Tree Adaptive Approximation in the Hierarchical Tensor Format

Jonas Ballani* **and Lars Grasedyck**

Bericht Nr. 367 **Juni 2013**

Key words: hierarchical tensor format, hierarchical Tucker format, tensor train, matrix product states, black box approximation, tree adaptivity

AMS Subject Classifications: 15A69, 65F99

Institut für Geometrie und Praktische Mathematik
RWTH Aachen

Templergraben 55, D–52056 Aachen (Germany)

---

* Financial support from the DFG SPP–1324 under grant GR 3179/2-2 gratefully acknowledged.

Tree Adaptive Approximation in the Hierarchical Tensor Format

Jonas Ballani [*], Lars Grasedyck

June 24, 2013

The hierarchical tensor format allows for the low-parametric representation of tensors even in high dimensions d. The efficiency of this representation strongly relies on an appropriate hierarchical splitting of the different directions $1, \dots, d$ such that the associated ranks remain sufficiently small. This splitting can be represented by a binary tree which is usually assumed to be given. In this paper, we address the question of finding an appropriate tree from a subset of tensor entries without any a priori knowledge on the tree structure. We propose an agglomerative strategy that can be combined with rank-adaptive cross approximation techniques such that tensors can be approximated in the hierarchical format in an entirely black box way. Numerical examples illustrate the potential and the limitations of our approach.

1 Introduction

High-dimensional problems are encountered in many areas of practical interest, as e.g. in stochastics, quantum chemistry, or optimization. In many cases, the solution to a high-dimensional problem can be represented or approximated by a *tensor*

$$A \in \mathbb{R}^{n_1 \times \cdots \times n_d}$$

of *order* (or *dimension*) $d \in \mathbb{N}$ with $n_1, \dots, n_d \in \mathbb{N}$. As soon as the order d is large enough, the explicit representation of A in terms of all its entries $A_{(i_1,\dots,i_d)}$, $i_\mu = 1, \dots, n_\mu$, $\mu = 1, \dots, d$, becomes prohibitively expensive. This has motivated the development of data-sparse tensor representations that can be applied even in high dimensions d. For a detailed introduction to tensor representations we refer the reader to [9, 7, 6].

A quite general framework for the low-parametric representation of tensors has been introduced in [8] which we further analyzed in [4]. In the so-called *hierarchical tensor* (or *hierarchical Tucker*) *format*, a tensor is represented by a number of parameters

[*]Financial support from the DFG SPP-1324 under grant GR 3179/2-2 gratefully acknowledged.

that scales only linearly in the dimension d. As a key ingredient, this format relies on an appropriate hierarchy of subspaces which can be related to specific matrix representations of a given tensor. Based on this strong connection to matrices, powerful tools have been developed that allow for (approximate) arithmetic operations with tensors even in high dimensions d.

The efficiency of the hierarchical tensor format crucially depends on an appropriate splitting of the different directions $1, \dots, d$ in a hierarchical way. At the top level $t = D := \{1, \dots, d\}$, the index set t is subdivided into disjoint subsets $t_1, t_2 \subset D$ with $t = t_1 \cup t_2$. Afterwards, the subdivision process is recursively continued with t_1 and t_2 until the bottom level of singletons $t = \{\mu\}$, $\mu \in D$, has been reached. The splitting can be represented by a binary tree T_D which is usually assumed to be given.

Once a tree T_D has been chosen, each node $t \in T_D$ with $t \subset D$ is associated to a rank $k_t \in \mathbb{N}$ that results from a specific matrix representation of A depending on t. It turns out that the storage complexity for the representation of A in the hierarchical tensor format lies in $\mathcal{O}(dk^3 + dkn)$ where $k := \max_{t \in T_D} k_t$, $n := \max_{\mu \in D} n_\mu$. Since the ranks k_t strongly depend on the index sets $t \subset D$, the storage complexity of a given tensor may be very sensitive to the choice of the tree T_D, cf. [5].

In some cases, the complexity for the representation of tensors in the hierarchical format does not depend on the choice of the tree T_D. In particular, any tensor A given in *CP format* (canonical polyadic, CANDECOMP/PARAFAC) with

$$A = \sum_{j=1}^{r} u_{1,j} \otimes \ldots \otimes u_{d,j}, \qquad u_{\mu,j} \in \mathbb{R}^{n_\mu},$$

can be represented in the hierarchical format where all ranks k_t are bounded by r for *any* tree, cf. [8]. Nevertheless, it might be advantageous to represent A in the hierarchical format as one can have $k_t \ll r$ for most $t \in T_D$, cf. [8]. Moreover, the hierarchical representation can be exploited in order to perform efficient (approximate) arithmetics with tensors that do not suffer from the structural weakness of the CP format (non-closedness of the representation, cf. [3]).

A special instance of the hierarchical tensor format is the so-called *TT format* (tensor train, matrix product states) from [12, 10] where the tree T_D is restricted to a linear structure. This means that one only needs to find a suitable permutation of the indices $1, \dots, d$ which facilitates the construction of an appropriate dimension tree. Note, however, that one can construct examples for which there exists a representation in the hierarchical format where all ranks are bounded by $k_t \leq k$ whereas the ranks in the TT format can only be bounded by $k^{\log_2(d)/2-1}$ for *any* permutation, cf. [5].

In this paper, we aim at finding an appropriate tree T_D for a given tensor A without any a priori knowledge on the tree structure. The tensor A is then approximated with respect to T_D up to some (heuristic) target accuracy ε by a black box strategy which we introduced in [1]. Black box approximation strategies tailored to the TT format have also been developed in [11].

Since the number of possible trees scales exponentially in the dimension d, a global optimization of the storage cost over all T_D is mostly too expensive. Instead, we propose

to construct a tree in a bottom up way by a successive agglomeration of disjoint subsets of D. Although this strategy is known from hierarchical clustering algorithms, we cannot directly apply them to our problem since we are not aware of a suitable distance function measuring the closeness of arbitrary subsets $t_1, t_2 \subset D$. As an alternative, we introduce a cluster criterion that completely relies on the ranks k_t which depend on A and the subsets $t \subset D$.

The rest of this paper is organized as follows. In Section 2, we recall the main ingredients of the hierarchical tensor format. In Section 3, we introduce our new clustering strategy based on a successive agglomeration of disjoint subsets $t \subset D$. Since one mostly does not know the ranks k_t in advance, Section 4 is devoted to rank estimation techniques. In Section 5, we shortly analyze the complexity of our strategy in terms of the number of required tensor evaluations. Finally, we study the prospects and limitations of our approach in Section 6 by a number of numerical examples.

2 Hierarchical Tensor Format

Given $d \in \mathbb{N}$ and $n_1, \dots, n_d \in \mathbb{N}$, let $\mathcal{I}_\mu := \{1, \dots, n_\mu\}$ for $\mu = 1, \dots, d$ and define

$$\mathcal{I} := \mathcal{I}_1 \times \cdots \times \mathcal{I}_d.$$

We first introduce a matrix representation of a tensor $A \in \mathbb{R}^{\mathcal{I}}$.

Definition 1 (matricization)**.** Let $D := \{1, \dots, d\}$. Given a subset $t \subset D$ with complement $s := D \setminus t$, the *matricization*

$$\mathcal{M}_t : \mathbb{R}^{\mathcal{I}} \to \mathbb{R}^{\mathcal{I}_t} \otimes \mathbb{R}^{\mathcal{I}_s}, \qquad \mathcal{I}_t := \bigtimes_{\mu \in t} \mathcal{I}_\mu,\ \mathcal{I}_s := \bigtimes_{\mu \in s} \mathcal{I}_\mu,$$

of a tensor $A \in \mathbb{R}^{\mathcal{I}}$ is defined by its entries

$$\mathcal{M}_t(A)_{(i_\mu)_{\mu \in t}, (i_\mu)_{\mu \in s}} := A_{(i_1, \dots, i_d)}, \qquad i_\mu \in \mathcal{I}_\mu,\ \mu \in D.$$

In order to allow for a structured and data-sparse representation of tensors, subsets $t \subset D$ which can be organized as a binary tree are of special interest.

Definition 2 (partition tree, dimension tree)**.** Let J be an arbitrary index set with $\#J < \infty$ and denote its power set by $\mathcal{P}(J)$. A tree $T_J \subset \mathcal{P}(J)$ is called a *partition tree* (for J) if the following three conditions hold:

(a) the index set J is the root of the tree T_J,

(b) all vertices $t \in T_J$ are non-empty subsets $t \subset J$,

(c) every vertex $t \in T_J$ with $\#t \geq 2$ has two sons $t_1, t_2 \in T_J$ with the property

$$t = t_1 \cup t_2, \qquad t_1 \cap t_2 = \emptyset.$$

The set of leaves of T_J is defined by $\mathcal{L}(T_J) := \{t \in T_J : \#t = 1\}$. For all $t \in T_J \setminus \mathcal{L}(T_J)$, we denote the set of sons of t by $\mathrm{sons}(t)$. A partition tree T_D for the particular index set $J = D = \{1, \dots, d\}$ is called a *dimension tree*.

In the next example, we introduce two important special cases of dimension trees.

Example 3. (a) In a balanced binary tree T_D, each node $t \in T_D \setminus \mathcal{L}(T_D)$ with $t = \{\mu_1, \dots, \mu_q\} \subset D$, $q > 1$, has two sons $t_1, t_2 \in T_D$ of the form

$$t_1 = \{\mu_1, \dots, \mu_r\}, \qquad t_2 = \{\mu_{r+1}, \dots, \mu_q\}, \qquad r := \lceil q/2 \rceil.$$

An example for $d = 7$ is depicted in Figure 1. The balanced tree is of minimal depth $\lceil \log_2 d \rceil$.
(b) In the so-called *TT format* introduced in [12, 10], the dimension tree is a simple linear tree, where all nodes $t \in T_D$ are of the form

$$t = \{q\} \quad \text{or} \quad t = \{q, \dots, d\}, \qquad q = 1, \dots, d.$$

An example for $d = 7$ is depicted in Figure 1. The TT tree is of maximal depth $d - 1$.

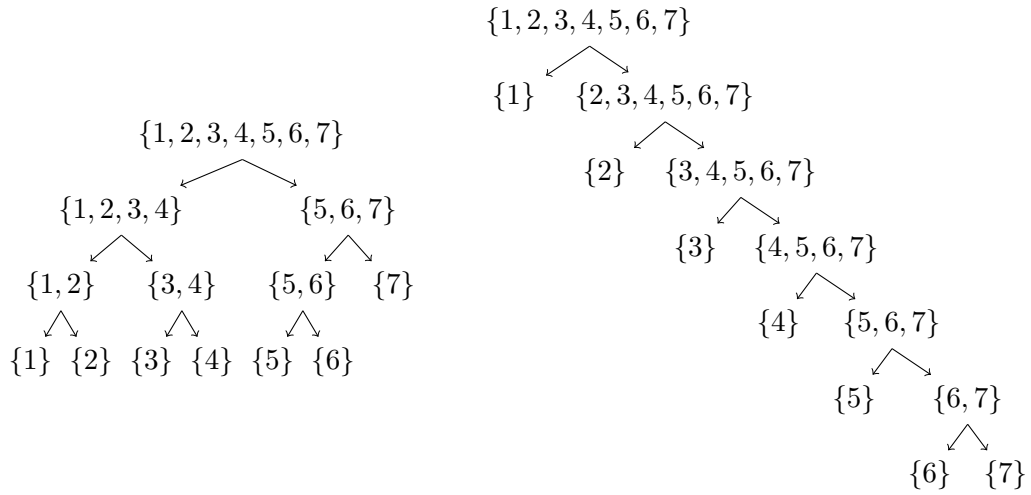

Figure 1: Left: Balanced binary tree. Right: Linear TT tree.

Based on the concept of the matricization of tensors and the definition of a dimension tree, we can now introduce the hierarchical tensor format.

Definition 4 (hierarchical rank, hierarchical format)**.** Let T_D be a dimension tree. The *hierarchical rank* $\boldsymbol{k} := (k_t)_{t \in T_D}$ of a tensor $A \in \mathbb{R}^{\mathcal{I}}$ is defined by

$$k_t := \mathrm{rank}(\mathcal{M}_t(A)), \quad t \in T_D.$$

For a given hierarchical rank $\boldsymbol{k} := (k_t)_{t \in T_D}$, the *hierarchical format* $\mathcal{H}(T_D, \boldsymbol{k})$ is defined by

$$\mathcal{H}(T_D, \boldsymbol{k}) := \{A \in \mathbb{R}^{\mathcal{I}} : \mathrm{rank}(\mathcal{M}_t(A)) \le k_t,\, t \in T_D\}.$$

Given a tensor $A \in \mathcal{H}(T_D, \boldsymbol{k})$, the subspaces $\mathcal{U}_t := \operatorname{image}(\mathcal{M}_t(A)) \subset \mathbb{R}^{\mathcal{I}_t}$, $t \in T_D$, fulfil the so-called *nestedness property*

$$\mathcal{U}_t \subset \mathcal{U}_{t_1} \otimes \mathcal{U}_{t_2}, \qquad t \in T_D \setminus \mathcal{L}(T_D),\ \operatorname{sons}(t) = \{t_1, t_2\}. \tag{1}$$

This allows for a recursive representation of A by the relation

$$(U_t)_{\cdot,j} = \sum_{j_1=1}^{k_{t_1}} \sum_{j_2=1}^{k_{t_2}} (B_t)_{j,j_1,j_2} (U_{t_1})_{\cdot,j_1} \otimes (U_{t_2})_{\cdot,j_2}, \quad j = 1, \dots, k_t,$$

for all $t \in T_D \setminus \mathcal{L}(T_D)$ with $\operatorname{sons}(t) = \{t_1, t_2\}$ where $B_t \in \mathbb{R}^{k_t \times k_{t_1} \times k_{t_2}}$ and $U_t \in \mathbb{R}^{\mathcal{I}_t \times k_t}$ such that $A = (U_D)_{\cdot,1}$.

As a consequence, one only needs to store the matrices $U_t \in \mathbb{R}^{\mathcal{I}_\mu \times k_t}$ in the leaves $t = \{\mu\} \in \mathcal{L}(T_D)$ and the *transfer tensors* $B_t \in \mathbb{R}^{k_t \times k_{t_1} \times k_{t_2}}$ for all inner nodes $t \in T_D \setminus \mathcal{L}(T_D)$ in order to represent a tensor in $\mathcal{H}(T_D, \boldsymbol{k})$. The storage complexity for this representation then sums up to

$$N_{\text{storage}}(\mathcal{H}(T_D, \boldsymbol{k})) := \sum_{\substack{t \in T_D \setminus \mathcal{L}(T_D) \\ \operatorname{sons}(t) = \{t_1, t_2\}}} k_t k_{t_1} k_{t_2} + \sum_{\mu=1}^{d} k_{\{\mu\}} n_\mu. \tag{2}$$

In the special case $n_\mu = n$ for all $\mu = 1, \dots, d$ and $k_t \le k$ for all $t \in T_D$, the storage complexity lies in

$$\mathcal{O}\left(dk^3 + dnk\right).$$

Remark 5. In the TT format, for each node $t \in T_D \setminus \mathcal{L}(T_D)$ with $\operatorname{sons}(t) = \{t_1, t_2\}$ one either has $t_1 \in \mathcal{L}(T_D)$ or $t_2 \in \mathcal{L}(T_D)$. Hence, either $k_{t_1} \le n$ or $k_{t_2} \le n$ which leads to the bound

$$\mathcal{O}\left(dnk^2\right).$$

Note that the last term in (2) does not depend on the choice of the particular tree T_D. This results from the fact that for a given tensor $A \in \mathbb{R}^{\mathcal{I}}$ the rank tuple $(k_\mu)_{\mu \in D}$ with

$$k_\mu := \operatorname{rank}(\mathcal{M}_{\{\mu\}}(A)), \quad \mu \in D,$$

represents the *Tucker rank* of A which does not depend on the splitting of the directions $1, \dots, d$. In order to represent a tensor in the hierarchical format with minimal storage cost one therefore has to solve the following minimization problem.

Problem 6. Let $A \in \mathbb{R}^{\mathcal{I}}$ and let $k_t := \operatorname{rank}(\mathcal{M}_t(A))$ for all $t \subset D$. Among all possible dimension trees T_D find a minimizer of

$$\sum_{\substack{t \in T_D \setminus \mathcal{L}(T_D) \\ \operatorname{sons}(t) = \{t_1, t_2\}}} k_t k_{t_1} k_{t_2}. \tag{3}$$

Unfortunately, the determination of the global optimum of Problem 6 results to be prohibitively expensive since the number of possible dimension trees grows exponentially in the dimension d. We therefore suggest to apply an agglomerative strategy known from hierarchical clustering in order to construct a suitable dimension tree in a bottom up way.

3 Agglomerative Clustering

Hierarchical clustering strategies are used to subdivide a set of data points into nested subsets of similar points (the *clusters*) by means of an appropriately chosen similarity measure. In a top down approach, one starts with the full set of data points which is successively refined until only atomic sets — possibly consisting of single data points — remain. In a bottom up approach, one starts with a partition into clusters of single data points which are successively joined until a single cluster containing the whole set of data points is reached. Typically, the similarity of clusters is measured by a given distance function which is assumed to fulfil the triangle inequality.

For the construction of a dimension tree, it is natural to consider the directions $1, \dots, d$ as data points that need to be clustered in a hierarchical way. This directly leads to the question of either using a top down or a bottom up approach and to the choice of an appropriate similarity measure for clusters $t \subset D := \{1, \dots, d\}$. Since we aim at minimizing the sum (3), we need to take the ranks k_t into account. Note that this means that we cannot simply use strategies known from hierarchical clustering as the clusters $t \subset D$ do not naturally possess properties that fulfil the triangle inequality.

Let $A \in \mathbb{R}^{\mathcal{I}}$ with $k_t := \operatorname{rank}(\mathcal{M}_t(A))$ for all $t \subset D$. What we know in advance is that whenever k_t is small, a cluster $t \subset D$ can be well separated from $D \setminus t$. This means that we can regard two disjoint clusters $t_1, t_2 \subset D$ as closely related whenever the associated rank k_t for $t := t_1 \cup t_2$ is small. Conversely, we can split a given cluster $t \subset D$ into two loosely related clusters t_1, t_2 with $t = t_1 \cup t_2$, $t_1 \cap t_2 = \emptyset$, whenever both k_{t_1} and k_{t_2} are small.

The first observation leads to a bottom up strategy. Starting with initial clusters $\{1\}, \dots, \{d\}$, we successively could try to join a subset of clusters to a new cluster $t \subset D$ such that the associated rank k_t remains small. The second observation would lead to a top down approach. Starting with the single cluster D containing all possible directions, we could try to split a cluster into subclusters such that the associated ranks for all subclusters remain small. However, the top down approach suffers from a severe difficulty. Already for the initial set D, one obtains a number of possible splittings into two disjoint subsets that scales exponentially in the dimension d. We therefore prefer to use an agglomerative bottom up strategy that avoids this exponential complexity.

Pairwise Clustering

Assume that we are given a partion $P = \{t_1, \dots, t_r\}$ of D with clusters $t_\nu \subset D$. In order to keep the sum (3) as small as possible, we would like to combine two clusters $s_1, s_2 \in P$ to a new cluster $s := s_1 \cup s_2$ such that the associated rank k_s is as small as possible. Another strategy is motivated by the nestedness property (1). Since k_{s_1}, k_{s_2} are the dimensions of the subspaces $\mathcal{U}_{s_1}, \mathcal{U}_{s_2}$ from (1), we could measure the quality of the new subspace $\mathcal{U}_s$ by the ratio $k_s/(k_{s_1} k_{s_2})$. We demonstrate the effect of both strategies by the following examples.

Example 7. Let $D := \{1, \dots, 8\}$ and let T_D be a balanced dimension tree as defined in Example 3. Moreover, let $\boldsymbol{k} = (k_t)_{t \in T_D}$ be a rank tuple defined by $k_t := 2$ for all

$t \in T_D \setminus D$, and $k_D := 1$. Assume that $A \in \mathcal{H}(T_D, \boldsymbol{k}) \subset \mathbb{R}^{\mathcal{I}}$, $\mathcal{I} := \{1,2\}^8$, is a random tensor fulfilling $k_t = \operatorname{rank}(\mathcal{M}_t(A))$ for all $t \in T_D$. The dimension tree is visualized in Figure 2 left where we use subscripts in order to indicate the rank k_t at a cluster t. For all subsets $t \subset D$ with $t \notin T_D$, we generically assume that the associated ranks $k_t = \operatorname{rank}(\mathcal{M}_t(A))$ are maximal under the condition that $A \in \mathcal{H}(T_D, \boldsymbol{k})$.

We now apply our agglomerative strategy to A, ignoring our knowledge on the structure of the tree T_D. Starting with the initial partition

$$P_0 := \{\{1\}, \dots, \{8\}\},$$

the lowest rank $k_t = 2$ is obtained by the union $t := t_1 \cup t_2$ of one of the pairs $t_1 = \{2\mu - 1\}, t_2 = \{2\mu\}$ for $\mu \in \{1, \dots, 4\}$. By agglomeration, we find a new partition of the form, e.g.,

$$P_1 := \{\{1,2\}, \{3\}, \dots, \{8\}\}.$$

As the combination of the cluster $\{1,2\}$ with one of the clusters $\{3\}, \dots, \{8\}$ corresponds to a rank of 4, afterwards we would again combine one of the pairs $t_1 = \{2\mu - 1\}, t_2 = \{2\mu\}$ for $\mu \in \{2, \dots, 4\}$. Two possible partitions are therefore given by

$$P_2' := \{\{1,2\}, \{3,4\}, \{5\}, \{6\}, \{7\}, \{8\}\}, \quad P_2'' := \{\{1,2\}, \{3\}, \{4\}, \{5,6\}, \{7\}, \{8\}\}.$$

One easily checks that also in the following steps only clusters are joined that correspond to the original structure of the dimension tree from Figure 2 left. However, since the agglomeration order is not fixed in advance, one may also end up with a tree of the shape shown in Figure 2 right.

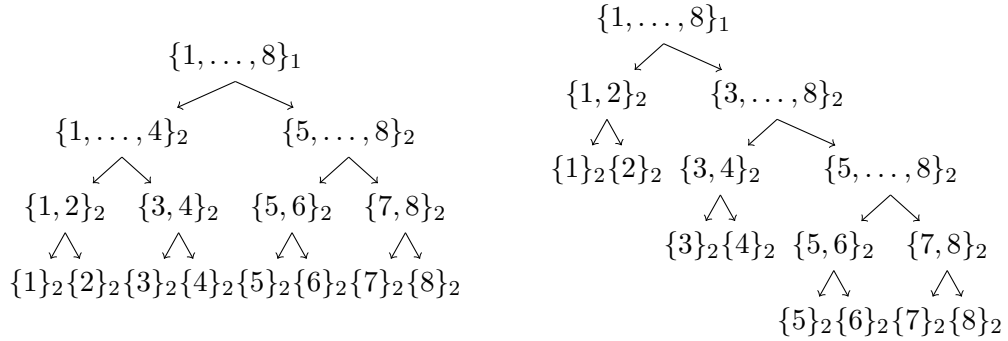

Figure 2: Left: original dimension tree for $d = 8$ with subscripts k_t at clusters t. Right: possible dimension tree after pairwise clustering.

Note that in this example the result of the agglomerative cluster strategy would have been the same if we had used the minimum of the ratios $k_t/(k_{t_1}k_{t_2}) \in \{1/2, 1\}$ as a decision criterion. In the next example, we study what can go wrong for this simple pairwise strategy.

Example 8. Let $D := \{1, \dots, 16\}$ and let T_D be a balanced dimension tree as defined in Example 3. Moreover, let $\boldsymbol{k} = (k_t)_{t \in T_D}$ be a rank tuple defined by

$$k_t := \begin{cases} 2, & t = \{\mu\},\ \mu = 1, \dots, 16, \\ 4, & t = \{2\mu - 1, 2\mu\},\ \mu = 1, \dots, 8, \\ 4, & t = \{4\mu - 3, \dots, 4\mu\},\ \mu = 1, \dots, 4, \\ 4, & t = \{8\mu - 7, \dots, 8\mu\},\ \mu = 1, 2, \\ 1, & t = D. \end{cases}$$

Assume that $A \in \mathcal{H}(T_D, \boldsymbol{k}) \subset \mathbb{R}^{\mathcal{I}}$, $\mathcal{I} := \{1, 2\}^{16}$, is a random tensor fulfilling $k_t = \operatorname{rank}(\mathcal{M}_t(A))$ for all $t \in T_D$. The dimension tree and the ranks k_t are visualized in Figure 3. As before, we assume for all subsets $t \subset D$ with $t \notin T_D$ that the associated ranks $k_t = \operatorname{rank}(\mathcal{M}_t(A))$ are maximal under the condition that $A \in \mathcal{H}(T_D, \boldsymbol{k})$.

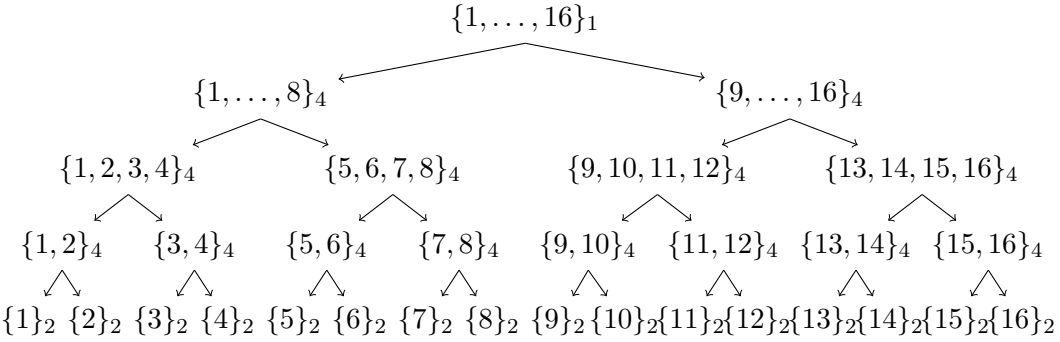

Figure 3: Original dimension tree for $d = 16$ with subscripts k_t.

We now apply our agglomerative strategy to A, ignoring our knowledge on the structure of the tree T_D. Starting with the initial partition

$$P_0 := \{\{1\}, \dots, \{16\}\},$$

all pairs of disjoint clusters $t_1, t_2 \in P_0$ lead to the minimal rank $k_t = 4$, $t := t_1 \cup t_2$. Hence, after eight agglomerative steps, we may have reached a partition of the form

$$P_1 = \{\{\mu, \mu + 8\} : \mu = 1, \dots, 8\}.$$

Now, all combinations of disjoint clusters $t_1, t_2 \in P_1$ lead to the minimal rank $k_t = 16$, $t := t_1 \cup t_2$. After four agglomerative steps, we may have reached a partition of the form

$$P_2 = \{\{\mu, \mu + 4, \mu + 8, \mu + 12\} : \mu = 1, \dots, 4\}.$$

Now, all combinations of disjoint clusters $t_1, t_2 \in P_2$ lead to the minimal rank $k_t = 256$, $t := t_1 \cup t_2$. This results in a possible partition

$$P_3 = \{\{\mu, \mu + 2, \mu + 4, \mu + 6, \mu + 8, \mu + 10, \mu + 12, \mu + 14\} : \mu = 1, 2\}$$

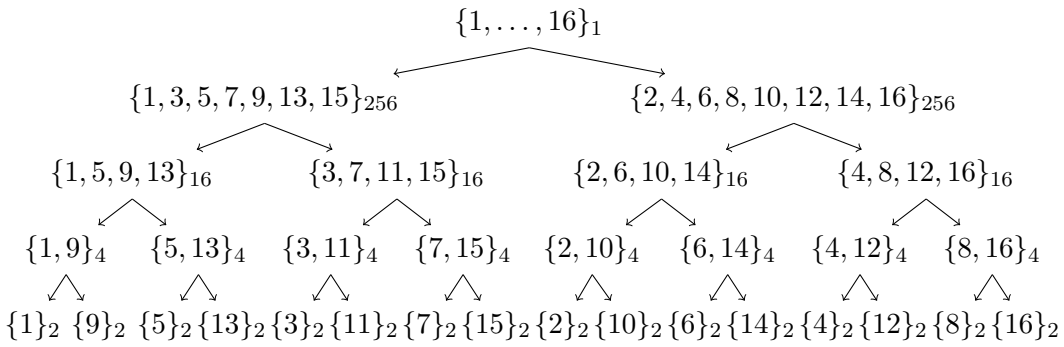

Figure 4: Dimension tree for $d = 16$ and pairwise clustering with subscripts k_t.

for which we eventually obtain $P_4 = \{D\}$. The final dimension tree is depicted in Figure 4. Note that the result would have been the same if we had used the ratios $k_t/(k_{t_1}k_{t_2}) = 1$ in each step.

Example 8 shows that both strategies did not lead to an appropriate dimension tree. One reason for this is that the optimal splitting of the directions $1, 2, 3, 4$ from $5, 6, 7, 8$ and $9, 10, 11, 12$ from $13, 14, 15, 16$ could not be detected by the combination of pairs of clusters. Therefore, a possible generalization is to allow the combination of a larger number of clusters in each agglomerative step.

Generalized Clustering

Let $P = \{t_1, \dots, t_r\}$ be again a partition of D with clusters $t_\nu \subset D$. For $p \geq 2$, the set

$$\mathcal{C}_p(P) := \{C \in \mathcal{P}(P) : 2 \leq \#C \leq p\}$$

contains the collection of possible combinations of at most p clusters from P. Among all $C \in \mathcal{C}_p(P)$, we can still try to minimize the associated rank k_s, $s := \bigcup_{t\in C} t$. Analogously, we could measure the ratio of the rank k_s corresponding to the new cluster $s := \bigcup_{t\in C}$ with respect to the given ranks k_t, $t \in C$. The following remark illustrates what we need to take into account when we want to compare ratios for different cardinalities of sets $C \in \mathcal{C}_p(P)$.

Remark 9. Let $t_1, t_2, u_1, u_2 \in P$ be pairwise disjoint clusters with $t := t_1 \cup t_2$, $u := u_1 \cup u_2$, and $s := t \cup u$. Assume that the associated ranks are given by $k_{t_1} = k_{t_2} = k_{u_1} = k_{u_2} = k_t = k_u = 2$ and $k_s = 4$. If we allow a combination of up to $p = 4$ clusters, we encounter that

$$\frac{k_s}{k_{t_1}k_{t_2}k_{u_1}k_{u_2}} = \frac{4}{2^4} = \frac{1}{4} < \frac{1}{2} = \frac{k_t}{k_{t_1}k_{t_2}}.$$

This means that the union of the clusters t and u would be rated better than the natural combination of t_1 and t_2. However, we find that $k_s/(k_t k_u) = 1$ which does not indicate any preference of joining t and u.

To overcome this problem, we suggest to use the following normalized ratio as a decision criterion.

Definition 10 (rank ratio)**.** Let P be a partition of D and let $C \subset P$. The ratio

$$\varrho(C) := \Big(k_s / \prod_{t \in C} k_t\Big)^{1/(\#C-1)}$$

is called the *rank ratio* with respect to C where $s := \bigcup_{t \in C} t$.

In Remark 9, we find that $\varrho(\{s\}) = (1/4)^{1/3} > 1/2 = \varrho(\{t\})$ such that the combination of t and u is not given any preference. We now come back to Example 8 and compare our agglomerative clustering strategy based on the absolute values of the ranks to the strategy using the rank ratios with $p = 4$.

Example 11. Assume first that in each agglomerative step the clusters contained in $C \in \mathcal{C}_p(P)$ are joined such the rank k_s, $s := \bigcup_{t \in C} t$, is minimal. In Example 8, the minimum k_s over all $C \in \mathcal{C}_4(P_0)$ is 4. Therefore, one cannot distinguish if it is better to join two arbitrary clusters from P_0 or, e.g., the clusters $\{1\}, \{2\}, \{3\}, \{4\}$. As a consequence, one may successively arrive at the partition P_1. Afterwards, the minimum k_s over all $C \in \mathcal{C}_4(P_1)$ is again 4 and one may end up at the partition P_2 as before. Once the partition P_2 is obtained, the rank 256 is unavoidable for some cluster in the final dimension tree.

Secondly, we use the rank ratio in order to decide which clusters we would like to join. The minimal ratio for all $C \in \mathcal{C}_4(P_0)$ is obtained by, e.g., $C = \{\{1\}, \{2\}, \{3\}, \{4\}\}$ for which $\varrho(C) = (4/2^4)^{1/3} = (1/4)^{1/3} < 1$. Note that for all $C \in \mathcal{C}_3(P_0)$ we have $\varrho(C) = 1$. In the next step, the minimal ratio is also obtained by $C = \{\{1,2,3,4\}, \{5\}, \{6\}, \{7\}\}$ since $k_s = 8$, $s := \bigcup_{t \in C} t$, such that $\varrho(C) = (8/(4 \cdot 2^3))^{1/3} = (1/4)^{1/3}$. Afterwards, the minimal ratio is obtained for $C = \{\{1, \ldots, 7\}, \{8\}\}$ (only) since $\varrho(C) = 4/(8 \cdot 2) = 1/4 < (1/4)^{1/3}$. Therefore, the splitting of the directions $1, \ldots, 8$ from $9, \ldots, 16$ is detected even in the worst case scenario. The maximal rank appearing at all clusters in the final dimension tree is 8.

The preceding example clearly indicates that it can be advantageous to use a relative cluster criterion like the rank ratio from Definition 10 instead of an absolute one. Let us now assume that for a given partition P of D an optimal set $C' \in \mathcal{C}_p(P)$ has been identified by either of the two strategies. We then need to update P by a new partition

$$P' := (P \setminus C') \cup \big\{\cup_{t \in C'} t\big\}. \tag{4}$$

The clusters $t \in C'$ still need to be organized as a binary tree. Due to Problem 6, we have to solve the following (local) minimization problem.

Problem 12. Let P be a partition of D and let $C \subset P$. For all $\tau \in \mathcal{P}(C)$ let $k_\tau := k_s$, $s := \bigcup_{t \in \tau} t$. Among all possible partition trees T_C find a minimizer of

$$\sum_{\substack{\tau \in T_C \setminus \mathcal{L}(T_C) \\ \mathrm{sons}(\tau) = \{\tau_1, \tau_2\}}} k_\tau k_{\tau_1} k_{\tau_2}.$$

As long as the cardinality of C' is small, one can try out all possibilities for $T_{C'}$ in order to solve Problem 12. Once an optimal tree $T_{C'}$ has been determined, we need to update the set of nodes $T \subset \mathcal{P}(D)$ from the previous step. The elements of $C' = \{t_1, \ldots, t_r\}$, $t_\nu \subset D$, are already contained in T such that we only need to add subsets $s \subset D$ to T for which there exists an inner node $\tau \in T_{C'} \setminus \mathcal{L}(T_{C'})$ with $s = \bigcup_{t \in \tau} t$. The update procedure is visualized for $C' = \{t_1, \ldots, t_4\}$ in Figure 5.

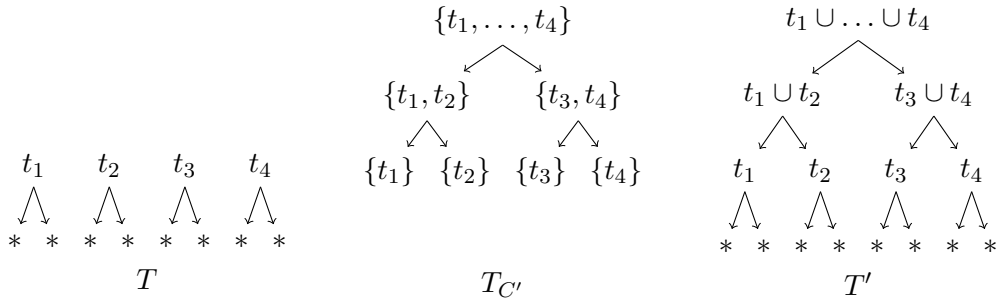

Figure 5: Agglomeration of nodes from T to T' by means of a local tree $T_{C'}$.

We have summarized our strategy in Algorithm 1.

Algorithm 1 $T = \text{BuildTree}(d, p)$

Define $P := T := \{\{1\}, \ldots, \{d\}\}$
while $\#P > 1$ **do**
  Choose $C' \in \mathcal{C}_p(P)$ with $C' := \operatorname{argmin}_{C \in \mathcal{C}_p(P)} \varrho(C)$
  Find an optimal partition tree $T_{C'}$ according to Problem 12
  Update $P := (P \setminus C') \cup \{\bigcup_{t \in C'} t\}$
  Update $T := T \cup \{\bigcup_{t \in \tau} t : \tau \in T_{C'} \setminus \mathcal{L}(T_{C'})\}$
end while
return T

4 Rank Estimation

In the previous section, we have assumed an exact knowledge of the ranks k_t for all $t \subset D$. If the ranks are not known a priori, one has to determine them numerically. As long as $\#\mathcal{I}$ is comparatively small, we can assume that a tensor $A \in \mathbb{R}^{\mathcal{I}}$ is available in its full representation such that we can perform standard rank revealing techniques for matrices applied to the matricizations $\mathcal{M}_t(A)$. For large $\#\mathcal{I}$, we have to base a rank estimate on a subset of tensor entries of A. This in turn means that in general — even with exact numerical computations — all rank estimation techniques for the matricizations of A need to remain heuristic.

For matrices $M \in \mathbb{R}^{I \times J}$, the *adaptive cross approximation* [2] is a well-known tech-

nique for the construction of low-rank approximations of the form

$$M \approx X^j := US^{-1}V^\top, \tag{5}$$

with matrices

$$U = M|_{I\times Q}, \quad S = M|_{P\times Q}, \quad V = M|_{J\times P},$$

and sets of pivots $P \subset I$, $Q \subset J$ with $\#P = \#Q = j$. The approximations X^j can be found in an incremental way by adding rank one updates to X^{j-1} which are constructed from the remainder $R^j := M - X^{j-1}$, cf. [2]. In each approximation step, the pivot sets P and Q are updated by a row and column index for which the modulus of the corresponding entry in R^j is as large as possible. The number of entries required from M then lies in $\mathcal{O}((\#I+\#J)k)$ where k corresponds to the rank of the final approximation X^k.

Let $t \subset D$, $s := D \setminus t$, and $M := \mathcal{M}_t(A) \in \mathbb{R}^{\mathcal{I}_t\times\mathcal{I}_s}$. As we have pointed out in [1], the cross approximation technique can be used to find low-rank approximations to M by two modifications:

1. The matrices U and V from (5) are never formed explicitly. The low-rank representation is used to compute entries of the remainder R^j directly.

2. Pivot elements are not sought in full rows or columns but only on so-called *fibers* $A_{(i_1,\dots,i_{\mu-1},\cdot,i_{\mu+1},\dots,i_d)} \in \mathbb{R}^{\mathcal{I}_\mu}$.

It can be shown that the construction of the pivot sets P and Q requires the evaluation of $\mathcal{O}(dnk^2)$ entries from A, $n := \max_{\mu\in D} n_\mu$, where the rank k of M is determined with respect to some (heuristic) target accuracy ε with $\|M - X^k\|_2 \lesssim \varepsilon\|M\|_2$.

An alternative approach is to first determine a random subset of row indices $I \subset \mathcal{I}_t$ and a random subset of column indices $J \subset \mathcal{I}_s$ such that $\#I$ and $\#J$ are small. We can then apply rank-revealing strategies to the submatrix $M := \mathcal{M}_t(A)|_{I\times J}$. Since we mostly do not know an appropriate size of M in advance, we suggest to adaptively enlarge the size of M by the following simple strategy.

For $\ell = 0, \dots, \ell_{\max}$, we construct a sequence of random index sets $I_\ell \subset \mathcal{I}_t$, $J_\ell \subset \mathcal{I}_s$ of size $s_\ell := \#I_\ell = \#J_\ell$ until the matrix $M_\ell := \mathcal{M}_t(A)|_{I_\ell\times J_\ell}$ has reached some level of redundancy that indicates the (heuristic) size of the rank of $\mathcal{M}_t(A)$. First, we start with small initial random pivot sets $I_0 \subset \mathcal{I}_t$, $J_0 \subset \mathcal{I}_s$, of size $s_0 := \#I_0 = \#J_0$ (say $s_0 = 10$). We can then estimate the rank k_0 of $M_0 := \mathcal{M}_t(A)|_{I_0\times J_0}$ by means of the standard adaptive cross approximation. If at step $\ell = 0, \dots, \ell_{\max}$ we have that $k_\ell < 2s_\ell$, we stop the iteration and return k_ℓ as a good rank estimate. Otherwise, we set $s_{\ell+1} := 3k_\ell$ and proceed to the next step. By means of this strategy we aim at approaching the true rank of $\mathcal{M}_t(A)$ from the bottom. The number of entries required from A then lies in $\mathcal{O}(sk)$ where s and k correspond to the size and the rank of the final submatrix M, respectively.

5 Complexity

In a short complexity analysis, we focus on a comparison of the computational cost for the proposed adaptive setup of a dimension tree with the cost for the final black box

approximation of a tensor by, e.g., [1]. In particular, we would like to compare the number of tensor evaluations arising from a combination of the two approaches.

In order to estimate the complexity of Algorithm 1, first note that each computation of $\varrho(C)$, $C \in \mathcal{C}_p(P)$, corresponds to the determination of a rank k_s, $s := \bigcup_{t \in C} t$, for which we assume a still abstract complexity of N_{rank} tensor evaluations. For the initial partition $P = \{\{1\}, \ldots, \{d\}\}$, we have

$$\#\mathcal{C}_p(P) = \binom{d}{2} + \ldots + \binom{d}{p} = \mathcal{O}(d^p).$$

The computation of $\varrho(C)$ for all $C \in \mathcal{C}_p(P)$ for the first time therefore requires $\mathcal{O}(d^p N_{\mathrm{rank}})$ tensor evaluations. After an agglomerative step, the partition P is updated to P' according to (4) such that the values $\varrho(C)$ need only be computed for all $C \in P'$ for which $s \in C$, $s := \bigcup_{t \in C'} t$. This requires at most $\mathcal{O}(d^{p-1} N_{\mathrm{rank}})$ tensor evaluations. As there are at most d agglomerative steps, we again arrive at a complexity of $\mathcal{O}(d^p N_{\mathrm{rank}})$.

The local optimization from Problem 12 does not require any tensor evaluations since all necessary ranks are already available. The combinatorial complexity is determined by the number of labeled full binary trees with p leaves which is given by $p! C_{p-1}$ where C_p denotes the p-th Catalan number. As long as $p \leq d/2$, we can hence estimate

$$p! C_{p-1} = \frac{p!(2p-2)!}{p!(p-1)!} = \prod_{j=0}^{p-2} (p+j) < d^{p-1}.$$

This means that in each agglomerative step the combinatorial complexity for the local minimization is also bounded by $\mathcal{O}(d^{p-1})$. This leads to an overall bound of $\mathcal{O}(d^p)$ which does not exceed the number of $\mathcal{O}(d^p N_{\mathrm{rank}})$ tensor evaluations.

Assume now that $N_{\mathrm{rank}} = \mathcal{O}(k^2)$ where k is the maximal rank of all matricizations considered in the agglomerative strategy. This bound can, e.g., be achieved by applying the rank estimation technique based on random submatrices as introduced in the last section. We then have to evaluate $\mathcal{O}(d^p k^2)$ tensor entries to construct a dimension tree by our adaptive strategy. This needs to be compared with the black box strategy from [1] which requires the evaluation of $\mathcal{O}(dk^3 + d^2 k^2 n)$ tensor entries. We can easily see that for $p = 2, 3$ both strategies may lead to comparable costs whereas for $p \geq 4$ the cost for the construction of the tree will dominate. This effect was also observed (though not rigorously analyzed) in the numerical examples presented in the next section.

6 Numerical Examples

In the numerical part, we are interested in the following questions:

1. How well can we recover a known tree structure by our clustering strategy?
2. How much do we have to invest for the estimation of the ranks k_t in order to reach a given target accuracy?

3. How large is the impact of the choice of the maximal cluster size p on the quality of the final tree?

Throughout this section, we restrict ourselves to tensors $A \in \mathbb{R}^{\mathcal{I}}$ with $\#\mathcal{I}_\mu := n$ for all $\mu = 1, \dots, d$. In order to compare the storage complexity of tensors $A \in \mathcal{H}(T_D, \boldsymbol{k})$ with respect to different trees T_D, we introduce the *effective rank* k_{eff} as the positive real solution of

$$(d-1)k_{\text{eff}}^3 + dnk_{\text{eff}} = N_{\text{storage}}(\mathcal{H}(T_D, \boldsymbol{k}))$$

where N_{storage} was defined in (2). A low value of k_{eff} then corresponds to a low storage complexity of a tensor with respect to T_D.

As discussed in Section 4, we heuristically estimate the rank of the matricization of a tensor by the rank of an appropriately chosen random submatrix for a prescribed accuracy ε_{est}. Once a tree T_D has been determined by our (deterministic) clustering strategy, we approximate a given tensor with respect to T_D by the black box strategy from [1] with a prescribed (heuristic) target accuracy $\varepsilon_{\text{final}}$. Since the determination of the ranks involves a random component, we perform 100 runs for each example and report the resulting average effective rank $k_{\text{eff}}^{\text{avg}}$ and the worst effective rank $k_{\text{eff}}^{\text{worst}}$ of the final approximation over all runs. Our first example is motivated by Example 8.

Example 13. Given $d \in \mathbb{N}$, let T_D be a balanced dimension tree as defined in Example 3(a). For $n = 2$, let $A \in \mathcal{H}(T_D, \boldsymbol{k}) \subset \mathbb{R}^{\mathcal{I}}$ be a random tensor with ranks

$$k_t := \begin{cases} 1, & t = D, \\ 2, & t \in \mathcal{L}(T_D), \\ 4, & \text{otherwise}, \end{cases}$$

such that the singular values of the matricizations $\mathcal{M}_t(A)$ at each node $t \in T_D$ decay approximately like 2^{-j}, $j = 1, \dots, k_t$. We now permute the indices of A such that the original structure of the tree T_D becomes invisible. We then apply our algorithm with a prescribed accuracy of $\varepsilon_{\text{est}} = \varepsilon_{\text{final}} = 10^{-4}$ in dimension $d = 8, 16, 32$ for different values of the maximal cluster size $p = 2, 3, 4$. For all runs, the accuracy of the final approximation was close to the presribed error tolerance.

The average and best effective rank for the adaptive strategy can be compared in Table 1 with the original effective rank $k_{\text{eff}}^{\text{orig}}$ which we had obtained taking into account the knowledge of the hidden tree structure. Moreover, we report the effective ranks $k_{\text{eff}}^{\text{bal}}$ and $k_{\text{eff}}^{\text{lin}}$ which we obtain for a fixed (unpermuted) balanced and linear dimension tree, respectively, without taking into account our adaptive strategy.

We can see that the ranks $k_{\text{eff}}^{\text{bal}}$ and $k_{\text{eff}}^{\text{lin}}$ rapidly increase with increasing d. For $d = 32$, we could not even meet the given target accuracy assuming a maximal number of 3000 pivot elements in (5). When applying our adaptive strategy, the original tensor structure tends to be the better resolved the more we increase the maximal cluster size p for all $d = 8, 16, 32$. However, with increasing d it gets more and more difficult to recover the original tree structure for all $p = 2, 3, 4$.

d	$k_{\text{eff}}^{\text{orig}}$	$k_{\text{eff}}^{\text{bal}}$	$k_{\text{eff}}^{\text{lin}}$	$p = 2$	$p = 3$	$p = 4$
8	3.0	4.8	4.5	4.3 / 4.3	3.1 / 3.2	3.0 / 3.0
16	3.2	19.5	19.2	8.8 / 13.0	4.1 / 4.9	4.0 / 4.0
32	3.3	> 200	> 200	26.5 / 35.1	13.7 / 34.0	5.2 / 8.3

Table 1: Random tensors with $n = 2$ with original effective ranks $k_{\text{eff}}^{\text{orig}}$. Ranks $k_{\text{eff}}^{\text{bal}}$ and $k_{\text{eff}}^{\text{lin}}$ for fixed balanced and linear dimension trees. Average and worst effective ranks $k_{\text{eff}}^{\text{avg}}$ / $k_{\text{eff}}^{\text{worst}}$ for different values of the maximal cluster size p.

This example indicates that an appropriate splitting of the directions $1, \dots, d$ may not be discovered on the level of pairwise combinations such that we need to apply our algorithm with $p > 2$. In the next example, we study the behaviour of our method with respect to different accuracies for the construction of the tree and for the final approximation.

Example 14. Given $d = 16$, let T_D be a balanced dimension tree as defined in Example 3(a). For $n = 20$, let $A \in \mathcal{H}(T_D, \boldsymbol{k}) \subset \mathbb{R}^{\mathcal{I}}$ be a random tensor with ranks $k_t = 20$ for all $t \in T_D \setminus D$ such that the singular values of the matricizations $\mathcal{M}_t(A)$ at each node $t \in T_D$ decay approximately like 2^{-j}, $j = 1, \dots, k_t$. Again, we permute the indices of A such that the original structure of the tree T_D becomes invisible. We now apply our algorithm with a maximal cluster size of $p = 2$ for different accuracies of $\varepsilon_{\text{est}}, \varepsilon_{\text{final}} \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$, $\varepsilon_{\text{final}} \leq \varepsilon_{\text{est}}$. For all runs, the accuracy of the final approximation was close to the presribed error tolerance.

The average and best effective rank for the adaptive strategy can be compared in Table 2 with the original effective rank $k_{\text{eff}}^{\text{orig}}$ which we had obtained taking into account the knowledge of the hidden tree structure. Moreover, we report the effective ranks $k_{\text{eff}}^{\text{bal}}$ and $k_{\text{eff}}^{\text{lin}}$ which we obtain for a fixed (unpermuted) balanced and linear dimension tree, respectively, without taking into account our adaptive strategy.

$\varepsilon_{\text{final}}$	$k_{\text{eff}}^{\text{orig}}$	$k_{\text{eff}}^{\text{bal}}$	$k_{\text{eff}}^{\text{lin}}$	ε_{est} = 1e-03	ε_{est} = 1e-04	ε_{est} = 1e-05	ε_{est} = 1e-06
1e-03	2.9	7.1	7.2	3.7 / 5.9	–	–	–
1e-04	5.4	33.3	35.2	10.7 / 21.9	5.6 / 6.9	–	–
1e-05	7.9	–	–	22.9 / 44.6	8.3 / 11.7	7.9 / 7.9	–
1e-06	10.7	–	–	42.5 / 68.8	11.8 / 21.6	10.7 / 10.7	10.7 / 10.8

Table 2: Random tensors in $d = 16$ with $n = 20$ and original effective ranks $k_{\text{eff}}^{\text{orig}}$. Ranks $k_{\text{eff}}^{\text{bal}}$ and $k_{\text{eff}}^{\text{lin}}$ for fixed balanced and linear dimension trees. Average and worst effective ranks $k_{\text{eff}}^{\text{avg}}$ / $k_{\text{eff}}^{\text{worst}}$ for different values for the accuracy ε_{est} controlling the generation of the tree and for the final approximation accuracy $\varepsilon_{\text{final}}$.

Given a fixed tree, the ranks $k_{\text{eff}}^{\text{bal}}$ and $k_{\text{eff}}^{\text{lin}}$ rapidly increase with the accuracy $\varepsilon_{\text{final}}$. For a too crude choice of the accuracy ε_{est}, the original tree structure cannot be resolved up to the final target accuracy $\varepsilon_{\text{final}}$. If however $\varepsilon_{\text{est}} \approx \varepsilon_{\text{final}}$, the original structure can be identified.

In a third example, we study the behaviour of our method for a tensor generated by a smooth function.

Example 15. Given $d = 16$, let $Q \in \mathbb{R}^{d\times d}$ and define

$$f(x) := \exp(-\tfrac{1}{2}x^\top Qx), \quad x \in \Omega := [-2, 2]^d.$$

In order to control the separability of the function f with respect to the variables $x_1, \dots, x_d$, we first introduce standard Givens rotations $G(i, j, \varphi) \in \mathbb{R}^{d\times d}$, $\varphi \in \mathbb{R}$, by

$$G(i,j,\varphi)_{k,\ell} := \begin{cases} \cos\varphi, & k = i,\ \ell = i, \ \text{ or } \ k = j,\ \ell = j,\\ \sin\varphi, & k = i,\ \ell = j,\\ -\sin\varphi, & k = j,\ \ell = i,\\ 1, & \ell = k \neq i, j,\\ 0, & \text{otherwise.} \end{cases}$$

Let now

$$Q := \left(\prod_{i=0}^{3}\prod_{j=1}^{3} G(4i+1, 4i+j+1, \varphi)\right)\prod_{j=1}^{3} G(1, 4j+1, \varphi).$$

Note that for $\varphi = 0$, we have $Q = \mathrm{Id}$ such that the function f is separable with respect to $x_1, \dots, x_d$. With increasing φ, we expect a stronger coupling between the directions linked through the Givens rotations.

For $n = 32$, let $A \in \mathbb{R}^{\mathcal{I}}$ be the discretization of f on an equidistant tensor grid on Ω of size n in each direction. As in the preceding examples, we permute the indices of A and apply our algorithm with maximal cluster sizes of $p = 2, 3, 4$ for different accuracies $\varepsilon_{\text{final}} = \varepsilon_{\text{est}}$. In Table 3 (top), we compare the results to the effective rank $k_{\text{eff}}^{\text{orig}}$ obtained by using a balanced binary tree for an unpermuted tensor with $\varphi = \pi/16$. We report the results of the same experiment with $\varphi = \pi/8$ in Table 3 (bottom).

As expected, a larger value of φ leads to a stronger coupling of f with respect to $x_1, \dots, x_d$ which is reflected by higher effective ranks in all examples. Moreover the original tensor structure tends to be better resolved if we increase the maximal cluster size up to $p = 4$. Note however that already for $p = 2$ moderate effective ranks could be obtained in all examples.

In the last example, we apply our algorithm to the discretization of a two-dimensional function using a tensorization approach.

Example 16. Let $\alpha \in \mathbb{R}$ and define a function $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ by

$$g(x, y) := (|y - \alpha x| + 1)^{-1}.$$

Given $d \in \mathbb{N}$, let $N := 2^d$ and define an equidistant grid on $[-1, 1)$ of mesh size $h := 2/N$ by

$$\xi_\ell = \ell h - 1, \qquad \ell = 0, \dots, N - 1.$$

$\varepsilon_{\text{final}}$	$k_{\text{eff}}^{\text{orig}}$	$k_{\text{eff}}^{\text{bal}}$	$k_{\text{eff}}^{\text{lin}}$	$p = 2$	$p = 3$	$p = 4$
1e-03	2.3	2.8	2.5	2.4 / 2.8	2.4 / 2.6	2.4 / 2.7
1e-04	3.9	7.5	6.5	5.1 / 6.6	4.7 / 6.7	4.7 / 6.5
1e-05	4.7	11.8	10.0	5.6 / 7.4	5.1 / 6.6	4.6 / 5.5
1e-06	6.9	22.3	17.7	8.7 / 12.2	7.3 / 10.3	6.8 / 9.4
1e-07	9.1	–	–	11.7 / 14.1	9.2 / 11.5	8.2 / 9.7
1e-08	11.0	–	–	15.3 / 24.8	13.5 / 17.2	10.8 / 13.7

$\varepsilon_{\text{final}}$	$k_{\text{eff}}^{\text{orig}}$	$k_{\text{eff}}^{\text{bal}}$	$k_{\text{eff}}^{\text{lin}}$	$p = 2$	$p = 3$	$p = 4$
1e-03	4.5	10.6	8.8	6.1 / 8.7	5.7 / 8.5	5.6 / 7.8
1e-04	7.8	25.4	19.7	10.0 / 14.1	8.6 / 13.8	7.8 / 11.7
1e-05	11.0	49.8	38.0	14.5 / 19.7	12.1 / 19.8	10.2 / 13.7
1e-06	14.7	71.4	46.7	20.0 / 30.7	16.7 / 26.5	13.8 / 19.5
1e-07	18.9	–	–	30.4 / 47.4	25.3 / 38.8	18.8 / 27.8
1e-08	22.9	–	–	35.7 / 48.4	28.6 / 42.0	21.5 / 32.3

Table 3: Function generated tensor from Example 15 for $\varphi = \pi/16$ (top) and $\varphi = \pi/8$ (bottom). Ranks $k_{\text{eff}}^{\text{bal}}$ and $k_{\text{eff}}^{\text{lin}}$ for fixed balanced and linear dimension trees. Average and worst effective ranks $k_{\text{eff}}^{\text{avg}}$ / $k_{\text{eff}}^{\text{worst}}$ for different values of the accuracy $\varepsilon_{\text{final}} = \varepsilon_{\text{est}}$ and of the maximal cluster size p.

The points ξ_ℓ may equally be indexed by a tuple $(i_1, \dots, i_d) \in \{0,1\}^d$, i.e.,

$$\xi_{(i_1,\dots,i_d)} := \xi_\ell, \qquad \ell = \sum_{\mu=1}^{d} i_\mu 2^{\mu-1}.$$

We now consider a tensor $A \in \mathbb{R}^{\mathcal{I}}$, $\mathcal{I} := \{0,1\}^{2d}$, defined by

$$A_{(i_1,\dots,i_{2d})} := g(\xi_{(i_1,\dots,i_d)}, \xi_{(i_{d+1},\dots,i_{2d})}), \qquad i_\mu \in \{0,1\},\ \mu = 1, \dots, 2d.$$

For $d = 8$, we can store all entries of A and perform standard singular value decompositions of the matricizations of A in order to determine their ranks up to an accuracy of $\varepsilon_{\text{est}} = \varepsilon_{\text{final}} = 10^{-8}$. We now perform our algorithm with $p = 2$ for different values of α and compare the resulting effective ranks to the approximation with respect to a fixed balanced and fixed linear tree in Table 4. A visualization of the corresponding trees produced by our algorithm is depicted in Figure 6.

For non-trivial values of α, the adaptive strategy allows for a much better resolution of the tensor structure than a fixed choice of the dimension tree. As Figure 6 illustrates, the tensor from the given example tends to be optimally represented by a linear structure. The corresponding (non-trivial) permutations of the directions $1, \dots, 2d$ could be identified by our algorithm. They indicate a typical pattern of the coupling of the components of the variables x and y in dependence on α.

α	$k_{\text{eff}}^{\text{bal}}$	$k_{\text{eff}}^{\text{lin}}$	$k_{\text{eff}}^{\text{adap}}$
0	2.5	2.4	2.4
1/4	11.6	9.8	3.8
1/2	16.7	14.9	3.8
3/4	20.4	19.0	4.7
1	23.6	22.6	3.9

Table 4: Tensorization of a two-dimensional function with 256×256 points. SVD-based rank estimation with accuracy $\varepsilon_{\text{final}} = 10^{-8}$. Ranks $k_{\text{eff}}^{\text{bal}}$ and $k_{\text{eff}}^{\text{lin}}$ for fixed balanced and linear dimension trees. Ranks $k_{\text{eff}}^{\text{adap}}$ with adaptively generated dimension trees for different values of α.

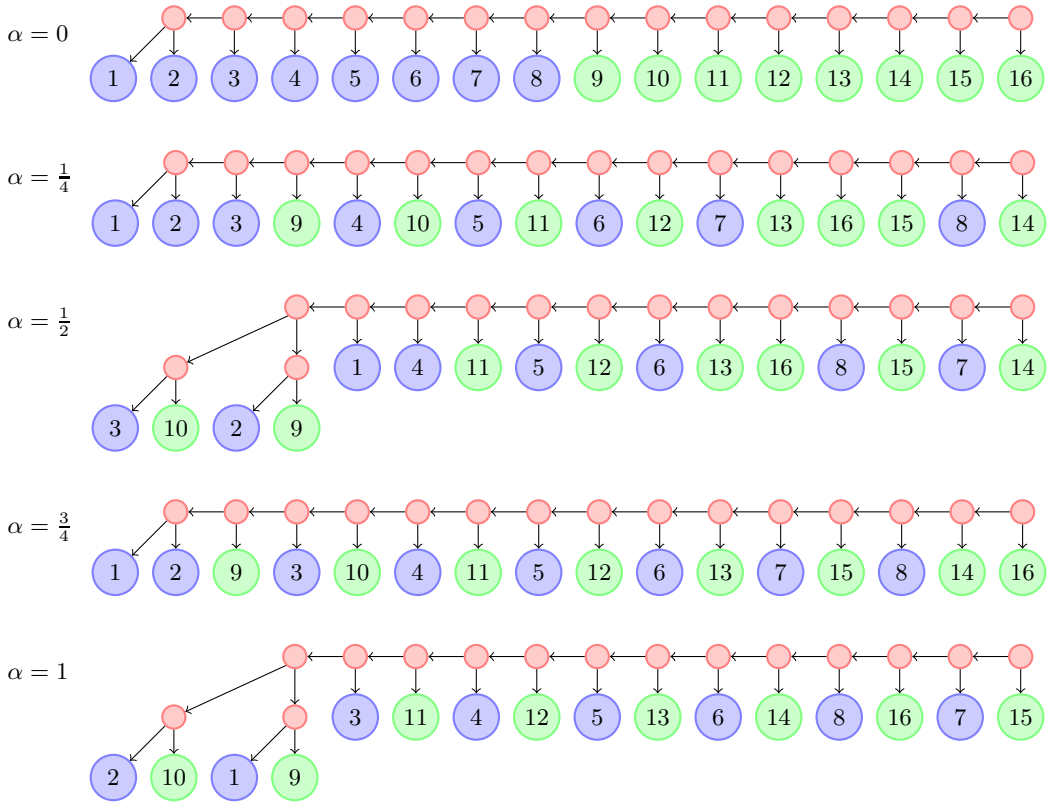

Figure 6: Tensorization of a two-dimensional function. Dimension trees for different values of α with components in x numbered from 1 to 8 (blue) and components in y numbered from 9 to 16 (green).

7 Conclusion

In this paper, we have introduced a strategy for the detection of an appriopriate dimension tree for tensors that can be represented in the hierarchical tensor format. Our

algorithm is based on an agglomeration of clusters in a bottom up way controlled by the relative reduction of the corresponding ranks in each step. By means of the numerical experiments, we verified that in many cases a reasonable tree structure could already be revealed by the combination of pairs of clusters. However, we also pointed out that one can find examples for which the combination of a higher number of clusters needs to be considered. On the one hand, this leads to higher computational costs for the construction of the tree in a setup step. On the other hand, one can benefit from the improved structure of the final representation in follow up computations.

References

[1] J. Ballani, L. Grasedyck, and M. Kluge. Black box approximation of tensors in hierarchical Tucker format. *Linear Algebra Appl.*, 438(2):639–657, 2013.

[2] M. Bebendorf. Approximation of boundary element matrices. *Numer. Math.*, 86(4):565–589, 2000.

[3] V. De Silva and L.-H. Lim. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM J. Matrix Anal. Appl.*, 30:1084–1127, 2008.

[4] L. Grasedyck. Hierarchical singular value decomposition of tensors. *SIAM J. Matrix Anal. Appl.*, 31:2029–2054, 2010.

[5] L. Grasedyck and W. Hackbusch. An introduction to hierarchical ($\mathcal{H}$-) rank and TT-rank of tensors with examples. *Comput. Methods Appl. Math.*, 11(3):291–304, 2011.

[6] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. Preprint, ANCHP, MATHICSE, EPF Lausanne, 2013.

[7] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus.* Springer, Berlin, 2012.

[8] W. Hackbusch and S. Kühn. A new scheme for the tensor representation. *J. Fourier Anal. Appl.*, 15(5):706–722, 2009.

[9] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[10] I. V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011.

[11] I. V. Oseledets and E. E. Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra Appl.*, 432:70–88, 2010.

[12] G. Vidal. Efficient classical simulation of slightly entangled quantum computations. *Phys. Rev. Lett.*, 91(14), 2003.