Received: 23 May 2019 | Revised: 10 January 2020 | Accepted: 17 January 2020

DOI: 10.1002/nla.2289

RESEARCH ARTICLE

WILEY

On algorithms for and computing with the tensor ring decomposition

Oscar Mickelin[1] | **Sertac Karaman**[2]

[1]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts

[2]Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts

Correspondence
Oscar Mickelin, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA.
Email: oscarmi@mit.edu

Funding information
Army Research Laboratory, DCIST program; Army Research Office, W911NF1510249; National Science Foundation, 1350685

Abstract

Tensor decompositions such as the canonical format and the tensor train format have been widely utilized to reduce storage costs and operational complexities for high-dimensional data, achieving linear scaling with the input dimension instead of exponential scaling. In this paper, we investigate even lower storage-cost representations in the tensor ring format, which is an extension of the tensor train format with variable end-ranks. Firstly, we introduce two algorithms for converting a tensor in full format to tensor ring format with low storage cost. Secondly, we detail a rounding operation for tensor rings and show how this requires new definitions of common linear algebra operations in the format to obtain storage-cost savings. Lastly, we introduce algorithms for transforming the graph structure of graph-based tensor formats, with orders of magnitude lower complexity than existing literature. The efficiency of all algorithms is demonstrated on a number of numerical examples, and in certain cases, we demonstrate significantly higher compression ratios when compared to previous approaches to using the tensor ring format.

KEYWORDS

graph-based tensor formats, tensor format conversions, tensor-ring format, tensors

1 | INTRODUCTION

Tensor decompositions were originally introduced in 1927[1] and have been widely used in several fields, ranging from scientific computing to data analysis, with an initial application in psychometrics.[2] With the advent of large-scale computing over the last decades, these decompositions have become even more relevant as they circumvent the "curse of dimensionality" by achieving operational complexities scaling linearly with the input dimension instead of exponentially. Recent applications include machine learning,[3,4] tensor completion,[5] and high-dimensional numerical analysis.[6–14] A large number of additional examples can be found in a recent review article[15] as well as a recent monograph.[16,17]

We consider the tensor ring format (or TR-format) of a tensor $T \in \mathbb{R}^{n_1 \times \ldots \times n_d}$, also known as the tensor chain format in earlier mathematics literature,[7] or matrix product states format with periodic boundaries in the physics literature[18–21] The format is given by tensors of the form

$$T(i_1, \ldots, i_d) = \text{Trace}\left(G_1(i_1) \cdot \ldots \cdot G_d(i_d)\right), \tag{1}$$

or in index-form

$$T(i_1, \dots, i_d) = \sum_{\alpha_0=1}^{r_0} \cdots \sum_{\alpha_{d-1}=1}^{r_{d-1}} G_1(\alpha_0, i_1, \alpha_1) \cdot \ldots \cdot G_d(\alpha_{d-1}, i_d, \alpha_0). \tag{2}$$

Here, the matrices $G_k(i_k)$ are of size $r_{k-1} \times r_k$ for each index i_k with $1 \leq i_k \leq n_k$. Each G_k can therefore be viewed as an order-three tensor in $\mathbb{R}^{r_{k-1} \times n_k \times r_k}$ and is called a core tensor of the representation in Equation (1). The vector $(r_0, r_1, \dots, r_d)$ with $r_d = r_0$ is called the TR-rank of the representation in Equation (1).

The TR-format can be seen as a natural extension of the successful tensor train format (TT-format)[6] where it is insisted that $r_0 = r_d = 1$, but the TR-format is known to have theoretical drawbacks in comparison. Generally, graph-based tensor formats with cycles in the associated graph are known to not be closed (in the Zariski topology),[22–24] which may lead to concerns about numerical stability issues in analogy to a classical setting.[25] Moreover, it was recently shown that for the TR-format, minimal TR-ranks for a given tensor need not be unique[24] (not even up to permutation of the indices $i_1, \dots, i_d$), leading to difficulties in their calculation. On the other hand, from a pragmatically practical viewpoint, the use of this format in numerical experiments has been seen to lead to lower ranks of the core tensors as compared to the TT-format,[4,26–29] resulting in higher compression ratios and lower storage costs required to represent a given tensor. The mathematics literature has therefore seen renewed interest in the TR-format in recent years, aiming to build on the success of the theory and applications of the TT-format while at the same time improving the compression ratios of the involved tensors even further.[4,26–31] Recent applications making use of the TR-format include compression of convolutional neural networks,[4] image and video compression,[28,29] tensor completion[32] as well as image and video reconstruction.[27] In this paper, we are interested in analysis and algorithms for computing with the TR-format that lead to higher compression ratios for representing tensors.

1.1 | Prior work

Previous work[28–30] has devised efficient singular value decomposition (SVD)- and alternating least-squares-based (ALS-based) approximation schemes to convert a tensor in full format to TR-format, has detailed how common linear algebra operations on the tensor level can be captured on the level of the TR-representation and has presented ways of converting a tensor into the TR-format from other common tensor formats such as the canonical or Tucker-format (e.g., the review article by Kolda and Bader[15] for an overview of these tensor formats).

For ALS-based algorithms, a recent result[33] characterizes the existence and nonexistence of spurious local minima in terms of the largest TR-rank. The geometric and algebraic properties of the TR-format have also been studied.[23,24] The closure of the set of tensors with bounded TR-rank has been related to geometric complexity theory.[23] The dimensions and generic ranks of the set of tensors with bounded ranks have been investigated by Ye and Lim,[24] together with additional properties about minimality of TR-ranks and intersections of sets of tensors with differently bounded ranks.

Handschuh[31] provides a method of transforming a given TT-representation into TR-format with $r_0 \neq 1$. Wang et al[27] consider completion of a tensor in the TR-format with missing entries, using iterations of ALS. Khoo and Ying[26] consider the problem of converting a tensor in full format into TR-format using only a limited sample of the tensor elements $T(i_1, \dots, i_d)$, which enables the algorithms to be applied to higher dimensional tensors. The nonsampled entries are recovered using an ALS-based iterative procedure.

1.2 | Remaining challenges

Compared to the analogous situation for the TT-format, a number of important questions remain a challenge:

- **Choice of TR-rank:** For the TT-format, it is known that each r_k is greater than or equal to the rank of the kth unfolding matrix of the tensor T (defined in Section 1.4 below) and equality is achieved when using the TT-SVD algorithm based on successive SVD-decompositions of the unfolding matrices.[6] In contrast, for the TR-format, there is not a single unique minimal rank of a given tensor.[24] The previously mentioned algorithms therefore require a manual choice of either r_0 (SVD-based algorithms) or the entire vector $(r_0, r_1, \dots, r_d)$ (ALS-based algorithms). How are these ranks to be chosen, and how does the end result depend on this choice?
- **On efficient rounding:** Arguably the most important algorithm for the TT-format is an efficient, SVD-based and noniterative rounding procedure to convert a TT-representation with suboptimal ranks into one with more beneficial ranks. Is there an efficient analogue also in the TR-format?

1.3 | Contributions

In order to answer these questions, we present the three main contributions in this paper.

1. **Importance of choice of TR-rank and heuristic algorithm:** We show that the compression ratio of the TR-format is highly dependent (in our examples, up to more than one order of magnitude) on the choice of the rank r_0 in Equation (1). In earlier work,[28–30] this choice was always kept fixed and not adapted to the underlying tensor, but finding the optimal r_0 is crucial for the efficiency of the TR-format. Using the notion of minimal TR-ranks defined below, we clarify why this is the case. By leveraging a certain invariance of the TR-format under cyclic shifts of the tensor dimensions $n_1, \dots, n_d$, we show how to improve compression ratios even further. We detail a heuristic algorithm to find a low-cost choice of r_0 and cyclic shift in the previous SVD-based algorithms.
2. **Rounding, operations, and compressed tensor format conversions:** We describe an efficient SVD-based rounding procedure to decrease the ranks of a given TR-representation. Surprisingly, we show that earlier definitions of common linear algebra operations in the literature need to be redefined in order for ranks to be reduced when combined with the rounding procedure. As an application, this enables us to specify conversions from the TT- and canonical formats into TR-format that are more efficient than previous results in the literature,[28,29] leading to orders of magnitude higher compression ratios. Moreover, the complexity of these conversions is asymptotically lower than previous ideas[31] by a factor of more than $\max_k n_k^3$.
3. **Extension to graph-based formats:** We extend these ideas to general graph-based tensor formats. As an application, we present an algorithm for selecting a low-cost graphical format with which to represent a given tensor. Operations on graph-based formats also enable the aforementioned conversions from the TT- and canonical formats into TR-format.

The remainder of the paper is organized as follows. Section 2 details a heuristic algorithm for choosing r_0 and cyclic shift to achieve low storage cost. Section 3 defines a rounding procedure for the TR-format and redefines some common linear algebra operations in order to make them amenable to storage-cost savings using the rounding procedure. Section 4 extends the procedures to general graph-based formats and contains algorithms for converting representations of tensors between different graph-based formats. Section 5 concludes with a number of performance comparisons of the presented algorithms with algorithms previously described in the literature. Implementations of all algorithms in this paper are publicly available online*.

1.4 | Notation

Algorithms will be presented in pseudocode using MATLAB commands and notation. We will mostly follow the notation of Hackbush.[34] The ith standard basis vector will be denoted by e_i. We will use the Frobenius tensor norm $\|\cdot\|_F$ given by $\|T\|_F^2 := \sum_{i_1,\dots,i_d} |T(i_1,\dots,i_d)|^2$. The kth unfolding matrix of a tensor $T \in \mathbb{R}^{n_1 \times \dots \times n_d}$ will be denoted by $T_{\langle k \rangle} \in \mathbb{R}^{\left(\prod_{j=1}^{k} n_j\right) \times \left(\prod_{j=k+1}^{d} n_j\right)}$, and can be computed by reshape $\left(T, \left[\prod_{j=1}^{k} n_j, \prod_{j=k+1}^{d} n_j\right]\right)$ in MATLAB. The δ-rank of a matrix A is $\operatorname{rank}_\delta(A) := \min_{B:\|A-B\|_F \le \delta} \operatorname{rank}(B)$ and can be computed by calculating the rank of the result of a δ-truncated SVD on A, $\operatorname{SVD}_\delta(A)$. The k-mode product of a tensor T with a matrix $A \in \mathbb{R}^{\ell \times n_k}$ is a tensor $T \times_k A$ in $\mathbb{R}^{n_1 \times \dots \times \ell \times \dots \times n_d}$ defined by $(T \times_k A)(i_1,\dots,i_{k-1},j,i_{k+1},\dots,i_d) := \sum_{i_k=1}^{n_k} T(i_1,\dots,i_d) A(j,i_k)$. The Hadamard (or elementwise) product of two tensors T_1, T_2 in $\mathbb{R}^{n_1 \times \dots \times n_d}$ is defined by $(T_1 \circ T_2)(i_1,\dots,i_d) := T_1(i_1,\dots,i_d) T_2(i_1,\dots,i_d)$.

We will denote the group of circular shifts on d variables by S_d^c, which has generator $\gamma = (1, d, d-1, \dots, 2)$ in cycle notation,[35 p. 30]. This notation means that $\gamma(1) = d, \gamma(d) = d-1, \dots, \gamma(2) = 1$, that is, γ corresponds to a circular shift to the left by one step. The inverse shift γ^{-1} therefore corresponds to a circular shift to the right by one step, and can be written in cycle notation as $\gamma^{-1} = (2, 3, \dots, d, 1)$.

For any $\tau \in S_d^c$ and tensor $T \in \mathbb{R}^{n_1 \times \dots \times n_d}$, we define the *$\tau$-permuted tensor* $T^\tau \in \mathbb{R}^{n_{\tau^{-1}(1)} \times \dots \times n_{\tau^{-1}(d)}}$ by $T^\tau(i_1,\dots,i_d) = T(i_{\tau(1)},\dots,i_{\tau(d)})$. Tensor products will be denoted by $\cdot \otimes \cdot$, and since we will exclusively deal with finite-dimensional vector spaces, we will explicitly work with coordinate representations and we identify $(v_1 \otimes v_2 \otimes \dots \otimes v_d)(i_1, i_2, \dots, i_d) := v_1(i_1) \cdot v_2(i_2) \cdot \dots \cdot v_d(i_d)$, for $v_k \in \mathbb{R}^{n_k}$. The Kronecker product of two matrices A, B will be denoted by $A \otimes_{\mathrm{K}} B$, to avoid confusion.

*https://github.com/oscarmickelin/tensor-ring-decomposition

For graph-based tensor formats, there is an elegant and coordinate-free notation[23,24] which avoids explicit reference to cores and indices. However, we will be concerned with practical algorithms, which therefore are required to work explicitly with the cores and indices which arise in the implementation data structures. Our notation is therefore chosen to reflect this. Let $\mathcal{G} = (V, E)$ be an undirected graph with the set of vertices $V = \{1, \dots, d\}$ and with the set of edges E. We will denote an edge between vertices labeled by k_1 and k_2 by (k_1, k_2). The set of edges emanating from the vertex k will be denoted by $e(k)$. For each $i \in e(k)$, we will specify a maximal edge rank $r_{ik} \in \mathbb{N}$. A tensor $T \in \mathbb{R}^{n_1 \times \dots \times n_d}$ will then be said to be in $\mathcal{G}$-format if there exist core tensors for each vertex, denoted by $G_k(i_k, \times_{i \in e(k)} \alpha_{ik})$ where $1 \leq \alpha_{ik} \leq r_{ik}$, such that $T(i_1, \dots, i_d)$ equals the contraction over all α_{ik} of $G_k(i_k, \times_{i \in e(k)} \alpha_{ik})$. We will then refer to the collection of r_{ik} for $1 \leq k \leq d$ and $i \in e(k)$ as the $\mathcal{G}$-ranks of T. See for example, the recent review article by Orús[20] for more information. In the case when $\mathcal{G}$ is a cycle of d vertices, the $\mathcal{G}$-format is precisely the TR-format.

2 | CONVERSION FROM FULL FORMAT TO TR-FORMAT

This section describes the conversion of a tensor in full format into TR-format. A negative result in Section 2.1 (Proposition 1) first describes the importance of the choice of r_0 and how the situation for the TR-format differs from the TT-format. Afterward in Section 2.2, we describe how considering cyclic shifts $\tau \in S_d^c$ gives an additional degree of freedom to be used in the conversion into TR-format. Both r_0 and the cyclic shift τ can be chosen via exhaustive search which gives a reduced storage-cost algorithm (Algorithm 2) in Section 2.3. Finally, in Section 2.4 we present an algorithm to produce a low-cost choice of r_0 and τ (Algorithm 3).

2.1 | Negative result

As presented by Zhao et al,[28,29] Algorithm 1 below can be used to compute an approximate TR-representation $\tilde{T}$ of a tensor T given in full format, given a desired accuracy ε. $\tilde{T}$ then satisfies $\|T - \tilde{T}\|_F \leq \varepsilon \|T\|_F$. Algorithm 1 computes the quantity $\delta := \frac{\varepsilon \|T\|_F}{\sqrt{d}}$ and requires a manual input of a divisor r_0 of $\text{rank}_\delta\left(T_{\langle 1 \rangle}\right)$. The choice of r_0 by Zhao et al[28,29] (and also for a related algorithm based on the skeleton/cross approximation[30]), is to minimize $|r_0 - \frac{\text{rank}_\delta(T_{\langle 1 \rangle})}{r_0}|$, but examples (see Section 5.1) show that this can lead to suboptimal compression ratios.

Algorithm 1. TR-SVD[28,29]

Input: d-tensor T in full format, accuracy ε, divisor r_0 of $\text{rank}_\delta\left(T_{\langle 1 \rangle}\right)$.
Output: Core tensors G_1, ..., G_d s.t. $\tilde{T}$ of form in Equation (1) has $\|T - \tilde{T}\|_F \leq \varepsilon \|T\|_F$.

1: Compute $\delta := \frac{\varepsilon \|T\|_F}{\sqrt{d}}$.
2: $C = \text{reshape}(T, [n_1, \frac{\text{numel}(T)}{n_1}])$ ▷ Initial step
3: $[U, \Sigma, V] = \text{SVD}_\delta(C)$
4: Put $r_1 := \text{rank}(\Sigma)$.
5: $G_1 = \text{permute}(\text{reshape}(U, [n_1, r_0, r_1]), [2, 1, 3])$
6: $C := \text{permute}(\text{reshape}(\Sigma V^T, [r_0, r_1, \prod_{j=2}^{d} n_j]), [2, 3, 1])$
7: Merge the last two indices by $C = \text{reshape}(C, [r_1, \prod_{j=2}^{d-1} n_j, n_d r_0])$.
8: **for** $k = 2 : d - 1$ **do** ▷ Main loop
9: $C = \text{reshape}(C, [r_{k-1} n_k, \frac{\text{numel}(C)}{(r_{k-1} n_k)}])$
10: $[U, S, V] = \text{SVD}_\delta(C)$
11: $r_k = \text{rank}(\Sigma)$
12: $G_k = \text{reshape}(U, [r_{k-1}, n_k, r_k])$
13: $C = \Sigma V^T$
14: **end for**
15: $G_d = \text{reshape}(C, [r_{d-1}, n_d, r_0])$ ▷ Final step

In Section 2.3, we will minimize storage costs of a TR-representation of a tensor by choosing r_0 appropriately. It is therefore of interest to compare the rank vectors r and r' resulting from the different choices r_0 and r'_0 in Algorithm 1. To this end, we will make use of the following concept. We will say that a vector $r := (r_0, \dots, r_{d-1}, r_0)$ is a minimal rank of T if (a) there exists a TR-representation of T with TR-ranks r, and (b) no other TR-rank r' of T satisfies $r' \leq r$ under the elementwise inequality in $\mathbb{R}^{d+1}$. The elementwise inequality is only a partial order on $\mathbb{R}^{d+1}$, so there can be multiple minimal ranks for a given tensor;[24] Proposition 1 below will show that this is in fact a common occurrence. For the TT-representation,[6] the ranks satisfy $r_k \geq \operatorname{rank}\left(T_{\langle k \rangle}\right)$ and an exact TT-decomposition using the TT-SVD algorithm results in ranks satisfying $r_k = \operatorname{rank}\left(T_{\langle k \rangle}\right)$. This is therefore the unique minimal rank with $r_0 = 1 = r_d$. For the TR-format, an analogous argument[29] shows that $r_0 r_k \geq \operatorname{rank}\left(T_{\langle k \rangle}\right)$, which will be used below.

Clearly, smaller rank vectors under the elementwise ordering result in lower storage cost. Rank vectors that are a priori known to be greater than others under the elementwise ordering can therefore be disregarded. Unfortunately, this situation does not occur when using Algorithm 1, as we show in Proposition 1.

Lemma 1. *Let r_0 divide $\operatorname{rank}\left(T_{\langle 1 \rangle}\right)$. If r denotes the rank obtained when running Algorithm 1 with precision $\varepsilon = 0$ and first rank r_0, then any minimal rank $r' \leq r$ has $r'_0 = r_0$ and $r'_1 = r_1$.*

Proof. The minimal rank obeys $r'_0 \leq r_0$ and $r'_1 \leq r_1$, by definition and $r'_0 r'_1 \geq \operatorname{rank}\left(T_{\langle 1 \rangle}\right) = r_0 r_1$, where equality holds by construction of the TR-SVD algorithm. The conclusion follows. ∎

Proposition 1. *Let r_0, r'_0 be two distinct divisors of $\operatorname{rank}\left(T_{\langle 1 \rangle}\right)$. If r and r' denote the ranks obtained when running Algorithm 1 with precision $\varepsilon = 0$ and first rank r_0 and r'_0, respectively, then there is no common minimal rank r_m satisfying $r_m \leq r$, and $r_m \leq r'$.*

Proof. This follows from Lemma 1 and the fact that $r_0 \neq r'_0$. ∎

Different choices of divisors r_0 and r'_0 are therefore incomparable a priori, so to find the rank which results in the lowest storage cost for a given accuracy, we need to compare the results of using all choices of divisor r_0 in Algorithm 1. This also implies that it is of interest to be able to convert a TR-representation with initial rank r_0 to one with initial rank $r'_0 \neq r_0$, and this is considered in Section 4.2 below.

2.2 | Cyclic shifts

We next consider cyclic shifts to reduce storage costs further. Let $\tau \in \mathrm{S}_d^c$ be a cyclic shift. If T is given in the form of Equation (1), then T^τ can be represented by

$$\begin{aligned} T^\tau(i_1, \dots, i_d) &= \sum_{\alpha_0=1}^{r_0} \cdots \sum_{\alpha_{d-1}=1}^{r_{d-1}} G_1(\alpha_0, i_{\tau(1)}, \alpha_1) \cdot \ldots \cdot G_d(\alpha_{d-1}, i_{\tau(d)}, \alpha_0) \\ &= \sum_{\alpha_0=1}^{r_0} \cdots \sum_{\alpha_{d-1}=1}^{r_{d-1}} G_{\tau^{-1}(1)}(\alpha_0, i_1, \alpha_1) \cdot \ldots \cdot G_{\tau^{-1}(d)}(\alpha_{d-1}, i_d, \alpha_0), \end{aligned} \tag{3}$$

using the fact that the trace of a product of matrices is unchanged under cyclic shifts of the matrices. T^τ is then of the format in Equation (1) with core tensors $G_{\tau^{-1}(1)}, \dots, G_{\tau^{-1}(d)}$. Conversely, we can fix a cyclic shift τ and compute a TR-representation of T^τ with cores $G_k(i_{\tau^{-1}(k)})$. A TR-representation of T is then given by the cores $G_{\tau(1)}(i_1), \dots, G_{\tau(d)}(i_d)$. We will see in the following that choosing the cyclic shift τ appropriately and computing a TR-representation in this way can result in far lower storage costs than without the use of the cyclic shift. The following is an illustrative example.

Example 1. Let $T \in \mathbb{R}^{n_1 \times \dots \times n_d}$ be a tensor given as a discretization of a function $f : \mathbb{R}^d \to \mathbb{R}$ that, in fact, does not depend on $x_2, \dots, x_{d-1}$. Consider a cyclic shift of the indices of T. Specifically, study $T^\tau \in \mathbb{R}^{n_d \times n_1 \times \dots \times n_{d-1}}$, for $\tau = (1, d, d-1, \dots, 2)^{-1}$. For $k \geq 2$, any unfolding matrix $T^\tau_{\langle k \rangle} \in \mathbb{R}^{(n_d n_1 \dots n_{k-1}) \times (n_k \dots n_{d-1})}$ has constant columns, so matrix rank 1. With $r_0 = 1$, Algorithm 1 results in $r_k = 1$ for $k \geq 2$ and consequently a low storage cost. Directly computing a TR-representation of T without making use of a cyclic shift will in general give ranks $r_k > 1$, incurring far higher storage cost. In other words, an appropriate choice of cyclic shift can result in great storage savings.

Note that cyclic shifts cannot be used in the same way when considering the TT-format. Storage costs of individual tensors can be reduced by fixing one given shift of the indices. However, operations in the TT-format such as addition can then only be performed on the two tensors if they use the same shifts. This makes this approach impractical for the TT-format. It should also be noted that this approach is not valid for general permutations, since for permutations τ that are not cyclic shifts, there is in general a k such that $\alpha_{\tau^{-1}(k)} \neq \alpha_{\tau^{-1}(k+1)}$.

2.3 | Algorithm with lower storage cost

The previous two sections provide an algorithm lowering the total storage cost, written out in Algorithm 2. It consists of an outer loop over each cyclic shift $\tau \in \mathrm{S}_d^c$ and an inner loop over each divisor r_0 of $\mathrm{rank}_\delta\left(T^\tau_{\langle 1\rangle}\right)$. For each choice of (τ, r_0), Algorithm 1 can be used to compute a tensor representation, and the representation with the lowest storage cost can be retained.

Algorithm 2. Reduced storage TR-SVD

Input: Full d-tensor T, accuracy ε.
Output: Core tensors G_1, ..., G_d s.t. $\tilde{T}$ of form in Equation (1) has $\|T - \tilde{T}\|_F \leq \varepsilon \|T\|_F$.

```
for k = 1 : d do
    Put τ_k = (1, d, d − 1, … , 2)^{k−1}
    for each r_0 divisor of rank_δ(T^{τ_k}_{⟨1⟩}) do
        Call Algorithm 1 on T^{τ_k} with first rank r_0
        Store T^{τ_k} if its TR-representation has the lowest storage cost so far
    end for
end for
```

This leads to a total complexity which is $C(d)$ times the cost of a single call to Algorithm 1, where $C(d) = \sum_{\tau_k \in \mathrm{S}_d^c}\left(\text{number of divisors of } \mathrm{rank}_\delta\left(T^{\tau_k}_{\langle 1\rangle}\right)\right) \sim \mathcal{O}\left(d \cdot \mathrm{div}(r)\right)$, if $\mathrm{rank}_\delta\left(T^{\tau_k}_{\langle 1\rangle}\right) \sim r$ for all k. Here, $\mathrm{div}(r)$ denotes the number of divisors of a positive integer r. The asymptotic complexity is then $\mathcal{O}(dr^{1/\mathrm{loglog}(r)})$ times that of a call to Algorithm 1. This procedure requires keeping the currently smallest (which can be significantly larger than the actually smallest) representation stored during the entire duration of the algorithm. It can, however, be run in parallel on up to d processors simultaneously.

2.4 | Heuristic reduced-cost algorithm

It would be desirable to determine a choice (τ, r_0) leading to low storage cost without exhausting all possibilities. In this section, we describe a heuristic approach to this choice with small extra computational cost compared to Algorithm 1.

2.4.1 | Heuristic choice of cyclic shift

Let $r_k(T)$ denote the TR-ranks of the tensor T as computed by Algorithm 1 for some choice of r_0 and ε that will be clear from the context. Note that the storage cost of a tensor in TR-format is given by the sum $\sum_k r_{k-1} r_k n_k$, so a low storage cost can be achieved by minimizing $\max_k r_k(T^\tau)$ over $\tau \in \mathrm{S}_d^c$. This would however require a sweep over all indices k for each choice of τ and therefore incur the same cost as computing the TR-SVD decompositions of all T^τ. However, the TT-ranks r_k as computed by the TT-SVD algorithm are typically nondecreasing up to an index k_*, after which they are nonincreasing. As a less-costly alternative to minimizing $\max_k r_k(T^\tau)$ over $\tau \in \mathrm{S}_d^c$, one can then instead minimize $r_2(T^\tau)$, in an attempt to minimize the slope of $r_k(T^\tau)$ as a function of k, and therefore minimize its maximum value. We therefore make the following definition.

Definition 1. The kth interaction matrix of a tensor $T \in \mathbb{R}^{n_1 \times \dots \times n_d}$ is denoted by $\mathcal{IM}_k(T) \in \mathbb{R}^{(n_k n_{k+1}) \times (n_{k+2} \dots n_d n_1 \dots n_{k-1})}$. It is given by $T^{\tau_k}_{\langle 2 \rangle}$, where $\tau_k = (1, d, d-1, \dots, 2)^{k-1}$. The k:th interaction rank of T is defined to be $ir_k(T) := \operatorname{rank}(\mathcal{IM}_k(T))$.

We will see in the examples below that choosing

$$k_* = \underset{k=1,\dots,d}{\operatorname{argmin}}\ ir_k(T). \tag{4}$$

and calling TR-SVD on $T^{\tau_{k_*}}$ results in low storage cost.

2.4.2 | Heuristic choice of divisor

After choosing k_*, we have the choice of choosing a divisor r_0 of $\operatorname{rank}_\delta\left(T^{\tau_{k_*}}_{\langle 1 \rangle}\right)$ when running TR-SVD above. As an alternative to looping over all possible divisors, we can choose the divisor to be close to the interaction ranks. In detail, we choose $r_0^* | \operatorname{rank}_\delta\left(T^{\tau_{k_*}}_{\langle 1 \rangle}\right)$ as

$$r_0^* = \underset{r_0 | \operatorname{rank}_\delta\left(T^{\tau_{k_*}}_{\langle 1 \rangle}\right)}{\operatorname{argmin}} \left| ir_{k_*-1}(T) - \frac{\operatorname{rank}_\delta T^{\tau_{k_*}}_{\langle 1 \rangle}}{r_0} \right| + \left| ir_{k_*}(T) - r_0 \right|. \tag{5}$$

The intuition behind this is that r_0 controls the strength of interaction between i_{k_*-1} and i_{k_*} in the TR-format, since larger values of r_0 accommodate a higher number of components of the corresponding core tensors G_{k_*-1} and G_{k_*}. Likewise, $\frac{\operatorname{rank}_\delta\left(T^{\tau_{k_*}}_{\langle 1 \rangle}\right)}{r_0}$ equals r_1 for the TR-representation of $T^{\tau_{k_*}}$. It therefore measures the strength of interaction between i_{k_*} and i_{k_*+1}.

Another measure of the strength of interaction between i_{k_*-1} and i_{k_*} in the full tensor T is $ir_{k_*}(T)$, since incoherence in these dimensions implies a large rank for the matrix $\mathcal{IM}_{k_*}(T)$, as in Example 1. Likewise, a strong interaction between i_{k_*} and i_{k_*+1} would result in a large rank of the matrix $\mathcal{IM}_{k_*-1}(T)$. We therefore choose r_0 to match these two measures of interaction as well as possible.

2.4.3 | Heuristic algorithm

We combine the steps in the previous two sections into an algorithm, which then consists of

- a preprocessing step wherein the interaction ranks $ir_k(T)$ are computed for each $k = 1, 2, \dots, d$. We choose k_* from Equation (4) and subsequently r_0^* from Equation (5). This results in a choice (τ_{k_*}, r_0^*).
- one call of Algorithm 1 on $T^{\tau_{k_*}}$, using the divisor r_0^* of $\operatorname{rank}_\delta\left(T^{\tau_{k_*}}_{\langle 1 \rangle}\right)$ producing cores $\tilde{G}_k(i_{\tau^{-1}_{k_*}(k)})$. A TR-representation of T is then given by the cores $\tilde{G}_{\tau_{k_*}(k)}(i_k)$.

We summarize this in Algorithm 3.

Algorithm 3. Heuristic TR-SVD

Input: Full d-tensor T, accuracy ε.
Output: Core tensors G_1, ..., G_d s.t. $\tilde{T}$ of form in Equation (1) has $\|T - \tilde{T}\|_F \leq \varepsilon \|T\|_F$.

1: Compute $\mathcal{IM}_k(T)$, $ir_k(T)$ for $k = 1, \dots, d$. ▷ Preprocessing step
2: Choose k_* from Equation (4) and subsequently r_0^* from Equation (5).
3: Run Algorithm 1 on $T^{\tau_{k_*}}$, using the divisor r_0^* of $\operatorname{rank}_\delta\left(T^{\tau_{k_*}}_{\langle 1 \rangle}\right)$ to obtain cores $\tilde{G}_k(i_{\tau^{-1}_{k_*}(k)})$.
4: Set $G_k(i_k) = \tilde{G}_{\tau_{k_*}(k)}(i_k)$.

Since $\text{rank}\left(T^{\tau_{k_*}}_{\langle 1\rangle}\right) \leq n_{k_*}$, the cost of the preprocessing step of Algorithm 3 is

$$\mathcal{O}\left((n_1n_2 + n_2n_3 + \ldots + n_dn_1)\prod_{i=1}^{d} n_i + \text{div}(n_{k_*})\right) \sim \mathcal{O}\left(dn^{d+2} + n^{\frac{1}{\log\log(n)}}\right), \tag{6}$$

if $n_i \sim n$ for all i. Comparisons of performance in terms of actual storage cost and total computation times are performed in Section 5.1.

3 | OPERATIONS

This section details some common operations performed on the TR-format. We firstly define an SVD-based rounding procedure for the TR-format (Algorithm 4 in Section 3.1), and then find it necessary to redefine the addition of two TR-representations in order to make full use of the rounding procedure (Theorem 2 and Proposition 2 in Section 3.2). We finally comment on the computation of the Frobenius norm of a tensor in Section 3.3. For the remainder of this section, we will let T be a tensor for which a TR-decomposition of the form in Equation (1) has been computed.

Algorithm 4. TR-rounding

Input: d-tensor T with cores $G_1, \ldots, G_d$, TR-ranks $r_0, \ldots, r_d$, accuracy ϵ.
Output: d-tensor $\tilde{T}$ with cores $\tilde{G}_k$, TR-ranks $\tilde{r}_k \leq r_k$ and $\|T - \tilde{T}\|_F \leq \epsilon\|T\|_F$.

Compute the core-level accuracy $\delta = \frac{\epsilon\|T\|_F}{\sqrt{dr_0}}$
for $k = 1 : d-1$ **do** ▷ Structured QR-sweep
  $[Q_k, R_k] = \text{QR}(\text{reshape}(G_k, [r_{k-1}n_k, r_k]))$
  $G_k = \text{reshape}(Q_k, [r_{k-1}, n_k, \text{numel}(Q_k)/(r_kn_k)])$
  $G_{k+1} = G_{k+1} \times_1 R_k$
end for
$[Q_d, R_d] = \text{QR}(\text{reshape}(G_d, [r_{d-1}n_d, r_d]))$ ▷ Cyclic step for reducing end-ranks
$[U_d, \Sigma_d, V_d] = \text{SVD}_\delta(R_d)$
$\tilde{r}_d = \text{rank}(\Sigma_d)$
if $\tilde{r}_d < r_d$ **then**
  $G_d = \text{reshape}(Q_dU_d\Sigma_d, [r_{d-1}, n_d, \tilde{r}_d])$
  $G_1 = G_1 \times_1 V_d^T$
end if
for $k = d : -1 : 2$ **do** ▷ SVD-sweep for reducing remaining ranks
  $[U_k, \Sigma_k, V_k] = \text{SVD}_\delta(\text{reshape}(G_k, [r_{k-1}, n_kr_k]))$
  $\tilde{r}_{k-1} = \text{rank}(\Sigma_k)$
  $\tilde{G}_k = \text{reshape}(V_k^T, [\tilde{r}_{k-1}, n_k, \tilde{r}_k])$
  $G_{k-1} = G_{k-1} \times_3 \text{reshape}(U_k\Sigma_k, [r_{k-1}, n_k\tilde{r}_{k-1}])^T$
end for
$\tilde{G}_1 = G_1$

3.1 | Rounding

Analogously to the case of the TT-format,[6] we can perform a rounding of a TR-representation with suboptimal ranks $r_0, r_1, \ldots, r_d$. The proposed procedure is, just like the TT-rounding procedure,[6] Algorithm 2, based on a structured QR-decomposition which enables lower costs of the SVD-computations in Algorithm 1, and results in a TR-representation with ranks $\tilde{r}_0, \tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_{d-1}, \tilde{r}_d = \tilde{r}_0$. Unlike for the TT-rounding procedure, invariance under cyclic shifts of the successive matrix factors in the QR-decompositions of the reshaped cores contribute to an even lower storage cost. Also, the

motivation for the TT-rounding procedure is that it carries out the same steps as the TT-SVD algorithm. This is not easily generalized to the TR-format—the reshaping of the low-rank decomposition of the first unfolding matrix $T_{\langle 1\rangle}$ in Algorithm 1 cannot be captured on the level of cores in a straightforward way. This necessitates a different approach, which is detailed in Algorithm 4. Unlike for TT-SVD, the TR-SVD algorithm has no guarantee of quasi-optimality since the TR-format is not closed. Instead, only a bound on the approximation error is guaranteed. It is therefore enough to produce a rounding procedure with just a bound on the approximation error, which then does not necessitate the procedure to perform the same steps as the TR-SVD algorithm.

We now prove the correctness of Algorithm 4.

Theorem 1. *Given a tensor T with cores G_k, Algorithm 4 returns cores $\tilde{G}_k$ with corresponding tensor $\tilde{T}$ satisfying $\|T-\tilde{T}\|_F \leq \varepsilon\|T\|_F$.*

Proof. The main insights in the proof are that (a) rounding of a TR-representation of T can be related to the rounding of a related TT-representation by reshaping T to have first component of size $r_0 n_1$; (b) introducing a cyclic step in Algorithm 4 enables a reduction of the end rank r_0 and this can be encoded by introducing a helper index i_{d+1} with special structure.

We will relate Algorithm 4 to an application of the TT-rounding procedure on an extended tensor T_e in $\mathbb{R}^{r_0 n_1 \times n_2 \times \ldots \times n_{d-1} \times n_d \times n_{d+1}}$, with $n_{d+1} = r_d$ and $r_{d+1} = 1$. The entries of T_e are defined by its TT-representation $T_e(\alpha_0 i_1, i_2, \ldots, i_d, i_{d+1}) = \sum_{\alpha_1,\ldots,\alpha_d} \prod_{k=1}^{d+1} G_k(\alpha_{k-1}, i_k, \alpha_k)$, where the final core is $G_{d+1}(i_{d+1}) = e_{i_{d+1}} \in \mathbb{R}^{r_d \times 1}$. Note here that the index α_0 is not summed over. Using the Kronecker delta defined as $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise, we have

$$\begin{aligned}\sum_{\alpha_0, i_{d+1}=1}^{r_d} T_e(\alpha_0 i_1, i_2, \ldots, i_d, i_{d+1})\delta_{\alpha_0, i_{d+1}} &= \sum_{\alpha_0,\alpha_1,\ldots,\alpha_d} \left(\prod_{k=1}^{d} G_k(\alpha_{k-1}, i_k, \alpha_k)\right) \cdot e_{\alpha_0} \\ &= \text{Trace}\left(\sum_{\alpha_1,\ldots,\alpha_{d-1}} \prod_{k=1}^{d} G_k(\alpha_{k-1}, i_k, \alpha_k)\right) = T(i_1, \ldots, i_d), \end{aligned} \tag{7}$$

which will be used below. Next, a TT-rounding,[6] Algorithm 2 of T_e with precision $\frac{\varepsilon\|T\|_F}{\sqrt{d r_0}}$ (instead of with $\sqrt{d-1}$, since T_e has $d+1$ dimensions) results in a tensor $\hat{T}_e$ with cores $\hat{G}_k$ satisfying

$$\|T_e - \hat{T}_e\|_F \leq \frac{\varepsilon\|T\|_F}{\sqrt{r_0}}. \tag{8}$$

By the choice of the final core G_{d+1}, Algorithm 4 now returns precisely the cores

$$\tilde{G}_k = \begin{cases} \hat{G}_k, & k \neq 1, \\ \text{reshape}(\hat{G}_1, [r_0, n_1, \tilde{r}_1]) \times_1 \hat{G}_{d+1}^T, & k = 1, \end{cases} \tag{9}$$

where the last equation ignores the singleton dimension $r_{d+1} = 1$ in $\hat{G}_{d+1}$. This can be verified by writing out the steps in Reference 6, Algorithm 2 and using the fact that $\text{reshape}(G_{d+1}, [r_d, n_{d+1}]) = I_{r_d}$, by construction. This means that

$$\begin{aligned}\sum_{\alpha_0, i_{d+1}=1}^{r_d} \hat{T}_e(\alpha_0 i_1, i_2, \ldots, i_d, i_{d+1})\delta_{\alpha_0, i_{d+1}} &= \sum_{\alpha_0,\alpha_1,\ldots,\alpha_d} \left(\prod_{k=1}^{d} \hat{G}_k(\alpha_{k-1}, i_k, \alpha_k)\right) \hat{G}_{d+1}(\alpha_d, \alpha_0, 1) \\ &= \text{Trace}\left(\sum_{\alpha_1,\ldots,\alpha_{d-1}} \prod_{k=1}^{d} \tilde{G}_k(\alpha_{k-1}, i_k, \alpha_k)\right) = \tilde{T}(i_1, \ldots, i_d), \end{aligned} \tag{10}$$

which we now use. Write $\Delta T := T - \tilde{T}$, and $\Delta T_e := T_e - \hat{T}_e$. Equations (7) and (10) now give

$$\|\Delta T\|_F^2 = \sum_{i_1,\dots,i_d} |\Delta T(i_1,\dots,i_d)|^2 = \sum_{i_1,\dots,i_d} \left| \sum_{\alpha_0, i_{d+1}=1}^{r_d} \Delta T_e(\alpha_0 i_1,\dots,i_{d+1})\delta_{\alpha_0,i_{d+1}} \right|^2$$
$$\leq r_0 \sum_{\alpha_0,i_1,\dots,i_d} |\Delta T_e(\alpha_0 i_1,\dots,\alpha_0)|^2 \leq r_0 \sum_{\alpha_0,i_1,\dots,i_{d+1}} |\Delta T_e(\alpha_0 i_1,\dots,i_{d+1})|^2 \leq \varepsilon^2 \|T\|_F^2, \tag{11}$$

where we used Equation (8) in the last step and Cauchy-Schwarz in the third step. ∎

Remark 1. The precision $\frac{\epsilon}{\sqrt{dr_0}}$ is usually overly conservative in practice, stemming from the fact that the second inequality in Equation (11) is generically far from being tight, since the left-hand side sums over a factor of r_0 fewer elements than the right-hand side.

The complexity of Algorithm 4 is $\mathcal{O}\left(dnr^3\right)$ plus the complexity of calculating the norm $\|T\|_F$. We will see below that it is possible to compute $\|T\|_F$ with complexity $\mathcal{O}\left(\min_k r_k dnr^3\right)$. This reduces to the cost of computing the Frobenius norm in the TT-format[6] when $\min_k r_k = 1$, that is, in particular when $r_0 = 1 = r_d$.

3.2 | Addition

One criterion for the rounding procedure in Algorithm 4 to be of practical use, is for it to reduce the ranks of the added tensor $T + T$ to those of T. For this to hold, we need to redefine the addition operation described in earlier literature, as in the following.

Theorem 2. *Let T' and T'' be tensors having TR-representations with cores G'_k, G''_k for $k = 1,\dots,d$. The tensor $T = T' + T''$ can be written in TR-format with cores G_k where*

$$G_k(i_k) = \begin{bmatrix} G'_k(i_k) & 0 \\ 0 & G''_k(i_k) \end{bmatrix}, \tag{12}$$

for $k = 2,\dots,d-1$. If $G'_1 \in \mathbb{R}^{r'_0 \times n_1 \times r'_1}$ and $G''_1 \in \mathbb{R}^{r''_0 \times n_1 \times r''_1}$ with $r'_0 \geq r''_0$

$$G_1(i_1) = \left[\begin{array}{c|c} & G''_1(i_1) \\ G'_1(i_1) & 0_{(r'_0 - r''_0)\times r''_1} \end{array}\right], \quad G_d(i_d) = \left[\begin{array}{c|c} G'_d(i_d) & \\ G''_d(i_d) & 0_{r''_{d-1}\times(r'_0 - r''_0)} \end{array}\right]. \tag{13}$$

If $r'_0 \leq r''_0$:

$$G_1(i_1) = \left[\begin{array}{c|c} G'_1(i_1) & \\ 0_{(r''_0 - r'_0)\times r'_1} & G''_1(i_1) \end{array}\right], \quad G_d(i_d) = \left[\begin{array}{c|c} G'_d(i_d) & \\ G''_d(i_d) & 0_{r''_{d-1}\times(r'_0 - r''_0)} \end{array}\right]. \tag{14}$$

Proof. We consider only the case $r'_0 \geq r''_0$; the remaining case is treated similarly. We have

$$\prod_{k=1}^{d} G_k(i_k) = \left[\begin{array}{c|c} \displaystyle\prod_{k=1}^{d-1} G'_k(i_k) & \prod_{k=1}^{d-1} G''_k(i_k) \\ & 0_{(r'_0 - r''_0)\times r''_{d-1}} \end{array}\right] \left[\begin{array}{c|c} G'_d(i_d) & \\ G''_d(i_d) & 0_{r''_{d-1}\times(r'_0 - r''_0)} \end{array}\right]. \tag{15}$$

If we write the left and right matrices in the product as $\begin{bmatrix} A & C \\ B & 0 \end{bmatrix}$, $\begin{bmatrix} A_1 & B_1 \\ C_1 & 0 \end{bmatrix}$, respectively, then it follows that

$$T(i_1,\dots,i_d) = \text{Trace}\left(\begin{bmatrix} A & C \\ B & 0 \end{bmatrix}\begin{bmatrix} A_1 & B_1 \\ C_1 & 0 \end{bmatrix}\right) = \text{Trace}\begin{bmatrix} AA_1 + CC_1 & AB_1 \\ BA_1 & BB_1 \end{bmatrix}$$
$$= \text{Trace}\begin{bmatrix} AA_1 & AB_1 \\ BA_1 & BB_1 \end{bmatrix} + \text{Trace}\,(CC_1) = T'(i_1,\dots,i_d) + T''(i_1,\dots,i_d). \tag{16}$$

∎

Remark 2. Another valid choice of added cores would be to stack the cores $G'_1(i_1)$, $G_1{}''(i_1)$ and the cores $G_d{}'(i_d)$, $G_d{}''(i_d)$ as

$$\left[\begin{array}{c|c} G'_1(i_1) & \\ \hline G''_1(i_1) & 0_{r''_0\times(r'_1-r''_1)} \end{array}\right], \quad \left[\begin{array}{c|c} & G''_d(i_d) \\ \hline G'_d(i_d) & 0_{(r'_{d-1}-r''_{d-1})\times r''_0} \end{array}\right], \tag{17}$$

respectively, which is proved by a similar calculation. By permutation invariance, any choice of adjacent indices where the cores of one index is stacked as in the left matrix in Equation (17) and the cores of the other as in the right matrix in Equation (17) then defines cores of an added tensor $T = T' + T''$. In the remainder, we fix the choice in Theorem 2 for the sake of definiteness.

Remark 3. Note that, unlike in the previous works,[22,29] we treat the end cores in a special fashion. This is required in order for the rounding procedure to reduce the size of the formally added tensor T in Theorem 2. In the previous works,[22,29] the formally added tensor T is chosen to have all cores defined by Equation (12), including $k = 1, d$. In view of Proposition 2 below, a call of TR-rounding on T would in this case have no effect at all.

Proposition 2. *Let T' and T'' be tensors with cores G'_k and G''_k, respectively, and define the tensor T by its cores $G_k(i_k) = \begin{bmatrix} G'_k(i_k) & 0 \\ 0 & G''_k(i_k) \end{bmatrix}$, for $k = 1, \dots, d$ (including $k = 1$ and $k = d$). If the TR-ranks of the representations of T' and T'' are minimal, then calling Algorithm 4 on T with accuracy $\varepsilon = 0$ returns cores with same size as the input cores G_k.*

Proof. The QR-sweep of Algorithm 4 first performs a QR-decomposition of the matrix

$$A = \begin{bmatrix} G'_1(1)^T & 0 & G'_1(2)^T & 0 & \dots & 0 \\ 0 & G''_1(1)^T & 0 & G''_1(2)^T & \dots & G''_1(n_1)^T \end{bmatrix}^T. \tag{18}$$

The matrices $A' = \begin{bmatrix} G'_1(1)^T & G'_1(2)^T & \dots & G'_1(n_1)^T \end{bmatrix}^T$, and $A'' = \begin{bmatrix} G''_1(1)^T & G''_1(2)^T & \dots & G''_1(n_1)^T \end{bmatrix}^T$ both have full column rank. To see this, assume that for example, A' could be factored as UV, for $U \in \mathbb{R}^{r_0 n_1 \times r}$, $V \in \mathbb{R}^{r \times r_1}$, and some $r < r_1$. Then reshape$(U, [r_0, n_1, r])$, $G'_2 \times_1 V, G'_3, \dots, G'_d$ would be cores of a representation of T' with TR-rank vector strictly smaller than the one of $G'_1, \dots, G'_d$, which contradicts the minimality assumption.

It then follows that also A has full column rank, so it has a unique reduced QR-factorization up to signs. When this is computed using the Gram-Schmidt procedure, it is clear that the decomposition $A = QR$ has the sparsity pattern

$$Q = \begin{bmatrix} Q'^T_1 & 0 & Q'^T_2 & \dots & Q'^T_{n_1} & 0 \\ 0 & Q''^T_1 & 0 & \dots & 0 & Q''^T_{n_1} \end{bmatrix}^T, \tag{19}$$

and $R = \begin{bmatrix} R' & 0 \\ 0 & R'' \end{bmatrix}$. In the QR-sweep, the 1-mode contraction of R with the next tensor in the sweep, that is, the matrix multiplication of R with the unfolded matrix

$$\begin{bmatrix} G'_2(1) & 0 & G'_2(2) & 0 & \dots & 0 \\ 0 & G''_2(1) & 0 & G''_2(2) & \dots & G''_2(n_2) \end{bmatrix}, \tag{20}$$

therefore preserves the sparsity pattern of Equation (18), which continues until the last core. By a similar argument, the SVD-sweep also respects the sparsity pattern of Equation (18). This means that all matrices for which we calculate the SVD-decomposition have full rank and no ranks are reduced in the SVD-sweep. The conclusion then follows. ∎

The size of the formally added tensor T in Proposition 2 is therefore not reduced by rounding, and storage costs in practical computations quickly explode. On the contrary, defining the addition operation as in Theorem 2 leads to sizeable storage-cost reductions.

As a check, it is easy to see that an application of the rounding procedure on the tensor $T + T$ gives a result with same ranks as T, assuming these ranks are minimal. To see this, the first step of the QR-sweep is to perform a reduced QR-decomposition of the matrix

$$\begin{bmatrix} G_1(1)^T & G_1(2)^T & \dots & G_1(n_1)^T \\ G_1(1)^T & G_1(2)^T & \dots & G_1(n_1)^T \end{bmatrix}^T = Q_1 \begin{bmatrix} R_1 R'_1 \end{bmatrix}, \tag{21}$$

where Q_1 has orthogonal columns and clearly $Q_1R_1 = Q_1R_1'$. Now, by minimality of the ranks of T, the matrix $\left[G_1(1)^T\ G_1(2)^T\ \dots\ G_1(n_1)^T\right]^T$ has full column rank, and therefore a QR-decomposition unique up to signs. Since $Q_1R_1 = Q_1R_1'$, and both Q_1R_1 and Q_1R_1' are QR-decompositions of this matrix, it follows that actually $R_1 = R_1'$. In the next step of the rounding procedure, the matrix $[R_1R_1]$ is multiplied with

$$\begin{bmatrix} G_2(1) & 0 & G_2(2) & 0 & \dots & 0 \\ 0 & G_2(1) & 0 & G_2(2) & \dots & G_2(n_2) \end{bmatrix}, \tag{22}$$

which gives $[R_1G_2(1), R_1G_2(1), \dots, R_1G_2(n_2), R_1G_2(n_2)]$. Iterating this procedure shows that each Q_k has dimensions $r_{k-1}n_k \times r_k$, for $k = 1, \dots d$. The matrix V_d is then of dimension $r_d \times r_d$ by minimality of the ranks of r_k, so no cyclic step is performed. By minimality, no ranks are reduced in the SVD-sweep, so the result has ranks $r_0, \dots, r_{d-1}, r_d$.

After a draft of this manuscript appeared, Batselier[36] made the observation that rounding in the TR-format after applying the Hadamard product of a tensor to itself might not always reduce the resulting ranks to the original ranks. This means that different variations of Algorithm 4 might need to be used for different applications. Devising and comparing these different variations remains an important open problem.

3.3 | Frobenius norm

In this section, we present an algorithm to compute the Frobenius norm of a tensor T, which improves on the cost of the algorithms previously recorded in the literature. In a previous approach,[29] it is suggested to form the Hadamard product $T' = T \circ T$ after which T' can be contracted with a tensor of all ones. This leads to a total complexity $\mathcal{O}\left(dnr^4 + dr^6\right)$. However, the explicit formation of the large tensor T' can be avoided, and the resulting Kronecker structure can be used together with permutation invariance to lower the total complexity to $\mathcal{O}\left(dnr^3 \min_k r_k\right)$.

To describe the algorithm, note that $\|T\|_F = \text{Trace}\,(X_1 \dots X_d)$, where $X_k = \sum_{i_k} G_k(i_k) \otimes_K G_k(i_k)$. When computing the products $X_{\leq k} := X_1 \dots X_k$, we can proceed from left to right and use the Kronecker structure to achieve cost $\mathcal{O}\left(knr_0r^3\right)$, so the total cost of computing $\|T\|_F = \text{Trace}\,(X_1 \dots X_d)$ is $\mathcal{O}\left(dnr_0r^3\right)$. The large matrix $G_k(i_k) \otimes_K G_k(i_k)$ never has to be explicitly formed during the calculations, except for the initial matrix $G_1(i_1) \otimes_K G_1(i_1)$. Note also that $\|T\|_F$ is invariant under cyclic shifts, so we can choose the shift minimizing r_0, giving total cost $\mathcal{O}\left(dnr^3 \min_k r_k\right)$. This is summarized in Algorithm 5.

Algorithm 5. Frobenius norm

Input: d-tensor T with cores $G_1, \dots, G_d$.
Output: $\|T\|_F$

1: Compute $k_* = \text{argmin}_k r_k$ and set $\tau_{k_*} = (1, d, d-1, \dots, 2)^{k_*-1}$, $G_k'(i_k) = G_{\tau_{k_*}^{-1}(k)}(i_{\tau_{k_*}^{-1}(k)})$.
2: $A := \sum_{i_1} G_1'(i_1) \otimes_K G_1'(i_1)$
3: **for** $k = 2 : d$ **do**
4: $\quad X_{\leq k}(i_k) := A\left(G_k'(i_k) \otimes_K G_k'(i_k)\right)$ using the Kronecker structure to reduce cost
5: $\quad A = \sum_{i_k} X_{\leq k}(i_k)$
6: **end for**
7: $\|T\|_F = \text{Trace}\,(A)$

4 | GRAPH-BASED FORMATS

As noted in the literature,[24] the representation ranks and therefore the storage cost of a tensor can depend strongly on the structure of the graph used for the tensor format. It is therefore desirable to generalize the algorithms from the previous sections to general graph formats. These questions are discussed below and lead to an algorithm to, given a tensor, select a low-cost graph representation, and to an alternative conversion from canonical format into TR-format, producing higher compression ratios.

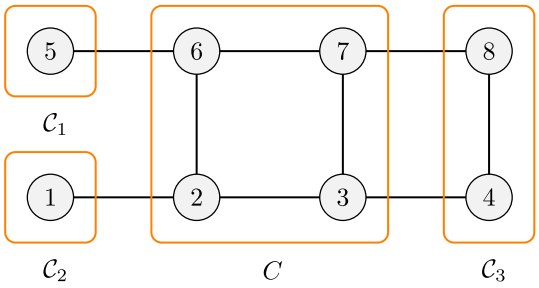

FIGURE 1 Illustration of the notation. This graph and cycle C have $i_C = \{i_2, i_3, i_6, i_7\}, i_{\mathcal{C}_1} = i_5, i_{\mathcal{C}_2} = i_1, i_{\mathcal{C}_3} = \{i_4, i_8\}$, $e_{C\mathcal{C}_1} = (5,6)$, $e_{C\mathcal{C}_2} = (1,2)$ and $e_{C\mathcal{C}_3} = \{(3,4),(7,8)\}$

4.1 | Rounding

For a general graph, we can perform a TR-rounding procedure on each cycle in the graph and alternate between different cycles to reduce overall ranks in a procedure we now describe. Let $\mathcal{G}$ be a graph containing a cycle C. The graph $\mathcal{G}\backslash C$ can be written in terms of its connected components $\mathcal{C}_k, k = 1, \dots, M$ as $\mathcal{G}\backslash C = \cup_{k=1}^{M} \mathcal{C}_k$. The cores of each connected component determine a tensor $T_{\mathcal{C}_k}(i_{\mathcal{C}_k}, \times_{(v,w)\in e_{C\mathcal{C}_k}} \alpha_{vw})$, where $e_{C\mathcal{C}_k}$ denotes the edges between C and $\mathcal{C}_K$ and $i_{\mathcal{C}_k}$ the indices of the vertices within $\mathcal{C}_k$. Likewise, we will use i_C to denote the indices of the vertices within C. See Figure 1 for an illustration.

The indices i_v of vertices v within $\mathcal{C}_k$ incident to a vertex w in C can be paired up with the edge (v,w), resulting in the core $T^e_{\mathcal{C}_k}(i^e_v) \in \mathbb{R}^{\times_{v\in\mathcal{C}_k}\left(\prod_{(v,w)\in e_{C\mathcal{C}_k}} n_v r_{vw}\right)}$, where for each $v \in \mathcal{C}_k$, we define $i^e_v = i_v$ for v not incident to C, and $i^e_v = i_v \prod_{(v,w)\in e_{C\mathcal{C}_k}} \alpha_{vw}$, written as one long index for all pairs (v,w) in $e_{C\mathcal{C}_k}$. Similarly, the indices in the cycle C can be paired up with the edges leading out of C to define a tensor

$$T^e_C(i^e_v) \in \mathbb{R}^{\times_{v\in C}\left(\prod_{k,(v,w)\in e_{C\mathcal{C}_k}} n_v r_{vw}\right)}, \tag{23}$$

where for all $v \in C$, we define $i^e_v = i_v$ for v not incident to any $\mathcal{C}_k$, and $i^e_v = i_v \prod_{k,(v,w)\in e_{C\mathcal{C}_k}} \alpha_{vw}$, written as one long index for (v,w) in $e_{C\mathcal{C}_k}$. The TR-rounding procedure can be applied to $T^e_C(i^e_v)$, giving a resulting tensor $\tilde{T}^e_C$, while leaving the remaining cores unaltered. The result can be reshaped back into the $\mathcal{G}$-format and we denote the result by $\tilde{T}$. The rounding error propagates to the whole tensor as in the following, which is a straightforward generalization of a result by Handschuh[31] section 6.

Theorem 3. $\|T - \tilde{T}\|_F \leq \|T^e_C - \tilde{T}^e_C\|_F \prod_{j=1}^{M} \|T^e_{\mathcal{C}_j}\|_F$.

Proof. The triangle inequality and the fact that the Frobenius norm is a crossnorm imply

$$\|T - \tilde{T}\|_F = \left\| \sum_{\substack{k,\alpha_{vw},\\ (v,w)\in e_{C\mathcal{C}_k}}} \left(T^e_C(i^e_C) - \tilde{T}^e_C(i^e_C)\right) \otimes \bigotimes_{j=1}^{M} T^e_{\mathcal{C}_j}(i^e_v) \right\|_F \leq \sum_{\substack{k,\alpha_{vw},\\ (v,w)\in e_{C\mathcal{C}_k}}} \|T^e_C(i^e_C) - \tilde{T}^e_C(i^e_C)\|_F \prod_{j=1}^{M} \|T^e_{\mathcal{C}_j}(i^e_v)\|_F. \tag{24}$$

Next, we use

$$\sum_{\beta_1,\dots,\beta_M} a(\beta_1,\dots,\beta_M) \prod_{j=1}^{M} b_j(\beta_j) \leq \left(\sum_{\beta_1,\dots,\beta_M} a(\beta_1,\dots,\beta_M)^2 \right)^{\frac{1}{2}} \prod_{j=1}^{M} \left(\sum_{\beta_j} b_j(\beta_j)^2 \right)^{\frac{1}{2}}, \tag{25}$$

which follows by induction on M from Cauchy-Schwarz, to conclude that

$$\|T - \tilde{T}\|_F^2 \leq \left(\sum_{\substack{k,\alpha_{vw},\\ (v,w)\in e_{C\mathcal{C}_k}}} \|T^e_C(i^e_C) - \tilde{T}^e_C(i^e_C)\|_F^2 \right) \left(\prod_{j=1}^{M} \sum_{\substack{\alpha_{vw},\\ (v,w)\in e_{C\mathcal{C}_j}}} \|T^e_{\mathcal{C}_j}(i^e_v)\|_F^2 \right) = \|T^e_C - \tilde{T}^e_C\|_F^2 \prod_{j=1}^{M} \|T^e_{\mathcal{C}_j}\|_F^2. \tag{26}$$

■

Because of the upper bound in Theorem 3, we can achieve a rounding error $\|T - \tilde{T}\|_F \leq \varepsilon \|T\|_F$ by performing the rounding of the cycle C with rounding error

$$\|T_C^e - \tilde{T}_C^e\|_F \leq \varepsilon \frac{\|T\|_F}{\prod_{j=1}^{M} \|T_{C_j}^e\|_F}. \tag{27}$$

Note that the second factor on the right-hand side is bounded from above by $\|T_C^e\|_F$, which can be seen by an application of Theorem 3 with $\tilde{T}_C^e = 0$.

We summarize the above considerations in Algorithm 6.

Algorithm 6. $\mathcal{G}$-truncation

Input: d-tensor T with cores $G_1, \dots, G_d$, $\mathcal{G}$-ranks r_{ik}, accuracy ε.
Output: d-tensor $\tilde{T}$ with cores $\tilde{G}_k$, $\mathcal{G}$-ranks $\tilde{r}_{ik} \leq r_{ik}$ and $\|T - \tilde{T}\|_F \leq \varepsilon \|T\|_F$.

1: Write $\mathcal{G}$ as a union of cycles and paths.
2: **for** each cycle or path C **do**
3: Call Algorithm 4 on T_C^e from Equation (23) with accuracy from Equation (27).
4: **end for**

4.2 | Transforming the graph structure

This section details an inexpensive way of transforming a tensor T given in a $\mathcal{G}$-graph-based format into a $\mathcal{G}'$-graph-based format, for any two graphs $\mathcal{G}$ and $\mathcal{G}'$.

The graph $\mathcal{G}$ can be transformed into the graph $\mathcal{G}'$ by successively inserting and deleting edges. We now describe how to capture these successive transformations on the level of the tensor T. For simplicity of the exposition, we describe the procedure of deleting an edge in TR-format (Algorithm 7) and inserting an edge into a TT-format (Algorithm 8).

Algorithm 7. Edge deletion

Input: TR-representation $G_1, \dots, G_d$ of d-tensor T, edge k to be deleted.
Output: TR-representation of T with $\tilde{r}_{k-1} = 1$.

1: Take $\tau_k = (1, d, d-1, \dots, 2)^{k-1}$
2: **for** $\ell = 1 : r_k$ **do**
3: define T_ℓ by $T_\ell(i_1, \dots, i_d) = \sum_{\substack{\alpha_0, \dots, \alpha_{d-1} \\ \alpha_d = \alpha_0}} \prod_{j=1}^{d} G_j(\alpha_{j-1}, i_j, \alpha_j) \delta_{\alpha_k, \ell}$
4: **end for**
5: $T = \sum_{\ell=1}^{r_k} T_\ell^{\tau_k}$
6: $T = T^{\tau_k^{-1}}$

Algorithm 8. Edge insertion

Input: TT-representation $G_1, \dots, G_d$ of d-tensor T, divisor $\tilde{r}_0$ of r_1.
Output: TR-representation $\tilde{G}_1, \dots, \tilde{G}_d$ of T with TR-ranks $\tilde{r}_k$.

1: Write $G_1(i_1) = \left[G_1^{(1)}(i_1) \cdots G_1^{(\tilde{r}_0)}(i_1) \right]$, and $G_2(i_2) = \left[G_2^{(1)}(i_2)^T \cdots G_2^{(\tilde{r}_0)}(i_2)^T \right]^T$, where each $G_1^{(j)}(i_1) \in \mathbb{R}^{1 \times \frac{r_1}{\tilde{r}_0}}$ and $G_2^{(j)}(i_2) \in \mathbb{R}^{\frac{r_1}{\tilde{r}_0} \times r_2}$
2: Form $\tilde{G}_1(i_1) = \left[G_1^{(1)}(i_1)^T \cdots G_1^{(\tilde{r}_0)}(i_1)^T \right]^T$ ▷ Stack first core
3: Form $\tilde{G}_2(i_2) = \left[G_2^{(1)}(i_2) \cdots G_2^{(\tilde{r}_0)}(i_2) \right]$ ▷ Stack second core
4: Form $\tilde{G}_k(i_k) = \mathrm{blockdiag}\,(G_k(i_k))$ for $k = 3, \dots, d$. ▷ Stack remaining cores

4.2.1 | Edge deletion

To describe the procedure for deleting an edge from a TR-representation, note that having $r_{k-1} = 1$ is equivalent to a TR-representation with the edge between vertices i_{k-1} and i_k deleted. By fixing all cores except for the kth one, the tensor can be written as a sum of r_{k-1} distinct tensors in TR-format, where each one has k:th core tensor $G_k(\ell, i_k, \cdot)$, for $1 \leq \ell \leq r_{k-1}$. We demonstrate in Theorem 4 that this addition can be performed on the level of the cores in such a way that the result has $k-1$th rank equal to 1 and the full procedure is written out in Algorithm 7.

Theorem 4. *If $\tilde{T}$ denotes the output of Algorithm 7 when called with input T, then $\tilde{T}$ has $k-1$:th TR-rank equal to 1 and $\tilde{T}(i_1, \dots, i_d) = T(i_1, \dots, i_d)$ for all $i_1, \dots, i_d$.*

Proof. With the notation from Algorithm 7, each $T_\ell^{\tau_k}$ can be represented with the cores $G_k(\ell, \cdot, \cdot)$, $G_{k+1}(\cdot, \cdot, \cdot), \dots, G_{k-1}(\cdot, \cdot, \ell)$ for fixed ℓ. Since the first core for each ℓ has size $1 \times n_k \times r_k$, it follows from Theorem 2 that $\sum_{\ell=1}^{r_k} T_\ell^{\tau_k}$ has first core of size $1 \times \cdot \times \cdot$, from which the first claim of the theorem follows. For the second, we have

$$\tilde{T}(i_1, \dots, i_d) = \sum_{\ell=1}^{r_k} T_\ell(i_1, \dots, i_d) = \sum_{\substack{\ell, \alpha_0, \dots, \alpha_d \\ \alpha_d = \alpha_0}} \delta_{\alpha_k, \ell} \prod_{j=1}^{d} G_j(\alpha_{j-1}, i_j, \alpha_j) = T(i_1, \dots, i_d). \tag{28}$$

■

Algorithm 7 itself has negligible cost. It can (and should) however be combined with a call to the TR-rounding procedure, resulting in a total complexity of $\mathcal{O}(dnr^3 r_{k-1}^3)$, since the cores of $\sum_{\ell=1}^{r_k} T_\ell^{\tau_k}$ have ranks not greater than rr_{k-1}. Note that $\tilde{r}_{k-1} = 1$ is unchanged by an application of TR-rounding.

4.2.2 | Edge insertion

Next, we detail the procedure of inserting an edge between indices i_1 and i_d in a TT-representation of T. This is equivalent to finding a TR-representation of T with $r_0 \neq 1$ and is achieved with negligible cost when performed as in Algorithm 8 below. The algorithm consists of a series of operations reshaping each core tensor G_k of T.

We now prove the correctness of Algorithm 8.

Theorem 5. *If $\tilde{T}$ denotes the output of Algorithm 8 when called with input T, then $\tilde{T}$ satisfies $\tilde{T}(i_1, \dots, i_d) = T(i_1, \dots, i_d)$ for all $i_1, \dots, i_d$.*

Proof. Write $G_1(i_1) = \begin{bmatrix} G_1^{(1)}(i_1) & \dots & G_1^{(\tilde{r}_0)}(i_1) \end{bmatrix}$, and $G_2(i_2) = \begin{bmatrix} G_2^{(1)}(i_2)^T & \dots & G_2^{(\tilde{r}_0)}(i_2)^T \end{bmatrix}^T$, where each $G_1^{(j)}(i_1) \in \mathbb{R}^{1 \times \frac{r_1}{\tilde{r}_0}}$ and $G_2^{(j)}(i_2) \in \mathbb{R}^{\frac{r_1}{\tilde{r}_0} \times r_2}$. Algorithm 8 returns $\tilde{G}_1(i_1) = \begin{bmatrix} G_1^{(1)}(i_1)^T & \dots & G_1^{(\tilde{r}_0)}(i_1)^T \end{bmatrix}^T$, $\tilde{G}_2(i_2) = \begin{bmatrix} G_2^{(1)}(i_2) & \dots & G_2^{(\tilde{r}_0)}(i_2) \end{bmatrix}$ and $\tilde{G}_k(i_k) = \text{blockdiag}\,(G_k(i_k))$ for $k = 3, \dots, d$. Writing $H = \prod_{k=3}^{d} G_k(i_k)$, this gives

$$\begin{aligned} \tilde{T}(i_1, \dots, i_d) &= \text{Trace} \begin{bmatrix} G_1^{(1)}(i_1) G_2^{(1)}(i_2) & \dots & G_1^{(1)}(i_1) G_2^{(\tilde{r}_0)}(i_2) \\ \vdots & \ddots & \vdots \\ G_1^{(\tilde{r}_0)}(i_1) G_2^{(1)}(i_2) & \dots & G_1^{(\tilde{r}_0)}(i_1) G_2^{(\tilde{r}_0)}(i_2) \end{bmatrix} \text{blockdiag}\,(H) \\ &= \sum_{\alpha_1=1}^{\tilde{r}_0} \text{Trace}\left(G_1^{(\alpha_1)}(i_1) G_2^{(\alpha_1)}(i_2) H \right) = \text{Trace}\,(G_1(i_k) G_2(i_k) H) = T(i_1, \dots, i_d). \end{aligned} \tag{29}$$

■

Just as for Algorithm 7, Algorithm 8 should be combined with an application of TR-rounding, resulting in a total cost $\mathcal{O}\left(dnr^3 \tilde{r}_0^3\right)$, since the TR-ranks of $\tilde{T}$ are no greater than $r\tilde{r}_0$.

Remark 4. When combined with the TR-rounding procedure, Algorithm 7 and Algorithm 8 have cost $\mathcal{O}\left(dnr^3 \tilde{r}_0^3\right)$ and $\mathcal{O}\left(dnr^3 r_{k-1}^3\right)$. The question of deleting and inserting edges was considered also in,31 producing algorithms with complexity $\mathcal{O}\left(dn^4 r^6 + n^6 r^6\right)$ and $\mathcal{O}\left(dn^4 r^3 \tilde{r}_0^3\right)$, respectively. Our results therefore have complexity lower by a factor greater than $\max_k n_k^3$.

Remark 5. The rounding procedure applied to the result of Algorithm 8 can be costly, since the ranks of the resulting tensor T can be large if r_0 is large. To mitigate this, let $\gamma = (1, d, d-1, \dots, 2)$. Note that T^γ is in the form $T^\gamma = \sum_{\alpha=1}^{\tilde{r}_0} T'_{\alpha_0}$, where T'_{α_0} has first core $G_2^{(\alpha_0)}(i_2)$, middle cores $G_k(i_k)$ for $k = 3, \dots, d$ and last core $G_1^{(\alpha_0)}(i_1)$. These can be added successively, with a rounding procedure with accuracy $\frac{\epsilon}{r_0}$ applied after each formal addition. The ranks are then reduced at every stage, leading to lower complexity. Similarly, for Algorithm 7, the rounding procedure can be performed after each addition.

Remark 6. Algorithm 8 can also be used to transform a TR-representation with ranks r_k into a TR-representation with ranks $r_0 \cdot \rho_1, \frac{r_1}{\rho_1}, r_2 \cdot \rho_1, \dots, r_d \cdot \rho_1$, where ρ_1 is any divisor of r_1.

The framework introduced in Algorithms 7 and 8 can be applied to transform a general graph $\mathcal{G}$ into another graph $\mathcal{G}'$ by successively inserting and deleting edges. Theorem 6 can be used to reduce storage sizes of the core tensors involved in the calculations.

4.3 | Applications of edge insertion and deletion

We now present two applications of Algorithms 7 and 8.

4.3.1 | Greedy algorithm for selecting graph structure

Consider a tensor T for which we would like to choose a graph $\mathcal{G}$ with which to represent T. We introduce a cost function $f_T(\mathcal{G})$ measuring the cost associated to representing T with the graph $\mathcal{G}$ (storage cost, maximum/average rank etc.) and would like to minimize $f_T(\mathcal{G})$. Clearly, an exhaustive search over all possible graph structures is computationally intractable because of their number. We therefore limit the scope to searching over graphs $\mathcal{G}$ with a certain structure and let the set of permissible graphs be denoted by $\mathbb{G}$. We consider the problem of finding $\mathcal{G}^* = \operatorname{argmin}_{\mathcal{G}\in\mathbb{G}} f_T(\mathcal{G})$. Algorithm 9 details the simplest attempt at finding a low-cost graph, although we do not claim any convergence to the minimum value. Two examples attaining low cost are shown in Section 5.4.

Algorithm 9. Greedy algorithm for selecting graph structure

Input: d-tensor T, set of permissible graphs $\mathbb{G}$.
Output: Graph $\mathcal{G} \in \mathbb{G}$ and T in $\mathcal{G}$-graph-based format.

```
Take an initial G ∈ 𝔾.
Compute a representation of T in G-format using Algorithm 1 and inserting or deleting edges using Algorithms 7 and 8.
for e ∈ {1, ..., d}^2 do
    G' = G ∪ e
    if G' ∈ 𝔾 then
        Compute representation of T in G'-format using Algorithm 8
        if f_T(G') < f_T(G) then
            G = G'
        end if
    end if
end for
```

4.3.2 | Converting canonical format into TR-format

We also detail a conversion of a tensor in canonical format to TR-format with increased compression ratio. This can be seen as a special case of the previous section when the initial graph is the chain on d elements and $\mathbb{G}$ is precisely the cycle on d elements in the same order as the chain. Given a tensor T in rank-r canonical format $T = \sum_{i=1}^{r} \bigotimes_{j=1}^{d} v_i^{(j)}$, a TT-representation of T is defined[6] by

$$G_1(i_1) = \left[v_1^{(1)}(i_1), \dots, v_r^{(1)}(i_1)\right], \quad G_d(i_d) = \left[v_1^{(d)}(i_d), \dots, v_r^{(d)}(i_d)\right]^T,$$
$$G_k(i_k) = \operatorname{diag}\left(v_1^{(k)}(i_k), \dots, v_r^{(k)}(i_k)\right), \quad k = 2, \dots, d-1. \tag{30}$$

In the TR-format, we now have an additional degree of freedom to choose the starting rank r_0 as any divisor of r, and therefore apply Algorithm 8 with this choice. The optimal choice of r_0 can be chosen in a loop where the representation with currently smallest storage cost is retained. A numerical example is shown in Section 5.3.

Remark 7. Unlike in previous works,[22,29] the end cores G_1 and G_d are treated differently from the other cores. This is done in order to ensure that the rounding procedure is able to reduce the TR-ranks of the tensor (cf, Remark 3).

5 | COMPUTATIONAL EXPERIMENTS

This section is devoted to numerical studies of the algorithms presented in Sections 2–4. All computations were carried out on a MacBook Pro with a 3.1 GHz Intel Core i5 processor and 16 GB of memory.

5.1 | Converting from full format to TR-format

We convert a tensor given in full format into a TR-representation and compute its storage cost. We compare the TT-representation with $r_0 = 1$, Algorithm 1 using a balanced representation[29,30] with $r_0 = \operatorname{argmin}|r_0 - \frac{\operatorname{rank}_\delta(T_{\langle 1\rangle})}{r_0}|$, and Algorithm 2, to Algorithm 3. We do not compare to other algorithms for TR-decompositions, since these have already been compared to the TR-SVD algorithm in the literature.[29] The tensors were taken to be discretizations of functions $f(x_1, \dots, x_d)$ on a grid in $\mathbb{R}^d$ with n discretization points in each dimension.

We first demonstrate our approach on a set of generic examples. It is easy to see that tensors with entries sampled from an absolutely continuous distribution have unfolded matrices of full rank, with probability 1. These therefore lead to decompositions with high rank. We will instead study a different set of randomly generated, generic examples. We choose $f(x)$ to be a multivariate polynomial with randomly generated exponents. Specifically, we generate exponents α_{ij} uniformly in the set $\{1, \dots, m_{\text{deg}}\}$, and compress the discretization of the five-dimensional function

$$\sum_{j=1}^{m_{\text{term}}} x_1^{\alpha_{1j}} x_2^{\alpha_{2j}} \cdot \ldots \cdot x_5^{\alpha_{5j}}, \tag{31}$$

into the TR-format. The parameters were set to $d = 5$, $n = 20$, $\varepsilon = 10^{-12}$ and the grid to be $[0, 1]^d$. The result is shown in Table 1 as a function of m_{deg} and m_{term}. As an average over 100 samples of α_{ij}, we see that both Algorithms 2 and 3 perform better than the TT-format, whereas Algorithm 1 with the balanced representation $r_0 = \operatorname{argmin}|r_0 - \frac{\operatorname{rank}_\delta(T_{\langle 1\rangle})}{r_0}|$ performs significantly worse. This shows that taking r_0 and the choice of cyclic shift into account can have considerable effect on the compression ratio.

In order to distinguish the contributions of the shift and the choice of r_0, we also make the following comparisons. We first determine the compression ratios when using no shift of the variables. We choose the corresponding r_0 by exhaustive search, the choice of r_0 in the heuristic Algorithm 3 and the balanced choice $r_0 = \operatorname{argmin}|r_0 - \frac{\operatorname{rank}_\delta(T_{\langle 1\rangle})}{r_0}|$. Next, we use the shift determined by Algorithm 3 and find the corresponding r_0 by exhaustive search and the balanced choice of $r_0 = \operatorname{argmin}|r_0 - \frac{\operatorname{rank}_\delta(T_{\langle 1\rangle})}{r_0}|$. The result is shown in Table 2. Comparing to Table 1, we see that both the choice of cyclic shift and r_0 contribute to higher compression ratios for both Algorithms 2 and 3.

Lastly, we study a few examples in greater detail, shown in Table 3. Unless otherwise noted, the parameters were set to $d = 5$, $n = 20$, $\varepsilon = 10^{-12}$ and the grid to be $[0, 1]^d$. The balanced choice of r_0 is ambiguous in that both r_0 and $\frac{\operatorname{rank}_\delta(T_{\langle 1\rangle})}{r_0}$ minimize the expression and we report the choice with the highest resulting storage cost.

TABLE 1 Storage cost and runtime in tensor ring format as fraction of storage cost and runtime in tensor train format, for tensors in Equation (31). Runtimes were computed averaged over 100 random samples of α_{ij}

	Algorithm 1									
	Storage quotient					**Runtime quotient**				
					m_{term}					
m_{deg}	**2**	**6**	**10**	**14**	**18**	**2**	**6**	**10**	**14**	**18**
2	1	1	1	1	1	0.9887	0.9923	0.9974	0.9947	0.9936
4	1	1.4661	1.6734	1.7357	1.6381	0.9950	1.0063	1.0079	1.0063	1.0067
6	1	1.6218	1.6359	1.8878	1.9491	0.9924	1.0096	1.0127	1.0335	1.0440
8	1	2.0743	1.7643	1.7724	2.0492	0.9940	1.0136	1.0083	1.0159	1.0590
10	1	1.9221	1.9383	2.6276	3.1629	0.9954	1.0088	1.0155	1.0837	1.1574
	Algorithm 2									
	Storage quotient					**Runtime quotient**				
					m_{term}					
m_{deg}	**2**	**6**	**10**	**14**	**18**	**2**	**6**	**10**	**14**	**18**
2	0.9093	0.9218	0.9361	0.9689	0.9783	12.0628	14.4779	14.6611	14.6384	14.6350
4	0.9129	0.8745	0.8845	0.9114	0.9032	13.6462	16.3108	18.2493	18.9628	19.0863
6	0.9489	0.8838	0.8953	0.9131	0.9074	13.9525	16.8535	17.7344	20.8090	22.0073
8	0.9437	0.9143	0.9006	0.9080	0.9260	13.9874	17.5574	19.0868	19.4535	19.8855
10	0.9550	0.9287	0.9147	0.9178	0.9366	14.2610	17.4405	19.6042	20.6393	21.2999
	Algorithm 3									
	Storage quotient					**Runtime quotient**				
					m_{term}					
m_{deg}	**2**	**6**	**10**	**14**	**18**	**2**	**6**	**10**	**14**	**18**
2	0.9412	0.9682	0.9469	0.9736	0.9783	5.5980	5.5146	5.5340	5.5154	5.5294
4	0.9785	0.9539	0.9293	0.9500	0.9454	5.5566	5.2291	5.1481	5.0998	5.0395
6	0.9827	0.9360	0.9496	0.9513	0.9470	5.5174	5.0911	4.8143	4.6878	4.6005
8	0.9812	0.9718	0.9453	0.9571	0.9764	5.5246	4.9621	4.6655	4.4752	4.3403
10	0.9962	0.9866	0.9605	0.9588	0.9845	5.4981	4.9188	4.5702	4.2490	4.1036

The first function in Table 3 has a coupling of the x_1 and x_d variables. Because of Example 1, one could therefore expect a correctly chosen cyclic shift of the variables to lead to significant storage savings, which is indeed the case. The second example has couplings between both x_1, x_d, and x_1, x_2. One might then expect the balanced choice of r_0 in Algorithm 1 to lead to good compression ratios. It is therefore somewhat surprising that this is not the case, and that Algorithm 2 gives an order of magnitude larger compression ratio.

The results suggest again that finding the optimal choice of r_0 and cyclic shift τ are crucial for storage savings when using the TR-format. Finally, we study if multiple cyclic shifts can attain the optimal compression ratios. For each choice of cyclic shift $(1, d, d-1, \dots, 2)^k$, for $k = 0, \dots, d-1$, we choose the corresponding r_0 both by exhaustive search and the balanced choice of $r_0 = \text{argmin}|r_0 - \frac{\text{rank}_\delta(T_{\langle 1 \rangle})}{r_0}|$. The results are shown in Table 4, and indicate that several shifts can lead to the highest compression ratio.

We finally include an example of compression of a real-world dataset. The Coil-100 dataset[38] consists of photographs of 100 objects at 70 different viewing angles. We subsample each of these images to an $n \times n$ image, for varying n. This results in a tensor in $\mathbb{R}^{100\times70\times3\times n\times n}$, which we reshape into a tensor in $\mathbb{R}^{10\times10\times10\times7\times3\times n\times n}$. The storage and run-times resulting from the compression algorithms are shown in Table 5. Note that the heuristic Algorithm 3 results in a near-optimal storage cost with very low run-time.

TABLE 2 Storage cost and runtime in tensor ring format as fraction of storage cost and runtime in tensor train format, for tensors in Equation (31). Runtimes were computed averaged over 100 random samples of α_{ij}

	No cyclic shift														
	Balanced choice					**Exhaustive search**					**Choice in Algorithm 3**				
	m_{term}														
m_{deg}	**2**	**6**	**10**	**14**	**18**	**2**	**6**	**10**	**14**	**18**	**2**	**6**	**10**	**14**	**18**
2	1	1	1	1	1	0.9458	0.9448	0.9569	0.9786	0.9867	0.9512	0.9782	0.9569	0.9836	0.9883
4	1	1.4661	1.6734	1.7357	1.6381	0.9654	0.9350	0.9303	0.9438	0.9437	0.9885	0.9639	0.9393	0.9600	0.9554
6	1	1.6218	1.6359	1.8878	1.9491	0.9752	0.9410	0.9517	0.9544	0.9480	0.9927	0.9460	0.9596	0.9613	0.9570
8	1	2.0743	1.7643	1.7724	2.0492	0.9775	0.9569	0.9510	0.9494	0.9692	0.9912	0.9818	0.9553	0.9671	0.9864
10	1	1.9221	1.9383	2.6276	3.1629	0.9812	0.9630	0.9590	0.9629	0.9747	1.0063	0.9966	0.9705	0.9688	0.9945
	Cyclic shift as in Algorithm 2														
	Balanced choice					**Exhaustive search**					**Choice in Algorithm 3**				
	m_{term}														
m_{deg}	2	6	10	14	18	2	6	10	14	18	2	6	10	14	18
2	0.9412	0.9682	0.9469	0.9736	0.9783	0.9299	0.9360	0.9419	0.9703	0.9783	0.9412	0.9682	0.9469	0.9736	0.9783
4	0.9785	1.3497	1.7010	1.7979	1.6809	0.9579	0.9330	0.9138	0.9393	0.9387	0.9785	0.9539	0.9293	0.9500	0.9454
6	0.9827	1.6332	1.6542	1.7661	1.8563	0.9714	0.9247	0.9385	0.9446	0.9355	0.9827	0.9360	0.9496	0.9513	0.9470
8	0.9812	2.0878	1.7566	1.8493	1.9250	0.9663	0.9494	0.9361	0.9440	0.9653	0.9812	0.9718	0.9453	0.9571	0.9764
10	0.9962	1.7905	2.0889	2.4774	3.0893	0.9812	0.9643	0.9453	0.9495	0.9734	0.9962	0.9866	0.9605	0.9588	0.9845

TABLE 3 Storage cost and runtime in tensor ring format as fraction of storage cost and runtime in tensor train format, for tensors in Section 5.1. Runtimes were computed averaged over 100 function calls. Algorithm 1 is the version of Algorithm 1 in which r_0 is computed by $r_0 = \text{argmin}|r_0 - \frac{\text{rank}_\delta(T_{(1)})}{r_0}|$

	Storage quotient			Runtime quotient		
$f(x)$	Algorithm 1*	Algorithm 2	Algorithm 3	Algorithm 1*	Algorithm 2	Algorithm 3
$\exp\left(\cos(x_1x_d + \sum_{k=2}^{d-1} x_k)\right)$	0.518	0.070	0.070	1.694	19.168	2.431
$\exp\left(\cos(x_1x_d + x_1x_2 + \sum_{k=3}^{d-1} x_k)\right)$	1.796	0.298	0.298	1.171	25.218	2.629
Park function 1[37]	0.941	0.217	0.217	0.661	15.158	1.563
$(d = 4, \text{grid } [10^{-10}, 1]^d)$						
$\left(1 + \sum_{k=1}^{d} x_k^2\right)^{-\frac{1}{2}}$	2.776	1	1	1.153	24.663	3.407
$\exp\left(\sum_{k=1}^{3} x_kx_{k+1}x_{k+2} + x_4x_5x_1\right)$	1.2771	0.7674	1	1.1257	21.5445	3.1206

TABLE 4 Storage cost in tensor ring format as fraction of storage cost in tensor train format and as function of the cyclic shift $(1, d, d-1, \ldots, 2)^k$. r_0 is chosen by exhaustive search and the balanced choice of r_0 in Algorithm 1

	Exhaustive search					Balanced choice				
k	0	1	2	3	4	0	1	2	3	4
$\exp\left(\sum_{k=1}^{3} x_kx_{k+1}x_{k+2} + x_4x_5x_1\right)$	1	1	**0.7674**	**0.7674**	1	1.2771	1.8851	1.7819	1.3999	1.8851
$\exp\left(\cos(x_1x_d + x_1x_2 + \sum_{k=3}^{d-1} x_k)\right)$	1	**0.2982**	**0.2982**	**0.2982**	**0.2982**	1.796	0.7115	**0.2982**	1.0406	0.5555
Park function 1[37]	**0.2169**	**0.2169**	**0.2169**	**0.2169**	—	0.941	0.8434	0.9478	0.9442	—

TABLE 5 Storage cost and runtime in tensor ring format as fraction of storage cost and runtime in tensor train format, for the Coil-100 dataset. Algorithm 1* is the version of Algorithm 1 in which r_0 is computed by $r_0 = \text{argmin}|r_0 - \frac{\text{rank}_\delta(T_{(1)})}{r_0}|$

	Storage quotient			Runtime quotient		
n	Algorithm 1*	Algorithm 2	Algorithm 3	Algorithm 1*	Algorithm 2	Algorithm 3
24	1.617	0.750	0.783	1.743	30.62	0.943
32	2.030	0.650	0.655	1.322	22.35	0.732
40	2.384	0.540	0.540	0.759	15.16	0.352

5.2 | Implicit PDE solvers

This section considers simple implicit finite difference solvers of linear singular value decompositions (PDEs) and compares the storage cost and runtime of solvers in the TR- and TT-format, respectively. We consider the wave equation with zero Dirichlet boundary conditions

$$\begin{cases} \frac{\partial^2 u}{\partial t^2}(x,t) = \Delta u(x,t), \quad x \in [0,1]^d, t \in [0,T] \\ u(x,0) = u_0(x), \\ u(x,t) = 0 \ \text{ on the boundary of } \ [0,1]^d. \end{cases} \tag{32}$$

For the simple solver, we firstly represent a first-order finite difference approximation of the Laplacian in TR- and TT-format. We consider a u_0 given in canonical format with $n = 2^5$ discretization points in each dimension and convert it into TT- and into the optimal TR-format in Section 4.3.2, using Remark 5. Afterward, implicit Euler is used as time-marching, and we explicitly compute the inverse $\left(I - (\Delta t)^2\Delta\right)^{-1}$ to be applied at each time step in the TT-format, using the procedure of Oseledets and Dolgov.[39] After each iteration, we perform a rounding using either TT-rounding, or TR-rounding. We choose $u_0(x)$ to have significantly higher compression ratio in the TR-format than the TT-format and

TABLE 6 Storage cost in tensor ring format as fraction of storage cost in tensor train format after 500 time steps, for tensors in Section 5.2

	$u_{0,1}(x)$		$u_{0,2}(x)$	
d	**Storage quotient**	**Runtime quotient**	**Storage quotient**	**Runtime quotient**
5	0.0499	1.2195	0.5322	1.3599
10	0.0225	0.5021	0.4840	1.0507
20	0.0170	0.3632	0.4778	0.9796
30	0.0295	0.9246	0.5212	1.0240

investigate whether or not the TR-rounding procedure manages to keep the compression ratio high, even after a large number of iterations. We show the results for the two initial functions

$$u_{0,1}(x) = \sum_{k=1}^{20} \sin(kx_1)\sin(kx_d), \quad u_{0,2}(x) = x_1 x_d + \sum_{k=2}^{d-1} x_k, \tag{33}$$

in Table 6. The parameters used were $\varepsilon = 10^{-12}$, $\Delta t = 10^{-3}$, and $n = 2^5$ discretization points in each dimension. The results show that the TR-format maintains lower storage cost than the TT-format after many iterations, possibly at the expense of longer runtime.

5.3 | Converting canonical format to TR-format

We consider the discretization of a function given in canonical format on a grid in $[0,1]^d$ with $n = 2^5$ discretization points in each dimension. Converting the resulting canonical decomposition into the TR-format in Equation (30) and using Remark 5, we find the optimal storage cost among all permutations and choices of r_0 and compare the costs to the TT-format obtained by $r_0 = 1$. The precision of the rounding was set to $\varepsilon = 10^{-12}$. We consider the four examples

$$T_1 = x_1 x_d + \sum_{k=2}^{d-1} x_k, \quad T_2 = \sum_{k=1}^{\frac{d}{2}} x_k x_{d-k+1}, \quad d \text{ even}, \tag{34}$$

$$T_3 = \sum_{k=1}^{\frac{d}{2}-1} x_k x_{k+1} x_{d-k} x_{d+1-k}, \quad d \text{ even}, \quad T_4 = \sum_{j=1}^{10} x_1^{\alpha_{1j}} x_2^{\alpha_{2j}} \cdot \ldots \cdot x_d^{\alpha_{dj}}, \tag{35}$$

where the α_{ij} are positive integers generated uniformly from the set $\{1, \ldots, 20\}$, and averaged over 100 samples. Since the variables x_1, x_d in T_1 are coupled, one can expect high compression ratios after choosing an appropriate cyclic shift, but it is less obvious if this is also true for the tensors T_2, T_3 and T_4. Figure 2 shows the results, with good storage savings for a range of dimensions and functions.

5.4 | Selecting graph structure

We apply Algorithm 9 to discretizations of the two functions

$$f(x) = \sum_{k=1}^{20} \sin(kx_1)\sin(kx_{\lfloor \frac{d}{2} \rfloor}), \quad g(x) = \sum_{k=1}^{\frac{d-1}{2}} x_{2k-1} x_{2k+1}, \quad d \text{ odd}, \tag{36}$$

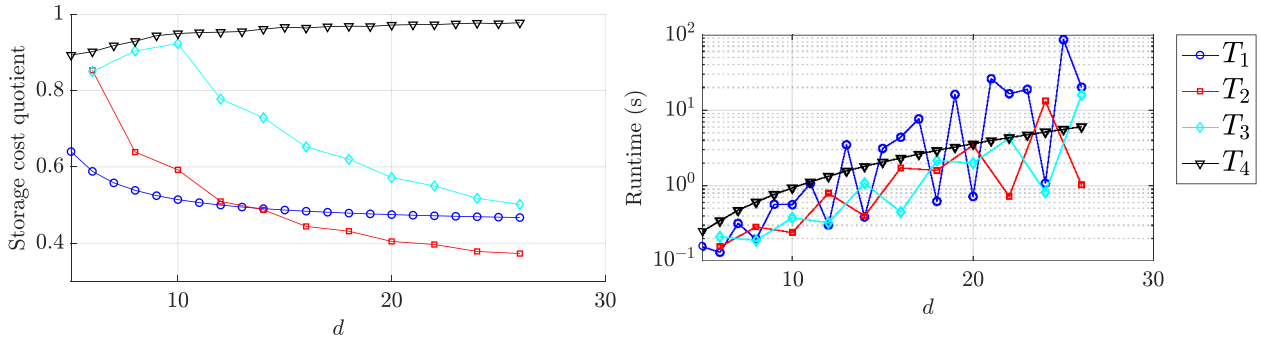

FIGURE 2 Left: Storage cost of tensor ring representation of tensors in Section 5.3 as fraction of storage cost in tensor train format and as function of d. Right: runtime in seconds

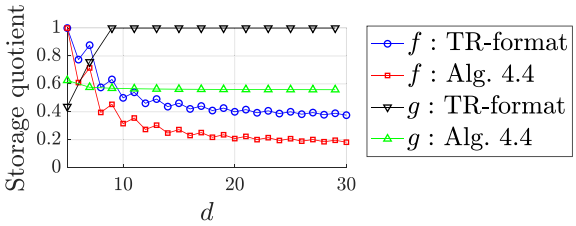

FIGURE 3 Storage cost of representations of tensors in Section 5.4 as fraction of storage cost in tensor train format and as function of d.

given in the canonical format. The grid $[0,1]^d$ used $n = 2^5$ grid points in each dimension. The set of permissible graph structures $\mathbb{G}$ for f and g were set to be all cycles with vertices $i_1, \dots, i_d$ and exactly zero or one chords, and all chains with edges between every other index i_k, i_{k+2}, for $k = 1, 3, 5, \dots, d-2$, respectively. For f, we repeated the application of Algorithm 9 to compare all graphs with exactly one chord. The initial graph format was the TR-format and the TT-format, respectively. The rounding accuracy was set to $\epsilon = 10^{-9}$. The result is presented in Figure 3 and clearly shows storage savings compared to the TT- and TR-formats.

6 | CONCLUSIONS

In this paper, we studied efficient tensor representations and computation in the tensor ring format. Firstly, we showed theoretically and numerically how two degrees of freedom in SVD-based algorithms for converting a tensor in full format into TR-format are crucial for obtaining low-cost representations. A heuristic algorithm achieving a low-cost representation with low runtime was introduced and tested numerically. Secondly, we presented a rounding procedure for the tensor ring format and showed how this required redefining common linear algebra operations to obtain a reduction of the storage-cost. Lastly, we devised algorithms for transforming the graph structure of a tensor in a graph-based format, producing even higher compression ratios. Numerical examples achieved up to more than an order of magnitude higher compression ratios than previous approaches to using the tensor ring format, without significantly affecting the runtime. An important direction for future work would be to also extend our algorithms to the case of incomplete tensors, where not all entries are used for the decomposition into the tensor-ring format.[26] Another important direction is to obtain a more thorough understanding of the geometric and algebraic properties of the TR-format.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation grant no 1350685, the Army Research Office grant no W911NF1510249 and the ARL DCIST program.

CONFLICT OF INTEREST

This work does not have any conflicts of interest.

ORCID

Oscar Mickelin https://orcid.org/0000-0003-0167-1992

REFERENCES

1. Hitchcock FL. The expression of a tensor or a polyadic as a sum of products. J Math Phys. 1927;6(1-4):164–189.
2. Tucker LR. Some mathematical notes on three-mode factor analysis. Psychometrika. 1966;31(3):279–311.
3. Novikov A, Podoprikhin D, Osokin A, Vetrov DP. Tensorizing neural networks. Advances in neural information processing systems. New York: Curran Associates; 2015;442-450.
4. Wang W, Sun Y, Eriksson B, Wang W, Aggarwal V. Wide compression: Tensor ring nets; 2018. arXiv preprint arXiv:180209052.
5. Yuan L, Zhao Q, Gui L, Cao J. High-order tensor completion via gradient-based optimization under tensor train format. Signal Process Image Commun. 2018;73:53-61.
6. Oseledets IV. Tensor-train decomposition. SIAM J Sci Comput. 2011;33(5):2295–2317.
7. Khoromskij BN. $\mathcal{O}(dlogn)$-Quantics approximation of N-d tensors in high-dimensional numerical modeling. Constr Approx. 2011;34(2):257–280.
8. Dolgov S, Khoromskij BN, Oseledets IV. Fast solution of parabolic problems in the tensor train/quantized tensor train format with initial application to the Fokker–Planck equation. SIAM J Sci Comput. 2012;34(6):A3016–A3038.
9. Khoromskaia V, Khoromskij B. Fast tensor method for summation of long-range potentials on 3D lattices with defects. Numer Linear Algebra Appl. 2016;23(2):249–271.
10. Dolgov S, Khoromskij B. Simultaneous state-time approximation of the chemical master equation using tensor product formats. Numer Linear Algebra Appl. 2015;22(2):197–219.
11. Hackbusch W, Khoromskij B, Sauter S, Tyrtyshnikov EE. Use of tensor formats in elliptic eigenvalue problems. Numer Linear Algebra Appl. 2012;19(1):133–151.
12. Oseledets IV, Savostyanov DV, Tyrtyshnikov EE. Cross approximation in tensor electron density computations. Numer Linear Algebra Appl. 2010;17(6):935–952.
13. Grasedyck L, Löbbert C. Distributed hierarchical SVD in the hierarchical tucker format. Numer Linear Algebra Appl. 2018;25(6):e2174.
14. Boussé M, Vervliet N, Domanov I, Debals O, De Lathauwer L. Linear systems with a canonical polyadic decomposition constrained solution: Algorithms and applications. Numer Linear Algebra Appl. 2018;25(6):e2190.
15. Kolda TG, Bader BW. Tensor decompositions and applications. SIAM Rev. 2009;51(3):455–500.
16. Cichocki A, Lee N, Oseledets I, et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. Found Trends Mach Learn. 2016;9(4-5):249–429.
17. Cichocki A, Phan AH, Zhao Q, et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives. Found Trends Mach Learn. 2017;9(6):431–673.
18. Affleck I, Kennedy T, Lieb EH, Tasaki H. Valence bond ground states in isotropic quantum antiferromagnets. Condensed matter physics and exactly soluble models. New York, NY: Springer, 1988; p. 253–304.
19. Perez-Garcia D, Verstraete F, Wolf MM, Cirac JI. Matrix product state representations; 2006. arXiv preprint quant-ph/0608197.
20. Orús R. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. Ann Phys. 2014;349:117–158.
21. Schollwöck U. The density-matrix renormalization group in the age of matrix product states. Ann Phys. 2011;326(1):96–192.
22. Khoromskij BN. Tensors-structured numerical methods in scientific computing: Survey on recent advances. Chemom Intell Lab Syst. 2012;110(1):1–19.
23. Landsberg JM, Qi Y, Ye K. On the geometry of tensor network states; 2011. arXiv preprint arXiv:11054449.
24. Ye K, Lim LH. Tensor network ranks; 2018. arXiv preprint arXiv:180102662.
25. De Silva V, Lim LH. Tensor rank and the ill-posedness of the best low-rank approximation problem. SIAM J Matrix Anal Appl. 2008;30(3):1084–1127.
26. Khoo Y, Lu J, Ying L. Efficient construction of tensor ring representations from sampling; 2017. arXiv preprint arXiv:171100954.
27. Wang W, Aggarwal V, Aeron S. Efficient low rank tensor ring completion. Paper presented at: Proceedings of the IEEE International Conference on Computer Vision IEEE; 2017. p. 5698–5706.
28. Zhao Q, Sugiyama M, Cichocki A. Learning efficient tensor representations with ring structure networks; 2017. arXiv preprint arXiv:170508286.
29. Zhao Q, Zhou G, Xie S, Zhang L, Cichocki A. Tensor ring decomposition; 2016. arXiv preprint arXiv:160605535.
30. Espig M, Naraparaju KK, Schneider J. A note on tensor chain approximation. Comput Vis Sci. 2012;15(6):331–344.
31. Handschuh S. Changing the topology of tensor networks; 2012. arXiv preprint arXiv:12031503.
32. Yuan L, Li C, Mandic D, Cao J, Zhao Q. Rank minimization on tensor ring: A new paradigm in scalable tensor decomposition and completion; 2018. arXiv preprint arXiv:180508468.
33. Chen Z, Li Y, Lu J. Tensor ring decomposition: Energy landscape and one-loop convergence of alternating least squares; 2019. arXiv e-prints;p. arXiv:1905.07101.
34. Hackbusch W. Tensor spaces and numerical tensor calculus. Vol 42. Berlin: Springer Science & Business Media; 2012.
35. Lang S. Algebra, volume 211 of graduate texts in mathematics. New York, NY: Springer-Verlag, 2002.
36. Batselier K. The trouble with tensor ring decompositions; 2018. arXiv preprint arXiv:181103813.

37. Surjanovic S, Bingham D. Virtual library of simulation experiments: Test functions and datasets. [cited 2018, July 2]. Available from: http://www.sfu.ca/~ssurjano.
38. Nene SA, Nayar SK, Murase H. Columbia Object Image Library (COIL-100). Technical report CUCS-006-96; 1996.
39. Oseledets IV, Dolgov S. Solution of linear systems and matrix inversion in the TT-format. SIAM J Sci Comput. 2012;34(5):A2718–A2739.

How to cite this article: Mickelin O, Karaman S. On algorithms for and computing with the tensor ring decomposition. *Numer Linear Algebra Appl*. 2020;e2289. https://doi.org/10.1002/nla.2289