition of the proof and providing some bounds.

We aim to find an optimal BBDT for computing the Tucker decomposition of a $2N+2$-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_{2N+2}}$ using ranks of approximation $R_1, \ldots, R_{2N+2}$. The reduction associates the first $2N$ dimensions of the tensor with the corresponding elements of S, and uses two more auxiliary dimensions. Forming a BBDT for this tensor yields $N+1$ dimensions in both sets $\mu(t_1)$ and $\mu(t_2)$ of the children t_1 and t_2 of the root. The goal is to have exactly N dimensions corresponding to elements of S in both $\mu(t_1)$ and $\mu(t_2)$, and to show that these two subsets of S provide an optimal solution for Balanced-Product-Partition for an optimal BBDT. We achieve this in two steps.

First, we analyze the case where both $\mu(t_1)$ and $\mu(t_2)$ have N dimensions associated with S, and provide lower and upper bounds for the TTM cost of a such BBDT. We show in this case that the cost of the BBDT is minimized when the multiplications of the elements of two associated subsets of S (with $\mu(t_1)$ and $\mu(t_2)$) are balanced (i.e., the sum of two multiplications is minimized), which is effectively the objective of Balanced-Product-Partition. Next, we argue that if a child of the root has $N-1$ (or equivalently, $N+1$) dimensions associated with S, then the BBDT cannot be optimal, exceeding the provided upper bound in the previous case. As both children of the root must have $N+1$ dimensions in total, there would be no other partitioning possibilities regarding these $2N$ dimensions associated with S, and the two subsets of S of size N associated with $\mu(t_1)$ and $\mu(t_2)$ in the former case would provide an optimal solution for Balanced-Product-Partition of S.

We set $R_i = s_i$ and $I_i = 3\prod_{s\in S} s$ for $i \in \mathbb{N}_{2N}$. We use $R_{2N+1} = R_{2N+2} = K_1(I_1)^N = K_1(3\prod_{s\in S} s_i)^N$ and $I_{2N+1} = I_{2N+2} = K_2 R_{2N+1}$ where the coefficients $K_1 \geq 1$ and $K_2 \geq 1$ are left to be determined appropriately in the course of the proof. This yields $\alpha_i = s_i/(3\prod_{s\in S} s) \leq 1/3$ for $i \in \mathbb{N}_{2N}$ which also implies $\beta_i = R_i/(1-\alpha_i) \leq 3s_i/2$. For dimensions $2N+1$ and $2N+2$, we get $\alpha_{2N+1} = \alpha_{2N+2} = K_2^{-1}$, and

$$\begin{aligned}\beta_{2N+1} = \beta_{2N+2} &= R_{2N+1}/(1-\alpha_{2N+1}) \\ &> R_{2N+1} = K_1(3\prod_{s\in S} s)^N \\ &> 3s_{max}/2 \geq \beta_i, i \in N_{2N},\end{aligned}$$

where $s_{max} = \max_{s\in S} s$. Hence, these two dimensions are to be multiplied after dimensions $1 \dots 2N$ in an optimal solution according to Theorem 1.

We start by investigating the cost of a BBDT where dimensions $2N+1$ and $2N+2$ reside in $\mu(t_1)$ and $\mu(t_2)$, respectively. Without loss of generality, suppose that $\mu(t_1) = \{1, \dots, N, 2N+1\}$ and $\mu(t_2) = \{N+1, \dots, 2N, 2N+2\}$, and that $\beta_1 \leq \dots \leq \beta_N$ and $\beta_{N+1} \leq \dots \leq \beta_{2N}$ (which can otherwise be obtained by a proper permutation of dimensions). Therefore, we perform TTMs with $\boldsymbol{\mathcal{X}}$ in the sequence of dimensions $1, \dots, N, 2N+1$ and $N+1, \dots, 2N, 2N+2$ for nodes t_2 and t_1, respectively, as this corresponds to the optimal dimension ordering stated in Theorem 1. Let $\mathcal{C}$ represent the TTM cost of an optimal BBDT with the given $\mu(t_1)$ and $\mu(t_2)$, and $\mathcal{Z} = \prod_{i\in\mathbb{N}_{2N+2}} I_i$ be the number of elements in $\boldsymbol{\mathcal{X}}$. Then, we can express the TTM cost as

$$\begin{aligned}\mathcal{C} = \mathcal{Z}(&s_1 + \alpha_1 s_2 + \dots + \alpha_1 \dots \alpha_{N-1} s_N + \alpha_1 \dots \alpha_N K_1(3\prod_{s\in S} s)^N + \\ &s_{N+1} + \alpha_{N+1} s_{N+2} + \dots + \alpha_{N+1} \dots \alpha_{2N-1} s_{2N} + \alpha_{N+1} \dots \alpha_{2N} K_1(3\prod_{s\in S} s)^N) + \\ &C(1, \dots, N, 2N+1) + C(N+1, \dots, 2N, 2N+2)),\end{aligned} \tag{5}$$

where the first two summands correspond to the TTM cost due to nodes t_2 and t_1, whereas $C(1, \dots, N, 2N+1)$ and $C(N+1, \dots, 2N, 2N+2)$ denote the

total TTM cost of all remaining nodes in the subtrees rooted at t_1 and t_2, respectively, in an optimal BBDT. We rewrite (5) as follows

$$\begin{aligned}\mathcal{C} = \mathcal{Z}K_1[K_1^{-1}(&s_1 + \alpha_1 s_2 + \cdots + \alpha_1 \dots \alpha_{N-1} s_N + \\ &s_{N+1} + \alpha_{N+1} s_{N+2} + \cdots + \alpha_{N+1} \dots \alpha_{2N-1} s_{2N}) + \\ &s_1 \dots s_N + s_{N+1} \dots s_{2N}] + \\ &C(1, \dots, N, 2N+1) + C(N+1, \dots, 2N, 2N+2),\end{aligned} \tag{6}$$

□

and obtain a trivial lower bound

$$\mathcal{C} > \mathcal{Z}K_1(s_1 \dots s_N + s_{N+1} \dots s_{2N}), \tag{7}$$

as the rest of the summands are positive. Next, we aim find an upper bound for $\mathcal{C}$ by bounding the costs $C(1, \dots, N, 2N+1)$ and $C(N+1, \dots, 2N, 2N+2)$ as following. $C(1, \dots, N, 2N+1)$ corresponds to the TTM cost using a BBDT having $N+1$ leaves (and at most $2N$ nodes in total excluding its root t_1). The number of elements in the tensor at the root of this sub-tree is $\mathcal{Z}_1 = \mathcal{Z}\alpha_{N+1} \dots \alpha_{2N} K_2^{-1}$ as the tensor is obtained from the multiplication of $\boldsymbol{\mathcal{X}}$ in all modes in $\mu(t_2)$. Each node can require at most one TTM per each dimension in $\mu(t_1)$, and the cost of the TTM in a dimension d cannot exceed $\mathcal{Z}_1 R_d$ since the tensor cannot grow larger after multiplications ($R_d \leq I_d$). Therefore, we obtain the following upper bound on the total TTM cost $C(1, \dots, N, 2N+1)$:

$$\begin{aligned}C(1, \dots, N, 2N+1) &\leq 2N\mathcal{Z}_1 \Big(\sum_{i=1}^{N} R_i + R_{2N+1}\Big) \\ &< \mathcal{Z}\alpha_{N+1} \dots \alpha_{2N} K_2^{-1} 2N \Big(\sum_{i=1}^{2N+2} R_i\Big).\end{aligned}$$

Finally, setting $K_2 = 2N(\sum_{i=1}^{2N+2} R_i)$ yields

$$C(1, \dots, N, 2N+1) < \mathcal{Z}\alpha_{N+1} \dots \alpha_{2N}.$$

We similarly obtain the upper bound $\mathcal{Z}\alpha_1 \dots \alpha_N$ for $C(N+1, \dots, 2N, 2N+2)$. Using these upper bounds in (6) gives

$$\begin{aligned}\mathcal{C} < \mathcal{Z}K_1\big(&K_1^{-1}(s_1 + \alpha_1 s_2 + \cdots + \alpha_1 \dots \alpha_{N-1} s_N + \\ &s_{N+1} + \alpha_{N+1} s_{N+2} + \cdots + \alpha_{N+1} \dots \alpha_{2N-1} s_{2N} + \\ &\alpha_{N+1} \dots \alpha_{2N} + \alpha_1 \dots \alpha_N) + \\ &s_1 \dots s_N + s_{N+1} \dots s_{2N}\big).\end{aligned}$$

Since $\alpha_i \leq 1/3$ for $i \in \mathbb{N}_{2N}$, setting $K_1 = (4s_{max} + 2)$ yields the final upper bound

$$\mathcal{C} < \mathcal{Z}K_1(s_1 \dots s_N + s_{N+1} \dots s_{2N} + 1). \tag{8}$$

Next, we analyze the case where dimensions $2N+1$ and $2N+2$ reside in the same child of the root, namely t_1. Suppose w.l.g that $\mu(t_1) = \{1, \dots, N-1, 2N+1, 2N+2\}$ and $\mu(t_2) = \{N, \dots, 2N\}$. Then, the TTM cost for t_2 becomes

$$\begin{aligned}
\mathcal{C}_{t_2} &= \mathcal{Z}(s_1 + \alpha_1 s_2 + \dots + \alpha_1 \dots \alpha_{N-2} s_{N-1} + \\
&\quad \alpha_1 \dots \alpha_{N-1} K_1 (3 \prod_{s \in S} s)^N + \alpha_1 \dots \alpha_{N-1} K_2^{-1} K_1 (3 \prod_{s \in S} s)^N) \\
&> \mathcal{Z} \alpha_1 \dots \alpha_{N-1} K_1 (3 \prod_{s \in S} s)^N \\
&= \mathcal{Z} K_1 (s_1 \dots s_{N-1} 3 \prod_{s \in S} s) \\
&\geq \mathcal{Z} K_1 (s_1 \dots s_N + s_{N+1} \dots s_{2N} + 1),
\end{aligned}$$

which already exceeds the upper bound in (8). Therefore, we conclude that dimensions $2N+1$ and $2N+2$ cannot reside in the same child of the root in an optimal solution. As a result, we obtain exactly N dimensions associated with S in $\mu(t_1)$ and $\mu(t_2)$ in an optimal solution.

Using this result and the bounds in (7) and (8) we can finally perform a reduction from the decision version of Balanced-Product-Partition: Given a set S of $2N$ positive integers, is there a $S' \subset S, |S'| = N$ such that $\prod_{s \in S'} s + \prod_{s \in S \setminus S'} s \leq \mathcal{C}$ for some $\mathcal{C} \geq 1$? We claim that such S' exists if and only if there exists a BBDT constructed in the aforementioned manner whose cost is smaller than $\mathcal{Z} K_1 (\mathcal{C}+1)$. If a such S' exists, then (8) suggests that one can construct a corresponding BBDT whose cost is less than $\mathcal{Z} K_1 (\mathcal{C}+1)$. On the contrary, if there exists no such S', then any S' yields $\prod_{s \in S'} s + \prod_{s \in S \setminus S'} s \geq \mathcal{C}+1$, in which case the cost of the associated BBDT exceeds $\mathcal{Z} K_1 (\mathcal{C}+1)$ due to (7), which concludes the proof.

3.1 An algorithm for finding an optimal BDT for HOOI

In this section, we provide an algorithm that finds an optimal BDT for minimizing the TTM cost of HOOI for $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ using ranks of approximation $R_1, \dots, R_N$. The main idea of the algorithm is to compute the cost of an optimal tree for smaller subsets of dimensions, then to use these solutions to construct an optimal tree for bigger subsets in a dynamic programming formulation, which is detailed in what follows.

For any subset $S = \{s_1, \dots, s_{|S|}\}$ of $\mathbb{N}_N$, let $\mathcal{C}_{ttm}(S)$ denote the optimal TTM cost of multiplying an $|S|$-dimensional tensor of size $I_{s_1} \times \dots \times I_{s_{|S|}}$ in all dimensions with the corresponding matrices $\mathbf{U}^{(s_1)}, \dots, \mathbf{U}^{(s_{|S|})}$ in HOOI. Note that the cost of the optimal ordering for these $|S|$ TTMs can be determined in $O(|S| \log |S|)$ time using the algorithm provided in Theorem 1. Next, we let $\boldsymbol{\mathcal{X}}_S$ denote an N-dimensional tensor obtained by multiplying $\boldsymbol{\mathcal{X}}$ in dimensions $\mathbb{N}_N \setminus S$, which has the size I_s in a dimension s if $s \in S$ and the size R_s

Algorithm 7 BDT-OPT-HOOI: Algorithm for finding the cost of an optimal BDT for HOOI

Input: S: Subset of dimensions for which the cost of an optimal BDT is to be found
Output: $\mathcal{C}_{tree}(S)$: The cost of an optimal BDT involving dimensions in S
1: **if** $\mathcal{C}_{tree}(S) \neq -1$ **then** ▶ $\mathcal{C}_{tree}(S)$ is already computed.
2: **return** $\mathcal{C}_{tree}(S)$
3: **for all** $S' \subset S$ **do**
4: $c = \rho(S)\pi(S \setminus S')\mathcal{C}_{ttm}(S') + \rho(S)\pi(S')\mathcal{C}_{ttm}(S \setminus S')$
5: $\mathcal{C}_{tree}(S) = \min(\mathcal{C}_{tree}(S), \text{BDT-OPT-HOOI}(S') + \text{BDT-OPT-HOOI}(S \setminus S') + c)$
6: **return** $\mathcal{C}_{tree}(S)$

otherwise. We also let $\mathcal{C}_{tree}(S)$ denote the TTM cost of an optimal BDT for an iteration of HOOI using $\boldsymbol{\mathcal{X}}_S$. Note that $\mathcal{C}_{tree}(S) = 0$ if $|S| \leq 1$. We use the notation $\pi(S) = \prod_{s \in S} I_s$ and $\rho(S) = \prod_{s \in \mathbb{N}_N \setminus S} R_s$ to denote the size of $\boldsymbol{\mathcal{X}}_S$ in multiplied and non-multiplied dimensions, in which case $\boldsymbol{\mathcal{X}}_S$ has $\pi(S)\rho(S)$ elements in total. Then, we can express $\mathcal{C}_{tree}(S)$ with the following recurrence relation:

$$\mathcal{C}_{tree}(S) = \min_{S' \in 2^S \setminus \emptyset} \big(\mathcal{C}_{tree}(S') + \mathcal{C}_{tree}(S \setminus S') + \rho(S)\pi(S \setminus S')\mathcal{C}_{ttm}(S') + \rho(S)\pi(S')\mathcal{C}_{ttm}(S \setminus S')\big). \tag{9}$$

This is because in an optimal BDT for S whose root t has two children, namely t_1 and t_2, with mode sets S' and $S \setminus S'$, both subtrees rooted at t_1 and t_2 should be optimal BDTs respectively for the sets S' and $S \setminus S'$ of dimensions (otherwise replacing these subtrees with an optimal one would reduce the overall cost). The cost of performing TTMs in an optimal order for obtaining the tensors of t_1 and t_2 are respectively given in the third and the fourth summands. Here, the TTM costs $\mathcal{C}_{ttm}(S')$ and $\mathcal{C}_{ttm}(S \setminus S')$ are scaled with the size of $\boldsymbol{\mathcal{X}}_S$ in the remaining modes. Note that there are $2^{|S|}$ possible entries for π, ρ, $\mathcal{C}_{ttm}$ in this formulation. As each entry of these functions can be computed in $O(|S| \log |S|)$ time, we can precompute these two tables in $O(2^{|S|}|S| \log |S|)$ time and using $O(2^{|S|})$ memory. This way, each entry $\mathcal{C}_{tree}(S)$ can be computed and stored in $O(2^S)$ time using the solutions to the subproblems in (9). As a result, by considering all subsets of $\mathbb{N}_N$, the computational cost of finding the optimal BDT for the original tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1, \ldots, I_N}$ becomes

$$\binom{N}{0}2^N + \binom{N}{1}2^{N-1} + \binom{N}{2}2^{N-2} + \cdots + \binom{N}{N}2^0 = 3^N.$$

We have in addition $O(2^N N \log N)$ and $O(2^N)$ computational and memory costs, respectively, for constructing the auxiliary tables described above. Note in comparison that the cost of first TTM involving the root node is bounded by $\Omega(\prod_{i=1}^N I_i R_{min})$ where $R_{min} = \min_{i \in \mathbb{N}_i} R_i$, and for an execution of HOOI with K iterations this yields the overall lower bound $\Omega(K \prod_{i=1}^N I_i R_{min})$ for the computational cost. In practice, this lower bound easily exceeds $O(3^N)$ as $I_i \gg 3$ in most practical scenarios [8], rendering the cost of finding an optimal tree negligible. In the extreme cases where most dimension sizes consist of

2s and 3s so that this lower bound is comparable to 3^N, one can heavily prune the possible bi-partitions S and $S \setminus S'$ in (9) as most dimensions have the same sizes, and thereby reduce the cost significantly. Similarly, the cost of storing the tensor, $O(\prod_{i=1}^{N} I_i)$, is always bounded by $\Omega(2^N)$, the cost of storing the subproblems in the dynamic programming formulation. We provide the algorithm for finding the cost of an optimal BDT $\mathcal{C}_{tree}(S)$ for any subset S of dimensions of $\boldsymbol{\mathcal{X}}$ in Algorithm 7. We expect the lookup tables $\mathcal{C}_{ttm}(.)$, $\pi(.)$, and $\rho(.)$ to be computed beforehand as mentioned.

Once all entries of $\mathcal{C}_{tree}$ are computed, one can make another top-down pass over $\mathcal{C}_{tree}$ to construct the actual BDT for a set S as follows. We first find the subset S' yielding the optimal cost $\mathcal{C}_{tree}(S)$ in (9), which yields the partition of dimensions for the children of the root node. Then, we recursively determine the rest of the tree by iterating over $\mathcal{C}_{tree}(S')$ and $\mathcal{C}_{tree}(\mathbb{N}_N \setminus S')$ in the same manner. Executing this procedure starting from $\mathcal{C}_{tree}(\mathbb{N}_N)$ enables constructing an optimal BDT similarly in $O(3^N)$ time, and we use $O(N^2)$ more space, since a BDT has $O(N)$ nodes each having $O(N)$ elements in its dimension set. This gives the overall time and space complexities $O(3^N)$ and $O(2^N)$, respectively, to find an optimal BDT for HOOI. We finally note that the formulation (9) can trivially be extended to consider k-ary splittings of dimensions as well (where each node may have up to k children), yielding an overall time and space complexities $O((k+1)^N)$ and $O(2^N)$, which enables searching optimal k-ary dimension trees albeit with a notable increase in the computational cost.

4 Finding an optimal dimension tree for computing dense CP decomposition

Similar to dimension tree-based HOOI algorithm, the structure of the dimension tree plays a crucial role in the computational cost of MTTKRPs in CP-ALS, and we are interested in finding an optimal tree structure minimizing this cost. In the following part, we provide two theorems pertaining to the computation of an optimal dimension tree for CP-ALS.

Theorem 3. *For any N-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, except $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{2\times 2\times 2}$, an optimal dimension tree that minimizes the TTV cost in the execution of CP-ALS must be binary.*

Proof. The proof is by construction of a BDT in the following manner. We take any dimension tree $\mathcal{T}$ which is not binary, focus on its any subtree rooted at a node t having $K > 2$ children, and finally replace this subtree with another having $K - 1$ children and a smaller cost. This ensures that a non-binary dimension tree cannot be optimal, as any such tree can be transformed into a BDT having less cost using a sequence of such transformations.

Let $t_1, \ldots, t_K$ be the children of the root t, and $\pi(m) = \prod_{n \in \mu(m)} I_n$ denote the size of the tensor of a tree node m. Let also $\mathcal{C}(t)$ be the TTV cost of an optimal dimension tree rooted at t, excluding the cost of the root. We first

consider the case where t is not the root of $\mathcal{T}$. We can express the cost $\mathcal{C}_k$ of the original subtree with K children as

$$\mathcal{C}_k = R\big(2K\pi(t) + \sum_{k=1}^{K} \frac{\pi(t)}{\pi(t_k)}\big) + \sum_{k=1}^{K} \mathcal{C}(t_k), \tag{10}$$

where, for each dimension $k \in \mathbb{N}_K$, $R\frac{\pi(t)}{\pi(t_k)}$ is the Kronecker product cost, and $2R\pi(t)$ corresponds to the total cost of matricization and the multiplication of all R tensors with this product.

We now consider a variant of this tree where the nodes t_1 and t_2 are joint using an auxiliary node t_{12}, which is made a child of t with $\mu(t_{12}) = \mu(t_1) \cup \mu(t_2)$ (thus $\pi(t_{12}) = \pi(t_1)\pi(t_2)$). The rest of the children $t_3, \ldots, t_K$ of t are kept the same to obtain a subtree whose root t has $K-1$ children and the following cost:

$$\begin{aligned}\mathcal{C}_{k-1} =& R\big(2(K-1)\pi(t) + \sum_{i=3}^{K} \frac{\pi(t)}{\pi(t_i)} + \frac{\pi(t)}{\pi(t_{12})} \\ &+ 4\pi(t_{12}) + \frac{\pi(t_{12})}{\pi(t_1)} + \frac{\pi(t_{12})}{\pi(t_2)}\big) + \sum_{i=1}^{K} \mathcal{C}(t_i).\end{aligned} \tag{11}$$

Let $L = \frac{\pi(t)}{\pi(t_1)\pi(t_2)}$. By taking the difference of (10) and (11) we finally obtain

$$\begin{aligned}\mathcal{C}_k - \mathcal{C}_{k-1} =& R\big(2\pi(t) + \frac{\pi(t)}{\pi(t_1)} + \frac{\pi(t)}{\pi(t_2)} - \frac{\pi(t)}{\pi(t_1)\pi(t_2)} \\ &- 4\pi(t_1)\pi(t_2) - \pi(t_2) - \pi(t_1)\big) \\ \geq& R\big(\frac{\pi(t)}{\pi(t_1)} + \frac{\pi(t)}{\pi(t_2)} - \frac{\pi(t)}{\pi(t_1)\pi(t_2)} - \pi(t_2) - \pi(t_1)\big) \\ =& R\big(\pi(t_2)L + \pi(t_1)L - L - \pi(t_2) - \pi(t_1)\big) \\ =& R\big((L-1)\big(\pi(t_2) + \pi(t_1)\big) - L\big) \\ \geq& R\big(4(L-1) - L\big) \geq 2R,\end{aligned} \tag{12}$$

as $\prod(t_k) \geq 2$ for all $k \in \mathbb{N}_K$, and $L \geq 2$ (since $I_n \geq 2$ for all $n \in \mathbb{N}_N$). Hence, the modified subtree whose root has $K-1$ children always provides a smaller cost in this case.

Next, we consider the case where t is the root of $\mathcal{T}$. In this case, the matricization costs only $\pi(t)$ for the root's tensor, hence the cost of the original tree having K children becomes

$$\mathcal{C}_k = K\pi(t) + R\big(K\pi(t) + \sum_{k=1}^{K} \frac{\pi(t)}{\pi(t_k)}\big) + \sum_{k=1}^{K} \mathcal{C}(t_k). \tag{13}$$

This time, we assume w.l.g that t_1 and t_2 have the two minimum tensor sizes among all children, i.e., $\pi(t_1), \pi(t_2) \leq \pi(t_k)$ for all $k = 3, \ldots, K$, which also

implies that $\pi(t_1), \pi(t_2) \leq L$. For the tree having $K-1$ children, we obtain the following cost:

$$\begin{aligned}\mathcal{C}_{k-1} =&(K-1)\pi(t) + R\big((K-1)\pi(t) + \sum_{i=3}^{K} \frac{\pi(t)}{\pi(t_i)} + \frac{\pi(t)}{\pi(t_{12})} \\ &+ 4\pi(t_{12}) + \frac{\pi(t_{12})}{\pi(t_1)} + \frac{\pi(t_{12})}{\pi(t_2)}\big) + \sum_{i=1}^{K} \mathcal{C}(t_i). \end{aligned} \tag{14}$$

By taking the difference of (13) and (14) we obtain

$$\begin{aligned}\mathcal{C}_k - \mathcal{C}_{k-1} =&\pi(t) + R\big(\pi(t) + \frac{\pi(t)}{\pi(t_1)} + \frac{\pi(t)}{\pi(t_2)} - \frac{\pi(t)}{\pi(t_1)\pi(t_2)} \\ &- 4\pi(t_1)\pi(t_2) - \pi(t_2) - \pi(t_1)\big) \\ =&\pi(t) + R\big(L\pi(t_1)\pi(t_2) + L\pi(t_2) + L\pi(t_1) - L - 4\pi(t_1)\pi(t_2) - \pi(t_2) - \pi(t_1)\big) \\ =&\pi(t) + R\big((L-4)\pi(t_1)\pi(t_2) + (L-1)(\pi(t_1) + \pi(t_2)) - L\big). \end{aligned} \tag{15}$$

For $L \geq 4$, (15) yields

$$\begin{aligned}\mathcal{C}_k - \mathcal{C}_{k-1} >&R\big((L-1)(\pi(t_1) + \pi(t_2)) - L\big) \\ \geq&R\big(4(L-1) - L\big) \geq 8R.\end{aligned}$$

For $L = 3$, we obtain

$$\mathcal{C}_k - \mathcal{C}_{k-1} =\pi(t) + R\big(- \pi(t_1)\pi(t_2) + 2(\pi(t_1) + \pi(t_2)) - 3\big).$$

As $\pi(t_1), \pi(t_1) \leq L = 3$, we only have four possibilities for $\pi(t_1)$ and $\pi(t_2)$. For $\pi(t_1) = \pi(t_2) = 3$, we get

$$\mathcal{C}_k - \mathcal{C}_{k-1} =\pi(t) + R\big(- 9 + 2(6) - 3\big) = \pi(t) > 0.$$

$\pi(t_1) = 2$,$\pi(t_2) = 3$ and $\pi(t_1) = 3$,$\pi(t_2) = 2$ yields

$$\mathcal{C}_k - \mathcal{C}_{k-1} =\pi(t) + R\big(- 6 + 2(5) - 3\big) = \pi(t) + R > 0.$$

The last possibility $\pi(t_1) = 2$ and $\pi(t_2) = 2$ gives

$$\mathcal{C}_k - \mathcal{C}_{k-1} =\pi(t) + R\big(- 4 + 2(4) - 3\big) = \pi(t) + R > 0.$$

Finally, for $L = 2$, (15) becomes

$$\mathcal{C}_k - \mathcal{C}_{k-1} =\pi(t) + R\big(- 3\pi(t_1)\pi(t_2) + \pi(t_1) + \pi(t_2) - 2\big).$$

Note that in this case, the only option is to have $\pi(t_1) = \pi(t_2) = 2$, which also implies that $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{2\times 2\times 2}$, hence $\pi(t) = 8$. As a result, we obtain

$$\mathcal{C}_k - \mathcal{C}_{k-1} =8 + R\big(- 3(4) + 2 + 2 - 2\big) = 8 - 10R < 0$$

for all $R \geq 1$. Therefore, we conclude that for all tensors except those in $\mathbb{R}^{2\times 2\times 2}$, the optimal dimension tree is binary. □

In the next theorem, we show that finding an optimal dimension tree is NP-hard.

Theorem 4 (Optimal dimension tree for CP decomposition)**.** *Finding an optimal dimension tree that minimizes the cost of TTVs in computing CP decomposition is NP-hard.*

Proof. The corresponding decision problem obviously belongs to NP. We perform a reduction from Product-Partition using a set $S = \{s_1, \dots, s_N\}$ of positive integers. We construct an $N+4$-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_{N+4}}$ whose first N dimensions correspond to the elements of S, i.e., $I_n = s_n$ for $n \in \mathbb{N}_N$. We let $I_{N+1} = I_{N+2} = I_{N+3} = I_{N+4} = k$ where k is to be determined appropriately in the course of the proof. We similarly name the first N and the last 4 dimensions as *type-1* and *type-2*, respectively.

We now analyze the cost of an optimal dimension tree that minimizes the MTTKRP cost in executing CP-ALS for $\boldsymbol{\mathcal{X}}$. This tree must be a BDT as suggested by Theorem 3. Let t be the root this with children t_1 and t_2. In an optimal BDT, we expect $\mu(t_1)$ to consist of dimensions corresponding to elements in an optimal solution $S' \subset S$ of Product-Partition. All dimensions corresponding to $S \setminus S'$ are similarly expected to belong to $\mu(t_2)$. In this scenario, we would only have three configurations for the partitioning of type-2 dimensions to $\mu(t_1)$ and $\mu(t_2)$ due to symmetry, namely $N+1, N+2, N+3, N+4 \in \mu(t_1)$, $N+1, N+2, N+3 \in \mu(t_1)$ and $N+4 \in \mu(t_2)$, and $N+1, N+2 \in \mu(t_1)$ and $N+3, N+4 \in \mu(t_4)$.

We first consider the case where $N+1, N+2 \in \mu(t_1)$ and $N+3, N+4 \in \mu(t_4)$, and analyze the cost of a BDT for a given partition of type-1 dimensions to $\mu(t_1)$ and $\mu(t_2)$. Without loss of generality, let $\mu(t_1) = \{1, \dots, K, N+1, N+2\}$ and $\mu(t_2) = \{K+1, \dots, N, N+3, N+4\}$ represent a such partition for some $K, 0 \le K \le N$ (all possible partitions can be obtained by a proper K and a permutation of the elements of S). We can express the cost of this BDT as

$$\begin{aligned}\mathcal{C} =& (2R+2)k^4 \prod_{n \in \mathbb{N}_N} s_n + Rk^2 \prod_{n=1}^{K} s_n + Rk^2 \prod_{n=K+1}^{N} s_n \\ &+ C(1, \dots, K, N+1, N+2) + C(K+1, \dots, N, N+3, N+4), \end{aligned} \tag{16}$$

where the first summand is the cost of the matricization and the multiplication of $\boldsymbol{\mathcal{X}}$ with the Khatri-Rao products of corresponding matrices for t_1 and t_2, the second and the third summands correspond to the cost of forming the Khatri-Rao product for t_2 and t_1, respectively. Here, $C(1, \dots, K, N+1, N+2)$ and $C(K+1, \dots, N, N+3, N+4)$ denote the total MTTKRP cost of subtrees rooted at t_1 and t_2, respectively, excluding the cost of t_1 and t_2. Both t_1 and t_2 have two children as $|\mu(t_1)|, |\mu(t_2)| \ge 2$. Each child of t_1 incurs a cost $2Rk^2 \prod_{n=1}^{K} s_n$ for matricizing the tensor of t_1, then multiplying it with the Kronecker products. Similarly, each of two children of t_2 has a cost of at least $2Rk^2 \prod_{n=K+1}^{N} s_n$. With these costs at hand (which exclude the cost of forming

the Kronecker products for the children of t_1 and t_2, and further costs down the tree), we obtain the following lower bound from (16):

$$\mathcal{C} > (2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + 5Rk^2(\prod_{n=1}^{K} s_n + \prod_{n=K+1}^{N} s_n). \tag{17}$$

Next, we aim to find an upper bound for an optimal $C(1,\ldots,K,N+1,N+2)$ and $C(K+1,\ldots,N,N+3,N+4)$, which in turn would yield an upper bound for $\mathcal{C}$. Let t_{11} and t_{12} be the children of t_1. We consider the cost for the case where $N+1 \in \mu(t_{11})$ and $N+2 \in \mu(t_{12})$. In this case, the size of the tensor of t_1 is $Rk^2\prod_{n=1}^{K}$; therefore, cost of both t_{11} and t_{12} are upper bounded by $2Rk^2\prod_{n=1}^{K} + Rk\prod_{n=1}^{K}$ as the cost of forming the Kronecker products cannot exceed $Rk\prod_{n=1}^{K}$ with the given partition of type-2 dimensions. In addition, note that t_1 is the root of a subtree having $K+2$ leaf nodes, and having up to $K+1$ non-leaf nodes each of which has two children; hence, it cannot have more than $2K$ nodes excluding t_1, t_{11}, and t_{12}. These nodes are descendants of either t_{11} or t_{12}; therefore, each of these nodes incurs a cost which cannot exceed $2Rk\prod_{i=1}^{K} s_i + Rk\prod_{i=1}^{K} s_i = 3Rk\prod_{i=1}^{K} s_i$, where the first and the second summands are upper bounds for the cost of the matricization of the tensor and its multiplication with the Kronecker product, and the cost of forming the Kronecker product, respectively. This results in the upper bound $C(1,\ldots,k,N+1,N+2) < 4Rk^2\prod_{n=1}^{K} s_n + (6K+2)Rk\prod_{n=1}^{K} s_n$. We similarly obtain the upper bound $4Rk^2\prod_{n=K+1}^{N} s_n + (6N-6K+2)Rk\prod_{n=K+1}^{N} s_n$ for $C(k+1,\ldots,N,N+3,N+4)$ for the case where $N+3 \in \mu(t_{21})$ and $N+4 \in \mu(t_{22})$. Finally, setting $k = l(6N+4)\prod_{i=1}^{N} s_i$ for any $l \geq 1$, and replacing these two upper bounds in (16) yields

$$\begin{aligned}
\mathcal{C} <& (2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + Rk^2\Big(5\prod_{n=1}^{K} s_n + 5\prod_{n=K+1}^{N} s_n + \\
& k^{-1}((6K+2)\prod_{i=1}^{K} s_i + (6N-6K+2)\prod_{i=K+1}^{N} s_i)\Big) \\
<& (2R+2)Rk^4 \prod_{n\in\mathbb{N}_N} s_n + 5Rk^2(\prod_{n=1}^{K} s_n + \prod_{n=k+1}^{N} s_n + 1).
\end{aligned} \tag{18}$$

Next, we analyze the MTTKRP cost using two other partitionings of type-2 dimensions. Without loss of generality, we only consider the cases where $\mu_1 = \{1,\ldots,K\}$ and $\mu_2 = \{K+1,\ldots,N+1,N+2,N+3,N+4\}$, and $\mu_1 = \{1,\ldots,K,N+1\}$ and $\mu_2 = \{K+1,\ldots,N,N+2,N+3,N+4\}$. Considering only the cost of t_1 and t_2, the former case yields

$$\mathcal{C} > (2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + R\prod_{n=1}^{k} s_n + Rk^4 \prod_{n=k+1}^{N} s_n, \tag{19}$$

while the latter gives

$$\mathcal{C} > (2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + Rk \prod_{n=1}^{k} s_n + Rk^3 \prod_{n=k+1}^{N} s_n. \tag{20}$$

Setting $l = 5\prod_{i=1}^{N} s_i + 1$ provides

$$\begin{aligned}\mathcal{C} &> (2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + 5Rk^2(6N+2)(\prod_{n=1}^{N} s_n + 1) \\ &> (2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + 5Rk^2(\prod_{n=1}^{K} s_n + \prod_{n=K+1} N s_n + 1)\end{aligned} \tag{21}$$

in both cases, which exceeds the upper bound provided in (18). Therefore, we conclude that these two partitionings cannot provide an optimal BDT.

We can now perform the reduction from the decision version of Product-Partition, knowing that an optimal BDT assigns exactly two dimensions of type-2 to each children t_1 and t_2 of the root t, and has the lower and the upper bounds provided in (17) and (18) with respect to the partitioning of type-1 dimensions to $\mu(t_1)$ and $\mu(t_2)$. We claim that a $S' \subseteq S$ with $\prod_{s\in S'} s + \prod_{s\in S\setminus S'} \leq C$ exists for some $C \geq 1$ if and only if there is a BDT for $\boldsymbol{\mathcal{X}}$ constructed in the aforementioned manner whose MTTKRP cost is smaller than $(2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + 5Rk^2(C+1)$ for any positive integer R. If there exists a such S', then (18) suggests that we can construct a BDT whose cost is inferior to $(2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + 5Rk^2(C+1)$. If $\prod_{s\in S'} s + \prod_{s\in S\setminus S'} \geq C+1$ of all subsets $S' \subseteq S$, then (17) implies that the cost of all BDTs exceed $(2R+2)k^4 \prod_{n\in\mathbb{N}_N} s_n + 5Rk^2(C+1)$, which concludes the proof. □

4.1 An algorithm for finding an optimal BDT for CP-ALS

Here, we adopt the algorithm described in Section 3.1 to find an optimal BDT minimizing the MTTKRP/TTV cost within a CP-ALS iteration. We similarly use $\mathcal{C}_{tree}(S)$ to denote the cost of an optimal BDT. We use $\mathcal{C}_{mat}(S)$ to denote the cost of matricizing the tensor $\boldsymbol{\mathcal{X}}_S$, which equals to $\pi(S)$ if $S = \mathbb{N}_N$ and to $\pi(S)R$ otherwise (as there are R tensors for each internal node). Finally, we use $\mathcal{C}_{ttv}(S) = \pi(S)R$ to denote the cost of performing MTTKRP using $\boldsymbol{\mathcal{X}}_S$, not including the cost of forming the Khatri-Rao product. We can then express the cost of an optimal BDT as

$$\mathcal{C}_{tree}(S) = \min_{S'\in 2^S} \big(\mathcal{C}_{tree}(S') + \mathcal{C}_{tree}(S\setminus S') + 2\mathcal{C}_{mat}(S) + 2\mathcal{C}_{ttv}(S) + \pi(S') + \pi(S\setminus S')\big). \tag{22}$$

where the last two summands represent the cost of forming the Khatri-Rao product. Similar to the HOOI case, using this formulation $\mathcal{C}_{tree}(\mathbb{N}_N)$ can be computed in $O(N2^N + 3^N)$ time using $O(2^N)$ space for tables $\mathcal{C}_{tree}$, $\mathcal{C}_{mat}$, $\mathcal{C}_{ttv}$, and π. The actual tree can be likewise constructed using the procedure in

Section 3.1. We provide the algorithm for finding the optimal BDT for a given subset S of dimensions in Algorithm 8, and we similarly expect that lookup tables $\mathcal{C}_{ttv}(.)$, $\mathcal{C}_{mat}(.)$, and $\pi(.)$ are computed beforehand.

Algorithm 8 BDT-OPT-CPALS: Algorithm for finding the cost of an optimal BDT for CP-ALS

Input: S: Subset of dimensions for which the cost of an optimal BDT is to be found
Output: $\mathcal{C}_{tree}(S)$: The cost of an optimal BDT involving dimensions in S
1: **if** $\mathcal{C}_{tree}(S) \neq -1$ **then** ▶ $\mathcal{C}_{tree}(S)$ is already computed.
2: **return** $\mathcal{C}_{tree}(S)$
3: **for all** $S' \subset S$ **do**
4: $c = 2\mathcal{C}_{mat}(S) + 2\mathcal{C}_{ttv}(S) + \pi(S') + \pi(S \setminus S')$
5: $\mathcal{C}_{tree}(S) = \min(\mathcal{C}_{tree}(S), \text{BDT-OPT-CPALS}(S') + \text{BDT-OPT-CPALS}(S \setminus S') + c)$
6: **return** $\mathcal{C}_{tree}(S)$

Similarly to the HOOI case, the cost of a single MTTKRP involving the original tensor within a CP-ALS iteration is $O(\prod_{i=1}^{N} I_i)$ and is bounded in most practical scenarios by $\Omega(3^N)$, rendering the cost of the optimal dimension tree algorithm negligible in comparison. In terms of memory cost, we still store the original tensor whose size is bounded by $\Omega(2^N)$ as in the previous case.

5 Related Work

Memoization techniques for computing Tucker and CP decompositions are introduced by Baskaran et al. [5] and Phan et al. [28], where they partition dimension sets into two subsets, compute and store TTMs/TTVs corresponding to both these subsets for reutilization. Kaya and Uçar generalize these schemes using dimension trees to hierarchically exploit the memoization, and thereby asymptotically reduce the number of TTV/TTM calls in computing sparse CP [18] and Tucker [14] decompositions from $O(N^2)$ to $O(N \log N)$.

Choi et al. [8] investigate computing dense Tucker decomposition in a distributed memory setting using dimension trees, and propose an algorithm for finding the optimal dimension tree in $O(4^N)$ time using $O(3^N)$ memory, which is further reduced in this work to $O(3^N)$ time using $O(2^N)$ memory. Finally, Li et al. [23] propose a memoization scheme for computing sparse CP decomposition concurrently with the work by Kaya and Uçar [18] where they reduce the amortized number of TTVs to $O(N^{1.5})$ per CP-ALS iteration.

6 Conclusion

In this paper, we investigated optimal tree structures in computing Tucker and CP decomposition of dense tensors with a recently introduced computational scheme which enables re-using common partial TTM and MTTKRP results across different subiterations of the tensor decomposition algorithms. We also

provided the first complexity results for this problem regarding an optimal dimension tree structure minimizing the associated computational costs. In particular, we proved that finding an optimal binary tree to minimize the cost of TTM and MTTKRP operations respectively in Dtree-Hooi and Dtree-Cp-Als algorithms is NP-hard. We further showed that the optimal tree must be binary for TTV, except in one degenerate instance. On the contrary, we provided a counter-example using an optimal ternary tree for TTM. Finally, we provided exact algorithms for finding an optimal BDT in $O(3^N)$ time using $O(2^N)$ space for both Tucker and CP cases. Despite the fact that these algorithms have exponential costs in N, we note that dense tensors also have exponential sizes in N, rendering these algorithms usable in practice. All these results lay the foundation for a complete analysis of the use of dimension trees for dense tensor decompositions.

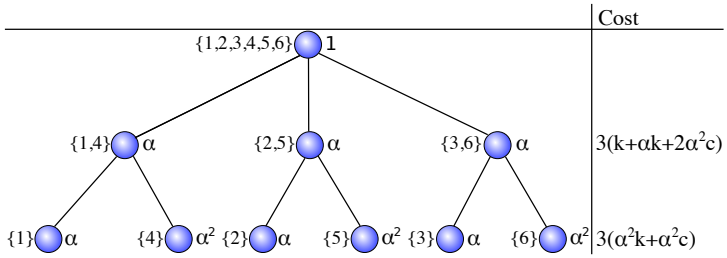
Acknowledgements This research was funded in part by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR). The authors would like to thank Bora Uçar for several discussions. Finally, the authors would like to thank both anonymous reviewers for their comments and suggestions, that helped us to improve this manuscript.

References

1. Acar, E., Dunlavy, D.M., Kolda, T.G.: A scalable optimization approach for fitting canonical tensor decompositions. Journal of Chemometrics **25**(2), 67–86 (2011)
2. Andersson, C.A., Bro, R.: The N-way toolbox for MATLAB. Chemometrics and Intelligent Laboratory Systems **52**(1), 1–4 (2000)
3. Bader, B.W., Kolda, T.G.: Efficient MATLAB computations with sparse and factored tensors. SIAM Journal on Scientific Computing **30**(1), 205–231 (2007)
4. Bader, B.W., Kolda, T.G., et al.: Matlab tensor toolbox version 2.6. Available online (2015)
5. Baskaran, M., Meister, B., Vasilache, N., Lethin, R.: Efficient and scalable computations with sparse tensors. In: Proceedings of the IEEE Conference on High Performance Extreme Computing, HPEC 2012, pp. 1–6 (2012). DOI 10.1109/HPEC.2012.6408676
6. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E.R.H., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI '10, pp. 1306–1313. AAAI Press (2010). URL http://dl.acm.org/citation.cfm?id=2898607.2898816
7. Carroll, D.J., Chang, J.: Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. Psychometrika **35**(3), 283–319 (1970)
8. Chakaravarthy, V.T., Choi, J., Joseph, D.J., Liu, X., Murali, P., Sabharwal, Y., Sreedhar, D.: On optimizing distributed tucker decomposition for dense tensors. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, IPDPS '17. Orlando, FL, USA (2017)
9. Choi, J.H., Vishwanathan, S.V.N.: DFacTo: Distributed factorization of tensors. In: 27th Advances in Neural Information Processing Systems, pp. 1296–1304. Montreal, Quebec, Canada (2014)
10. Grasedyck, L.: Hierarchical singular value decomposition of tensors. SIAM Journal on Matrix Analysis and Applications **31**(4), 2029–2054 (2010)
11. Harshman, R.A.: Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. UCLA Working Papers in Phonetics **16**, 1–84 (1970)

12. Håstad, J.: Tensor rank is np-complete. Journal of Algorithms **11**(4), 644 – 654 (1990)
13. Kang, U., Papalexakis, E., Harpale, A., Faloutsos, C.: GigaTensor: Scaling tensor analysis up by 100 times - Algorithms and discoveries. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, pp. 316–324. ACM, New York, NY, USA (2012)
14. Kaya, O., Uçar, B.: High-performance parallel algorithms for the Tucker decomposition of higher order sparse tensors. Tech. Rep. RR-8801, Inria, Grenoble – Rhône-Alpes (2015)
15. Kaya, O., Uçar, B.: Scalable sparse tensor decompositions in distributed memory systems. Tech. Rep. RR-8722, Inria, Grenoble – Rhône-Alpes (2015)
16. Kaya, O., Uçar, B.: Scalable sparse tensor decompositions in distributed memory systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, pp. 77:1–77:11. ACM, New York, NY, USA (2015). DOI 10.1145/2807591.2807624
17. Kaya, O., Uçar, B.: High performance parallel algorithms for the Tucker decomposition of sparse tensors. In: Proceedings of the 45th International Conference on Parallel Processing, ICPP '16, pp. 103–112 (2016). DOI 10.1109/ICPP.2016.19
18. Kaya, O., Uçar, B.: Parallel CP decomposition of sparse tensors using dimension trees. Research Report RR-8976, Inria - Research Centre Grenoble – Rhône-Alpes (2016)
19. Kolda, T.G., Bader, B.: The TOPHITS model for higher-order web link analysis. In: Proceedings of Link Analysis, Counterterrorism and Security, pp. 26–29 (2006)
20. Kolda, T.G., Bader, B.: Tensor decompositions and applications. SIAM Review **51**(3), 455–500 (2009)
21. Lathauwer, L.D., Moor, B.D.: From matrix to tensor: Multilinear algebra and signal processing. In: Proceedings of the Institute of Mathematics and Its Applications Conference Series, vol. 67, pp. 1–16 (1998)
22. Lathauwer, L.D., Moor, B.D., Vandewalle, J.: A multilinear singular value decomposition. SIAM Journal on Matrix Analysis and Applications **21**(4), 1253–1278 (2000)
23. Li, J., Choi, J., Perros, I., Sun, J., Vuduc, R.: Model-driven sparse CP decomposition for higher-order tensors. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, IPDPS '17, pp. 1048–1057. Orlando, FL, USA (2017)
24. Ng, C., Barketau, M., Cheng, T., Kovalyov, M.Y.: Product partition and related problems of scheduling and systems reliability: Computational complexity and approximation. European Journal of Operational Research **207**(2), 601 – 604 (2010). DOI https://doi.org/10.1016/j.ejor.2010.05.034
25. Nion, D., Mokios, K.N., Sidiropoulos, N.D., Potamianos, A.: Batch and adaptive PARAFAC-based blind separation of convolutive speech mixtures. IEEE Transactions on Audio, Speech, and Language Processing **18**(6), 1193–1207 (2010). DOI 10.1109/TASL.2009.2031694
26. Nion, D., Sidiropoulos, N.D.: Tensor algebra and multidimensional harmonic retrieval in signal processing for mimo radar. IEEE Transactions on Signal Processing **58**(11), 5693–5705 (2010). DOI 10.1109/TSP.2010.2058802
27. Perros, I., Chen, R., Vuduc, R., Sun, J.: Sparse hierarchical Tucker factorization and its application to healthcare. In: Proceedings of the 2015 IEEE International Conference on Data Mining, ICDM 2015, pp. 943–948 (2015)
28. Phan, A.H., Tichavský, P., Cichocki, A.: Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations. IEEE Transactions on Signal Processing **61**(19), 4834–4846 (2013). DOI 10.1109/TSP.2013.2269903
29. Rendle, S., Lars, T.S.: Pairwise interaction tensor factorization for personalized tag recommendation. In: Proceedings of the Third ACM International Conference on Web Search and Data Mining, WSDM '10, pp. 81–90. ACM, New York, NY, USA (2010). DOI 10.1145/1718487.1718498
30. Rendle, S., Leandro, B.M., Nanopoulos, A., Schmidt-Thieme, L.: Learning optimal ranking with tensor factorization for tag recommendation. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '09, pp. 727–736. ACM, New York, NY, USA (2009). DOI 10.1145/1557019.1557100
31. Sidiropoulos, N.D., Bro, R., Giannakis, G.B.: Parallel factor analysis in sensor array processing. IEEE Transactions on Signal Processing **48**(8), 2377–2388 (2000). DOI 10.1109/78.852018

Fig. 3: A ternary dimension tree for $\boldsymbol{\mathcal{X}}$. The μ set of the node is provided on the left, and the tensor size coefficient is provided on the right of each tree node. The TTM cost of each level is provided on the right.



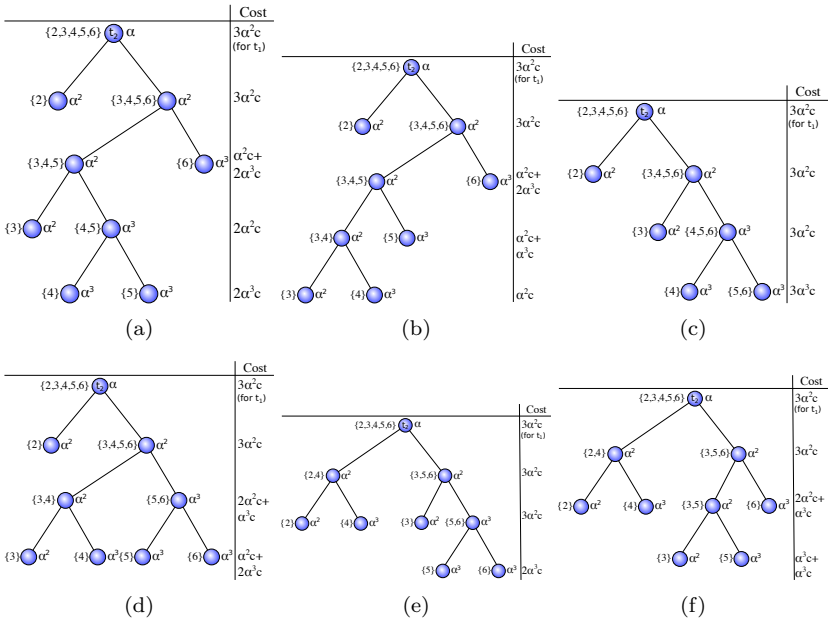
32. Smith, S., Karypis, G.: A medium-grained algorithm for sparse tensor factorization. In: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016, pp. 902–911 (2016)
33. Smith, S., Ravindran, N., Sidiropoulos, N.D., Karypis, G.: SPLATT: Efficient and parallel sparse tensor-matrix multiplication. In: Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium, IPDPS '15, pp. 61–70. IEEE Computer Society, Hyderabad, India (2015)
34. Symeonidis, P., Nanopoulos, A., Manolopoulos, Y.: Tag recommendations based on tensor dimensionality reduction. In: Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys '08, pp. 43–50. ACM, New York, NY, USA (2008). DOI 10.1145/1454008.1454017
35. Vasilescu, M.A.O., Terzopoulos, D.: Multilinear analysis of image ensembles: TensorFaces. In: Computer Vision—ECCV 2002, pp. 447–460. Springer (2002)

A Counter-example: Tensor having no optimal BDT

Here, we provide a counter-example using a 6-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_6}$ to show that using a BDT is not necessarily optimal to compute Tucker decomposition. The first three and the last three dimensions of $\boldsymbol{\mathcal{X}}$ have identical sizes and ranks of approximation. Specifically, we let $I_1 = I_2 = I_3 = k$ and $R_1 = R_2 = R_3 = R < k$ in the first three dimensions, and $I_4 = I_5 = I_6 = R_4 = R_5 = R_6 = c$ in the last three dimensions. We call the first three and the last tree dimensions *type-1* and *type-2 dimensions*, respectively. Note that $\alpha = \alpha_1 = \alpha_2 = \alpha_3 < \alpha_4 = \alpha_5 = \alpha_6 = 1$, and similarly $\beta_1 = \beta_2 = \beta_3 < \beta_4 = \beta_5 = \beta_6 = \infty$, therefore, type-1 dimensions are to be multiplied before type-2 dimensions according to Theorem 1. In Figure 3, we provide a ternary dimension tree with a total cost of $(3+3\alpha+3\alpha^2)k+9\alpha^2 c$. We can choose c arbitrarily large so that the term $9\alpha^2 c$ dominates the cost, and α small enough so that $\alpha^0 c \gg 9\alpha^2 c$ and $\alpha^1 c \gg 9\alpha^2 c$. In this case, any BDT whose cost involves a term of order $\alpha^0 c$ or $\alpha^1 c$ cannot be optimal, having a greater cost than the provided ternary tree.

We now show that any BDT using $\boldsymbol{\mathcal{X}}$ either has a cost with a term of order $\alpha^0 c$ or $\alpha^1 c$, or with a term $9\alpha^2 c + d\alpha^3 c$ with $d \geq 1$; hence, cannot be optimal with a sufficiently large c and a sufficiently small α. We do this by exhaustively considering all possible BDTs and analyzing their costs, while aggressively pruning tree configurations that cannot provide optimality. Luckily, most non-optimal configurations can easily be pruned due to symmetry (as we only have two types of dimensions), leaving us with only a handful of instances to consider. We begin by partitioning type-2 dimensions to the children t_1 and t_2 of the root. There are only two possibilities: $4 \in \mu(t_1)$ and $5, 6 \in \mu(t_2)$, or $4, 5, 6 \in \mu(t_2)$. Since we have only three type-1 dimensions, in the former case either $\mu(t_1)$ or $\mu(t_2)$ will have one or zero type-1 dimension, while the other set having two or three of them. In this case, the TTM

Fig. 4: All possible configurations and associated TTM costs at each level of BDTs rooted at t_2. $\mu(t)$ is given on the left of each tree node t, On the right of a tree node, the size coefficient of its tensor is provided, i.e., t_2 has a tensor of size $(I_1I_2I_3I_4I_5I_6)\alpha$.



cost of the other vertex involves a term $\alpha^0 c$ or $\alpha^1 c$, which already renders this configuration non-optimal. Therefore, we only consider the partition $4, 5, 6 \in \mu(t_2)$, which is the only one that can possibly provide an optimal solution. In this case, there are three possible configurations after partitioning type-1 dimensions: $\mu(t_1) = \{1\}$ and $\mu(t_2) = \{2, 3, 4, 5, 6\}$, $\mu(t_1) = \{1, 2\}$ and $\mu(t_2) = \{3, 4, 5, 6\}$, or $\mu(t_1) = \{1, 2, 3\}$ and $\mu(t_2) = \{4, 5, 6\}$. Note that in the second and third configurations, computing TTM for t_1 involves a term $\alpha^1 c$ and $\alpha^0 c$, respectively, which prevents optimality. Hence, in the rest of the discussion we only consider the first configuration, in which t_1 incurs the TTM cost $3\alpha^2 c$. We focus on the cost of the sub-tree rooted at t_2, and count only the cost due to terms with the coefficient c. Note that if the remaining type-1 dimensions, namely 2 and 3, reside in the same child of t_2, the other child of t_2 incurs a cost of at least $\alpha^1 c$; therefore, 2 and 3 must reside in different children of t_2. In Figure 4 we provide six such possibilities, all of which incur a cost of $9\alpha^2 c + c\alpha^3 c$ with $c \geq 1$. Therefore, we conclude that a BDT cannot be optimal for the given $\boldsymbol{\mathcal{X}}$ for sufficiently small α and large c.