SIAM J. MATRIX ANAL. APPL.
Vol. 39, No. 3, pp. 1221–1244

COMPUTING LOW-RANK APPROXIMATIONS OF LARGE-SCALE MATRICES WITH THE TENSOR NETWORK RANDOMIZED SVD*

KIM BATSELIER†, WENJIAN YU‡, LUCA DANIEL§, AND NGAI WONG†

Abstract. We propose a new algorithm for the computation of a singular value decomposition (SVD) low-rank approximation of a matrix in the matrix product operator (MPO) format, also called the tensor train matrix format. Our tensor network randomized SVD (TNrSVD) algorithm is an MPO implementation of the randomized SVD algorithm that is able to compute dominant singular values and their corresponding singular vectors. In contrast to the state-of-the-art tensor-based alternating least squares SVD (ALS-SVD) and modified alternating least squares SVD (MALS-SVD) matrix approximation methods, TNrSVD can be up to 13 times faster while achieving better accuracy. In addition, our TNrSVD algorithm also produces accurate approximations in particular cases where both ALS-SVD and MALS-SVD fail to converge. We also propose a new algorithm for the fast conversion of a sparse matrix into its corresponding MPO form, which is up to 509 times faster than the standard tensor train SVD method while achieving machine precision accuracy. The efficiency and accuracy of both algorithms are demonstrated in numerical experiments.

Key words. curse of dimensionality, low-rank tensor approximation, matrix factorization, matrix product operator, singular value decompositon (SVD), tensor network, tensor train (TT) decomposition, randomized algorithm

AMS subject classifications. 15A69, 15A18, 15A23, 68W20

DOI. 10.1137/17M1140480

1. Introduction.

When Beltrami established the existence of the singular value decomposition (SVD) in 1873 [2],[1] he probably had not foreseen that this matrix factorization would become a crucial tool in scientific computing and data analysis [3, 9, 12, 15]. Among the many applications of the SVD are the determination of the numerical rank and condition number of a matrix, the computation of low-rank approximations and pseudoinverses, and solving linear systems. These applications have found widespread usage in many fields of science and engineering [12, 22]. Matrices with low numerical ranks appear in a wide variety of scientific applications. For these matrices, finding a low-rank approximation allows them to be stored inexpensively without much loss of accuracy. It is not uncommon for matrices in data analysis to be very large and classical methods to compute the SVD [13, 14, 15] can be ill-suited to handle large matrices. One proposed solution to compute a low-rank approximation of large data matrices is to use randomized algorithms [16, 21, 33]. An attractive feature of these algorithms is that they require only a constant number of passes over the data. It is even possible to find a matrix approximation with a single pass over

*Received by the editors July 25, 2017; accepted for publication (in revised form) by T. G. Kolda May 4, 2018; published electronically August 7, 2018.
http://www.siam.org/journals/simax/39-3/M114048.html
Funding: This work was supported by the Hong Kong Research Grants Council under General Research Fund (GRF) Project 17246416.

†Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong (kimb@eee.hku.hk, nwong@eee.hku.hk).
‡BNRist, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (yu-wj@tsinghua.edu.cn).
§Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (MIT), Cambridge, MA 02139 (luca@mit.edu).

[1]The original paper was written in Italian; for a recent English translation we refer the reader to [22, pp. 5–18].

the data [35]. This enables the efficient computation of a low-rank approximation of dense matrices that cannot be stored completely in fast memory [34].

Another way to handle large matrices is to use a different data storage representation. A matrix product operator (MPO), also called Tensor Train (TT) matrix [27], is a particular tensor network representation of a matrix that originates from the simulation of one-dimensional quantum-lattice systems [32]. This representation transforms the storage complexity of an $n^d \times n^d$ matrix into $O(dn^2r^2)$, where r is the maximal MPO-rank, effectively transforming the exponential dependence on d into a linear one. The main idea of this representation is to replace the storage of a particular matrix element by a product of small matrices. An efficient representation is then found when only a few small matrices are required. A Matrix Product State (MPS), also called a tensor train [25], is a similar tensor network representation of a vector. These tensor network representations have gained more interest over the past decade, together with their application to various problems [1, 4, 5, 6, 20, 24, 28, 29]. In particular, finding a low-rank approximation of a matrix based on the truncated SVD in the MPO representation is addressed in [19] with the development of the alternating least squares SVD (ALS-SVD) and modified alternating least squares SVD (MALS-SVD) methods. These two methods are shown to be able to compute a few extreme singular values of $2^{50} \times 2^{50}$ matrix accurately in a few seconds on desktop computers. Other related work is found in [10], where an ALS algorithm is developed based on the block TT format for block Rayleigh quotient minimization of symmetric matrices. In [11], the ALS method was extended with a basis enrichment step, chosen in accordance with the steepest descent algorithm, which can be seen as a computationally cheap approximation of the MALS method. In this article, we propose a randomized tensor network algorithm for the computation of a low-rank approximation of a matrix based on the truncated SVD. As we will demonstrate through numerical experiments, our proposed algorithm manages to achieve the same accuracy up to 13 times faster than MALS-SVD. Moreover, our algorithm is able to retrieve accurate approximations for cases where both ALS-SVD and MALS-SVD fail to converge. More specifically, the main contributions of this article are twofold.

1. We present a fast algorithm that is able to convert a given sparse matrix into MPO form with machine precision accuracy.
2. We present a MPO version of the randomized SVD algorithm that can outperform the current state-of-the-art tensor algorithms [19] for computing low-rank matrix approximations of large-scale matrices.[2]

This article is organized as follows. In section 2, some basic tensor concepts and notation are explained. We introduce the notion of MPOs in section 3. Our newly proposed algorithm to convert a matrix into the MPO representation is presented in section 4. In section 5, we present our randomized algorithm to compute an SVD low-rank approximation of a given matrix in MPO form. Numerical experiments in section 6 demonstrate both the fast matrix to MPO conversion as well as our randomized algorithm. We compare the performance of our matrix conversion algorithm with both the TT-SVD [25] and TT-cross [30] algorithms, while also comparing the performance of our randomized algorithm with the ALS-SVD and MALS-SVD algorithms [19]. Section 7 presents some conclusions together with an avenue of future research.

[2]MATLAB implementations of all algorithms are distributed under a GNU lesser general public license and can be freely downloaded from https://github.com/kbatseli/TNrSVD.

2. Tensor basics and notation. Tensors in this article are multi-dimensional arrays with entries either in the real or complex field. We denote scalars by italic letters, vectors by boldface italic letters, matrices by boldface capitalized italic letters, and higher-order tensors by boldface calligraphic italic letters. The number of indices required to determine an entry of a tensor is called the order of the tensor. A dth order or d-way tensor is hence denoted $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$. An index i_k always satisfies $1 \leq i_k \leq I_k$, where I_k is called the dimension of that particular mode. We use the MATLAB array index notation to denote entries of tensors. Suppose that $\boldsymbol{\mathcal{A}}$ is a 4-way tensor with entries $\boldsymbol{\mathcal{A}}(i_1, i_2, i_3, i_4)$. Grouping indices together into multi-indices is one way of reshaping the tensor. For example, a 3-way tensor can now be formed from $\boldsymbol{\mathcal{A}}$ by grouping the first two indices together. The entries of this 3-way tensor are then denoted by $\boldsymbol{\mathcal{A}}([i_1 i_2], i_3, i_4)$, where the multi-index $[i_1 i_2]$ is easily converted into a single index as $i_1 + I_1(i_2 - 1)$. Grouping the indices into $[i_1]$ and $[i_2 i_3 i_4]$ results in an $I_1 \times I_2 I_3 I_4$ matrix with entries $\boldsymbol{\mathcal{A}}(i_1, [i_2 i_3 i_4])$. The column index $[i_2 i_3 i_4]$ is equivalent to the linear index $i_2 + I_2(i_3 - 1) + I_2 I_3(i_4 - 1)$. In general, we define a multi-index $[i_1 i_2 \cdots i_d]$ as

$$[i_1 i_2 \cdots i_d] := i_1 + \sum_{k=2}^{d} (i_k - 1) \prod_{l=1}^{k-1} I_l. \tag{2.1}$$

Grouping indices together in order to change the order of a tensor is called reshaping and is an often used tensor operation. We adopt the MATLAB/Octave reshape operator "reshape($\boldsymbol{\mathcal{A}}, [I_1, I_2, \ldots, I_d]$)," which reshapes the d-way tensor $\boldsymbol{\mathcal{A}}$ into a tensor with dimensions $I_1 \times I_2 \times \cdots \times I_d$. The total number of elements of $\boldsymbol{\mathcal{A}}$ must be the same as $I_1 \times I_2 \times \cdots \times I_d$. The mode-$n$ matricization $\boldsymbol{\mathcal{A}}_{(n)}$ of a d-way tensor $\boldsymbol{\mathcal{A}}$ maps the entry $\boldsymbol{\mathcal{A}}(i_1, i_2, \ldots, i_d)$ to the matrix element with row index i_n and column index $[i_1 \cdots i_{n-1} i_{n+1} \cdots i_d]$.

Example 1. We illustrate the reshaping operator on the $4 \times 3 \times 2$ tensor $\boldsymbol{\mathcal{A}}$ that contains all entries from 1 up to 24. Its mode-1 matricization is

$$\boldsymbol{\mathcal{A}}_{(1)} = \text{reshape}(\boldsymbol{\mathcal{A}}, [4, 6]) = \begin{pmatrix} 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \\ 4 & 8 & 12 & 16 & 20 & 24 \end{pmatrix}.$$

Another important reshaping of a tensor $\boldsymbol{\mathcal{A}}$ is its vectorization, denoted as vec($\boldsymbol{\mathcal{A}}$) and obtained from grouping all indices into one multi-index.

Example 2. For the tensor of Example 1, we have

$$\text{vec}(\boldsymbol{\mathcal{A}}) \ = \text{reshape}(\boldsymbol{\mathcal{A}}, [24, 1]) = \begin{pmatrix} 1 & 2 & \cdots & 24 \end{pmatrix}^T.$$

Suppose we have two d-way tensors $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_d}$. The Kronecker product $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} \otimes \boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 J_1 \times I_2 J_2 \times \cdots \times I_d J_d}$ is then a d-way tensor such that

$$\boldsymbol{\mathcal{C}}([j_1 i_1], [j_2 i_2], \ldots, [j_d i_d]) = \boldsymbol{\mathcal{A}}(i_1, i_2, \ldots, i_d)\, \boldsymbol{\mathcal{B}}(j_1, j_2, \ldots, j_d). \tag{2.2}$$

Similarly, the outer product $\boldsymbol{\mathcal{D}} = \boldsymbol{\mathcal{A}} \circ \boldsymbol{\mathcal{B}}$ of the d-way tensors $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$ is a $2d$-way tensor of dimensions $I_1 \times \cdots \times I_d \times J_1 \times \cdots \times J_d$ such that

$$\boldsymbol{\mathcal{D}}(i_1, i_2, \ldots, i_d, j_1, j_2, \ldots, j_d) = \boldsymbol{\mathcal{A}}(i_1, i_2, \ldots, i_d)\, \boldsymbol{\mathcal{B}}(j_1, j_2, \ldots, j_d). \tag{2.3}$$

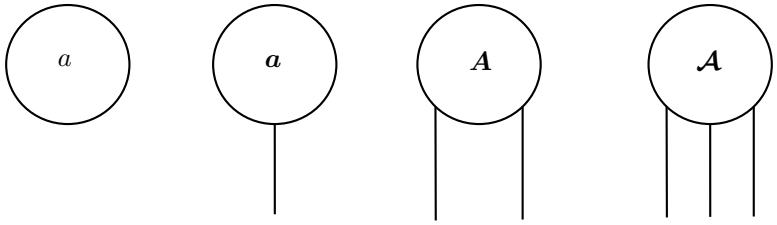

FIG. 2.1. *Graphical depiction of a scalar a, vector $\boldsymbol{a}$, matrix $\boldsymbol{A}$, and 3-way tensor $\boldsymbol{\mathcal{A}}$.*

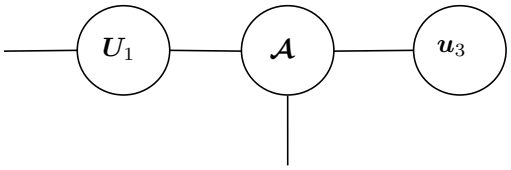

FIG. 2.2. *Summation over the first and third index of $\boldsymbol{\mathcal{A}}$ represented by connected lines in the tensor network graph.*

From (2.2) and (2.3) one can see that the Kronecker and outer products are interrelated through a reshaping and a permutation of the indices. A very convenient graphical representation of d-way tensors is shown in Figure 2.1. Tensors are here represented by circles and each "leg" denotes a particular mode of the tensor. The order of the tensor is then easily determined by counting the number of legs. Since a scalar is a zeroth-order tensor, it is represented by a circle without any lines. One of the most important operations on tensors is the summation over indices, also called contraction of indices. For example, the following mode product [18] of a 3-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I\times I\times I}$ with a matrix $\boldsymbol{U}_1 \in \mathbb{R}^{R\times I}$ and a vector $\boldsymbol{u}_3 \in \mathbb{R}^I$

$$\boldsymbol{\mathcal{A}} \times_1 \boldsymbol{U}_1 \times_3 \boldsymbol{u}_3^T = \sum_{i,j} \boldsymbol{\mathcal{A}}(i,:,j)\, \boldsymbol{U}_1(:,i)\, \boldsymbol{u}_3(j)$$

is graphically depicted in Figure 2.2 by connected lines between $\boldsymbol{\mathcal{A}}, \boldsymbol{U}_1$, and $\boldsymbol{u}_3$. Figure 2.2 also illustrates a simple tensor network, which is a collection of tensors that are interconnected through contractions. The tensor network in Figure 2.2 has two legs, which indicate that the network represents a matrix. This article uses a very particular tensor network structure, the MPO structure.

3. Matrix product operators. In this section, we give a brief introduction to the notion of MPOs. Simply put, an MPO is a linear chain of 4-way tensors that represents a matrix. This construction was originally used to represent a linear operator acting on a multi-body quantum system. Since their introduction to the scientific community in 2010 [27], MPOs have found many other applications. We now discuss the MPO representation of a matrix through an illustrative example. Suppose that we have a matrix $\boldsymbol{A}$ of size $I_1I_2I_3I_4 \times J_1J_2J_3J_4$, as shown in Figure 3.1. This matrix $\boldsymbol{A}$ can be represented by an MPO of four 4-way tensors, where the first tensor $\boldsymbol{\mathcal{A}}^{(1)}$ has dimensions $R_1 \times I_1 \times J_1 \times R_2$. Similarly, the kth tensor in the MPO of Figure 3.1 hence has dimensions $R_k \times I_k \times J_k \times R_{k+1}$. We require that $R_1 = R_5$, which ensures that the contraction of this particular MPO results in a 8-way tensor with entries $\boldsymbol{\mathcal{A}}(i_1, j_1, i_2, j_2, i_3, j_3, i_4, j_4)$. This requirement is indicated in Figure 3.1 by the

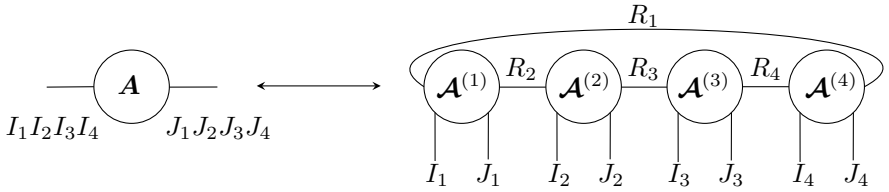

FIG. 3.1. *Representation of an $I_1I_2I_3I_4 \times J_1J_2J_3J_4$ matrix $\boldsymbol{A}$ as an MPO.*

connecting edge between $\boldsymbol{\mathcal{A}}^{(1)}$ and $\boldsymbol{\mathcal{A}}^{(4)}$. For the remainder of this article, we assume that $R_1 = R_5 = 1$, which is also called an open boundary condition. An MPO that satisfies $R_1 = R_5 > 1$ has a periodic boundary condition. The tensor obtained from doing all summations in the network can then be permuted and reshaped back into the original matrix with entries $\boldsymbol{A}([i_1i_2i_3i_4],[j_1j_2j_3j_4])$. The dimensions R_k, R_{k+1} of the connecting indices in an MPO are called the MPO-ranks and play a crucial role in the computational complexity of our developed algorithms. The MPO-ranks are called canonical if they attain their minimal value such that the MPO represents a given matrix exactly. A very special MPO is obtained when all MPO-ranks are unity. The contraction of a rank-1 MPO corresponds with the outer product of the individual MPO-tensors. Indeed, suppose that the MPO-ranks in Figure 3.1 are all unity. The 4-way tensors of the MPO are then reduced to matrices such that we can write $\boldsymbol{A}^{(k)} := \boldsymbol{\mathcal{A}}^{(k)}(1,:,:,1)\,(k = 1,\ldots,4)$ and

$$\boldsymbol{\mathcal{A}}(i_1,j_1,i_2,j_2,i_3,j_3,i_4j_4) = \boldsymbol{A}^{(1)}(i_1,j_1)\,\boldsymbol{A}^{(2)}(i_2,j_2)\,\boldsymbol{A}^{(3)}(i_3,j_3)\,\boldsymbol{A}^{(4)}(i_4,j_4),$$

which is exactly the outer product of the matrices $\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}$ with $\boldsymbol{A}^{(4)}$ into a 8-way tensor. The relation between the Kronecker and outer products, together with the previous example leads to the following important theorem.

THEOREM 3.1. *A matrix $\boldsymbol{A} \in \mathbb{R}^{I_1I_2\cdots I_d \times J_1J_2\cdots J_d}$ that satisfies*

$$\boldsymbol{A} = \boldsymbol{A}^{(d)} \otimes \cdots \otimes \boldsymbol{A}^{(2)} \otimes \boldsymbol{A}^{(1)}$$

has an MPO representation where the kth MPO-tensor is $\boldsymbol{A}^{(k)} \in \mathbb{R}^{1\times I_k \times J_k \times 1}\,(k = 1,\ldots,d)$ with unit canonical MPO-ranks.

It is important to note that the order of the MPO-tensors is reversed with respect to the order of the factor matrices in the Kronecker product. This means that the last factor matrix $\boldsymbol{A}^{(1)}$ in the Kronecker product of Theorem 3.1 is the first tensor in the corresponding MPO representation. Theorem 3.1 can also be written in terms of the matrix entries as

$$\boldsymbol{A}([i_1i_2\cdots i_d],[j_1j_2\cdots j_d]) = \boldsymbol{A}^{(d)}(i_d,j_d)\cdots\boldsymbol{A}^{(2)}(i_2,j_2)\,\boldsymbol{A}^{(1)}(i_1,j_1).$$

As mentioned earlier, the MPO-ranks play a crucial role in the computational complexity of the algorithms. For this reason, only MPOs with small ranks are desired. An upper bound on the canonical MPO-rank R_k for an MPO of d tensors for which $R_1 = R_{d+1} = 1$ is given by the following theorem.

THEOREM 3.2 (Modified version of Theorem 2.1 in [30]). *For any matrix $\boldsymbol{A} \in \mathbb{R}^{I_1I_2\cdots I_d \times J_1J_2\cdots J_d}$ there exists an MPO with MPO-ranks $R_1 = R_{d+1} = 1$ such that the canonical MPO-ranks R_k satisfy*

$$R_k \leq \min\left(\prod_{i=1}^{k-1} I_i J_i, \prod_{i=k}^{d} I_i J_i\right) \text{ for } k = 2, \ldots, d.$$

Proof. The upper bound on the canonical MPO-rank R_k can be determined from contracting the first $k-1$ tensors of the MPO together and reshape the result into the $I_1 J_1 \cdots I_{k-1} J_{k-1} \times R_k$ matrix $\boldsymbol{B}$. Similarly, we can contract the kth tensor of the MPO with all other remaining tensors and reshape the result into the $R_k \times I_k J_k \cdots I_d J_d$ matrix $\boldsymbol{C}$. The canonical rank R_k is now upper bounded by the matrix product $\boldsymbol{BC}$ due to $\text{rank}(\boldsymbol{BC}) \leq \min(\text{rank}(\boldsymbol{B}), \text{rank}(\boldsymbol{C}))$. ∎

These upper bounds are quite pessimistic and are attained for generic matrices. For example, a generic full-rank $I^{10} \times I^{10}$ matrix has an exact MPO representation of 10 MPO-tensors with $I_k = J_k = I$ and a canonical MPO-rank $R_6 = I^{10}$. This implies that any MPO representation with $R_6 < I^{10}$ will consequently be an approximation of that generic matrix. Sparse and structured matrices typically have canonical MPO-ranks that are lower than the upper bounds of Theorem 3.2. Keeping Theorem 3.2 in mind, we conclude that in order to keep computations with MPOs feasible it is required that the matrix under consideration has small canonical MPO-ranks or can be sufficiently approximated by a low-rank MPO.

Two important MPO operations are addition and rounding, which are easily generalized from MPS addition and rounding. Indeed, by grouping the I_k and J_k indices together, one effectively transforms the MPO into an MPS such that MPS addition and rounding can be applied. The addition of two MPOs concatenates corresponding tensors and results in an MPO for which the corresponding ranks, except R_1 and R_{d+1}, are added. The SVD-based rounding operation, very similar to Algorithms 5.2 and 5.3, repeatedly uses a truncated SVD on each of the MPO-tensors going from left-to-right and right-to-left in order to reduce the ranks R_k such that a specific relative error tolerance is satisfied. For more details on MPS addition and rounding we refer the reader to [25, p. 2305] and [25, p. 2308], respectively. An alternative MPO rounding operation that does not require the SVD but relies on removing parallel vectors is described in [17]. This alternative rounding procedure can be computationally more efficient for MPOs that consist of sparse tensors.

4. Converting a sparse matrix into an MPO. The standard way to convert a matrix into MPO form is the TT-SVD algorithm [25, p. 2301], which relies on consecutive reshapings of the matrix from which an SVD needs to be computed. This procedure is not recommended for matrices in real applications for two reasons. First, application-specific matrices tend to be sparse and computing the SVD of a sparse matrix destroys the sparsity, which results in requiring more and often prohibitive storage. Second, real-life matrices are typically so large that it is infeasible to compute their SVD. An alternative method to convert a matrix into an MPO is via cross approximation [30]. This method relies on heuristics to find subsets of indices for all modes of a tensor in order to approximate it. In practice, the cross approximation method can be very slow and not attain the desired accuracy. Our matrix to MPO conversion method relies on a partitioning of the sparse matrix such that it is easily written as the addition of rank-1 MPOs.

4.1. Algorithm derivation. We derive our algorithm with the following illustrative example. Suppose we have a sparse matrix $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ with the following sparsity pattern:

$$\begin{pmatrix} 0 & 0 & 0 & \boldsymbol{A}_{14} \\ 0 & \boldsymbol{A}_{22} & 0 & 0 \\ 0 & 0 & \boldsymbol{A}_{33} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Assume that each of the nonzero block matrices has dimensions $I_1 \times J_1$ and that $I = I_1 I_2, J = J_1 J_2$ such that the rows and columns of $\boldsymbol{A}$ are now indexed by $[i_1 i_2]$ and $[j_1 j_2]$, respectively. The main idea of our method is to convert each nonzero block matrix into a rank-1 MPO and add them all together. Observe now that

$$\begin{pmatrix} 0 & 0 & 0 & \boldsymbol{A}_{14} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \boldsymbol{E}_{14} \otimes \boldsymbol{A}_{14},$$

where $\boldsymbol{E}_{14} \in \mathbb{R}^{I_2 \times J_2}$ is a matrix of zeros except for $\boldsymbol{E}_{14}(1,4) = 1$. From Theorem 3.1 we know that $\boldsymbol{E}_{14} \otimes \boldsymbol{A}_{14}$ is equivalent with a rank-1 MPO where the first MPO-tensor is $\boldsymbol{A}_{14}$ and the second MPO-tensor is $\boldsymbol{E}_{14}$. Generalizing the matrix $\boldsymbol{E}_{14}$ to the matrix $\boldsymbol{E}_{ij}$ of zero entries except for $\boldsymbol{E}_{ij}(i,j) = 1$ allows us to write

$$\boldsymbol{A} = \boldsymbol{E}_{14} \otimes \boldsymbol{A}_{14} + \boldsymbol{E}_{22} \otimes \boldsymbol{A}_{22} + \boldsymbol{E}_{33} \otimes \boldsymbol{A}_{33}, \tag{4.1}$$

from which we conclude that the MPO representation of $\boldsymbol{A}$ is found from adding the unit-rank MPOs of each of the terms. Another important conclusion is that the MPO-rank for the particular MPO obtained from this algorithm is the total number of summations. The number of factors in the Kronecker product is not limited to two and depends on the matrix partitioning. Indeed, suppose we can partition $\boldsymbol{A}_{14}$ further into

$$\boldsymbol{A}_{14} = \begin{pmatrix} 0 & 0 \\ \boldsymbol{X}_{14} & 0 \end{pmatrix} = \boldsymbol{E}_{21} \otimes \boldsymbol{X}_{14};$$

then the first term of (4.1) becomes $\boldsymbol{E}_{14} \otimes \boldsymbol{E}_{21} \otimes \boldsymbol{X}_{14}$ and likewise for the other terms. A crucial element is that the matrix $\boldsymbol{A}$ is partitioned into block matrices of equal size, which is required for the addition of the MPOs. In general, for a given matrix $\boldsymbol{A} \in \mathbb{R}^{I \times J}$, we consider the partitioning of $\boldsymbol{A}$ determined by a Kronecker product of d matrices

$$\boldsymbol{A}^{(d)} \otimes \boldsymbol{A}^{(d-1)} \otimes \cdots \otimes \boldsymbol{A}^{(2)} \otimes \boldsymbol{A}^{(1)}$$

with $\boldsymbol{A}^{(k)} \in \mathbb{R}^{I_k \times J_k}$ $(k = 1, \ldots, d)$ and $I = \prod_{k=1}^{d} I_k, J = \prod_{k=1}^{d} J_k$. The algorithm to convert a sparse matrix into an MPO is given in pseudocode in Algorithm 4.1. The MATLAB implementation of Algorithm 4.1 in the tensor network randomized SVD (TNrSVD) package is matrix2mpo.m.

ALGORITHM 4.1. *Sparse matrix to MPO conversion*
***Input**: matrix $\boldsymbol{A}$, dimensions $I_1, \ldots, I_d, J_1, \ldots, J_d$.*
***Output**: MPO $\boldsymbol{\mathcal{A}}$ with tensors $\boldsymbol{\mathcal{A}}^{(1)}, \ldots, \boldsymbol{\mathcal{A}}^{(d)}$.*
  Initialize MPO $\boldsymbol{\mathcal{A}}$ with zero tensors.
  for** all nonzero matrix blocks $\boldsymbol{X} \in \mathbb{R}^{I_1 \times J_1}$ **do
    Determine $d-1$ $\boldsymbol{E}_{ij}$ matrices.
    Construct rank-1 MPO $\boldsymbol{\mathcal{T}}$ with $\boldsymbol{X}$ and $\boldsymbol{E}_{ij}$ matrices.
    $\boldsymbol{\mathcal{A}} \leftarrow \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{T}}$
  end for

4.2. Algorithm properties. Having derived our sparse matrix conversion algorithm, we now discuss some of its properties. Algorithm 4.1 has the following nice features, some of which we will address in more detail.

- Except for $\boldsymbol{\mathcal{A}}^{(1)}$, almost all the MPO-tensors will be sparse.
- The user is completely free to decide on how to partition the matrix $\boldsymbol{A}$, which determines the number of tensors in the resulting MPO.
- The generalization of Algorithm 4.1 to construct an MPO representation of a given tensor is straightforward.
- The maximal number of tensors in an MPO representation are easily deduced and given in Lemma 4.1.
- The obtained MPO-rank for a particular partitioning of the matrix $\boldsymbol{A}$ is also easily deduced and given in Lemma 4.2.
- A lower bound on the obtained MPO-rank for a fixed block size I_1, J_1 is derived in Lemma 4.3.
- As the dimensions of each of the MPO-tensors are known a priori, one can preallocate the required memory to store the final tensors in advance. Since the addition of MPOs can be done by concatenation of the respective tensors, no actual computation is required, which allows a fast execution of Algorithm 4.1.

The maximal number of tensors in an MPO representation of a matrix is determined by choosing a partitioning such that each block matrix of $\boldsymbol{A}$ becomes a single scalar entry and is given by the following lemma.

LEMMA 4.1. *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{I\times J}$, suppose d_I and d_J are the number of factors in the prime factorizations of I and J, respectively. Then the maximal number of tensors in an MPO representation of $\boldsymbol{A}$ is* $\max(d_I, d_J) + 1$.

The simple example that follows illustrates the maximal number of tensors $\max(d_I, d_J) + 1$ from Lemma 4.1.

Example 3. Let

$$\boldsymbol{A} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -5 & 0 & 0 \end{pmatrix} \in \mathbb{R}^{2\times 6}.$$

Then the prime factorizations are $2 = 2$ and $6 = 2 \times 3$, which sets $d_I = 1, d_J = 2$ and the maximal number of tensors in the MPO of $\boldsymbol{A}$ is $\max(1, 2) + 1 = 2 + 1 = 3$. Indeed, by setting $I_1 = 1, J_1 = 1, I_2 = 1, J_2 = 3, I_3 = 2, J_3 = 2$ we can write

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \otimes 2 + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \otimes -5.$$

We therefore have $\boldsymbol{\mathcal{A}}^{(1)} \in \mathbb{R}^{1\times 1\times 1\times 2}, \boldsymbol{\mathcal{A}}^{(2)} \in \mathbb{R}^{2\times 1\times 3\times 2}$, and $\boldsymbol{\mathcal{A}}^{(3)} \in \mathbb{R}^{2\times 2\times 2\times 1}$. Theorem 3.2 states that the canonical MPO-ranks satisfy $R_2 \leq 1, R_3 \leq 3$, which demonstrates that the MPO-ranks obtained from Algorithm 4.1 are not necessarily minimal.

The rank of the MPO obtained from Algorithm 4.1 for a particular partitioning of the matrix $\boldsymbol{A}$ is given by the following lemma.

LEMMA 4.2. *The MPO obtained from Algorithm* 4.1 *has a uniform MPO-rank equal to the total number of nonzero matrix blocks $\boldsymbol{X}$ as determined by the partitioning of $\boldsymbol{A}$.*

Lemma 4.2 follows trivially from the fact that ranks are added in MPO addition and all MPOs in Algorithm 4.1 are unit-rank. It is important to realize that the usage of Algorithm 4.1 is not limited to sparse matrices per se. One could apply Algorithm 4.1 to dense matrices but then the possible computational benefit of having to process only a few nonzero matrix blocks $\boldsymbol{X}$ is lost. It is also the case that the MPO-ranks can be reduced in almost all cases via a rounding procedure without the loss of any accuracy, as the upper bounds of Theorem 3.2 are usually exceeded. Partitioning the matrix $\boldsymbol{A}$ such that each term in Algorithm 4.1 corresponds with a single scalar entry sets the resulting MPO-rank to the total number of nonzero entries of $\boldsymbol{A}$. This might be too high in practice. On the other hand, choosing any I_k, J_k too large results in a large MPO-tensor $\boldsymbol{\mathcal{A}}^{(k)}$, which is also not desired. A strategy that can work particularly well is to use the Cuthill–Mckee algorithm [7] to permute $\boldsymbol{A}$ into a banded matrix with a small bandwidth. Grouping all nonzero entries together around the main diagonal also effectively reduces the number of nonzero block matrices and hence the total MPO-rank. A block size $I_1 \times J_1$ can then be chosen such that the bandwidth is covered by a few blocks. Algorithm 4.1 can then be applied to the permuted matrix. Other permutations may reduce the maximal MPO-rank even further. We will discuss choosing the partitioning of $\boldsymbol{A}$ in more detail in section 4.3.

Algorithm 4.1 will construct an MPO with a uniform MPO-rank, which will exceed the upper bounds from Theorem 3.2 in almost all cases. One can use a rounding step to truncate the MPO-ranks without the loss of any accuracy after Algorithm 4.1 has finished. Alternatively, one can apply a rounding step on the intermediate result as soon as the MPO-rank reaches a certain threshold during the execution of the algorithm. The following example illustrates the necessity of the rounding step.

Example 4. Suppose we have three random 2×2 matrices $\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}$ such that

$$\boldsymbol{A} = \boldsymbol{A}^{(3)} \otimes \boldsymbol{A}^{(2)} \otimes \boldsymbol{A}^{(1)}.$$

By Theorem 3.1, the matrix $\boldsymbol{A}$ has a canonical unit-rank MPO representation where the first MPO-tensor is $\boldsymbol{A}^{(1)}$ reshaped into a $1 \times 2 \times 2 \times 1$ tensor. Choosing $I_1 = J_1 = I_2 = J_2 = I_3 = J_3 = 2$ and applying Algorithm 4.1 result in an MPO with a uniform rank of 16. Applying a rounding step truncates each of these ranks down to unity.

For a fixed block size I_1, J_1 one can determine a lower bound for the resulting MPO-rank in the following manner.

LEMMA 4.3. *Let z be the number of nonzero elements of $\boldsymbol{A}$ and I_1, J_1 the first dimensions of the partitioning of $\boldsymbol{A}$. If $z = I_1 \times J_1 \times R$, then a lower bound for the MPO-rank obtained by Algorithm* 4.1 *is R.*

Proof. Suppose that we found a permutation such that all z nonzero entries can be arranged into R block matrices of size $I_1 \times J_1$. It then trivially follows that R will be the MPO-rank since $z = I_1 \times J_1 \times R$. ∎

In practice, it will be difficult, or in some cases impossible, to find a permutation such that all nonzero entries are nicely aligned into $I_1 \times J_1$ block matrices. The R in Lemma 4.3 is therefore a lower bound.

4.3. Choosing a partition. In this subsection we discuss choosing a partition of the matrix $\boldsymbol{A}$ prior to applying Algorithm 4.1. We suppose, without loss of generality,

that the dimensions of $\boldsymbol{A}$ have prime factorizations $I = I_1\, I_2 \cdots I_d$ and $J = J_1\, J_2 \cdots J_d$ with an equal amount of d factors. The number of factors can always be made equal by appending ones. Ultimately, the goal is to obtain an MPO with "small" MPO-ranks. Although Algorithm 4.1 constructs an MPO with ranks that are likely to exceed the canonical values, these ranks can always be truncated through rounding. Theorem 3.2 can be used for choosing a partition that minimizes the upper bounds in the hope that the canonical values are even smaller. The key idea is that the upper bounds depend on the ordering of the prime factors, as the following small example illustrates.

Example 5. Suppose the factorizations are $I = 35 = 7 \times 5 \times 1$ and $J = 12 = 3 \times 2 \times 2$. If we choose the ordering of the partition as $I_1 = 1, J_1 = 3, I_2 = 5, J_2 = 2, I_3 = 7, J_3 = 2$ then the upper bounds are $R_2 \leq \min(3, 140) = 3$ and $R_3 \leq \min(30, 14) = 14$. Choosing the partition $I_1 = 5, J_1 = 2, I_2 = 7, J_2 = 3, I_3 = 1, J_3 = 2$ changes the upper bounds to $R_2 \leq 10$ and $R_3 \leq 2$.

In light of the randomized SVD algorithm that is developed in section 5, it will be necessary to order the prime factors in a descending sequence. In this way, the first MPO-tensor will have a sufficiently large dimension in order to compute the desired low-rank approximation. Observe that if we use the descending ordering $I_1 = 7, J_1 = 3, I_2 = 5, J_2 = 2, I_3 = 1, J_3 = 2$ in Example 5, then the upper bounds are $R_2 \leq \min(21, 20) = 20$ and $R_3 \leq \min(210, 2) = 2$. Large matrices can have dimensions with a large number of prime factors. Choosing a partition with a large number of MPO-tensors usually results in a high number of nonzero block matrices $\boldsymbol{X}$ and therefore also in a large MPO-rank. The problem with such a large MPO-rank can be that it becomes infeasible to do the rounding step due to lack of sufficient memory. In this case one needs to reduce the number of MPO-tensors until the obtained MPO-rank is small enough such that the rounding step can be performed. Whether rounding is practical is very case-specific. For example, if the original matrix is sparse, then $\boldsymbol{\mathcal{A}}^{(1)}$ will also be sparse and one can better do rounding through the removal of parallel columns (as described in [17]) instead of using truncated SVDs to preserve the sparsity. This way of choosing a partition will be demonstrated in more detail by means of a worked-out example in section 6.1.

5. Tensor network randomized SVD.

5.1. The rSVD algorithm. Given a matrix $\boldsymbol{A}$, which does not need to be sparse, the rSVD computes a low-rank factorization $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ where $\boldsymbol{U}, \boldsymbol{V}$ are orthogonal matrices and $\boldsymbol{S}$ is a diagonal and nonnegative matrix. The prototypical rSVD algorithm [16, p. 227] is given as pseudocode in Algorithm 5.1. When a rank-K approximation is desired, we compute a rank-$(K + s)$ approximation after which only the first K singular values and vectors are retained. The parameter s is called the oversampling parameter and is required to make sure the range of $\boldsymbol{A}$ is sufficiently approximated. The value of s depends on the dimensions of $\boldsymbol{A}$, its singular spectrum and is usually set to 5 or 10. For more details on oversampling we refer the reader to [16, p. 240]. It has been shown that a slow decay of the singular values of $\boldsymbol{A}$ results in a larger approximation error. The power iteration $(\boldsymbol{A}\boldsymbol{A}^T)^q$ tries to alleviate this problem by increasing the decay of the singular values while retaining the same left singular vectors of $\boldsymbol{A}$. Common values for q are 1 or 2. Note that the computation of $\boldsymbol{Y}$ is sensitive to round-off errors and additional orthogonalization steps are required.

ALGORITHM 5.1. *Prototypical rSVD algorithm* [16, p. 227]
Input: *matrix* $\boldsymbol{A} \in \mathbb{R}^{I \times J}$, *target number* K, *oversampling parameter* s, *and exponent* q.
Output: *approximate rank-*$(K+s)$ *factorization* $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$, *with* $\boldsymbol{U}, \boldsymbol{V}$ *orthogonal and* $\boldsymbol{S}$ *is diagonal and nonnegative.*
Generate an $J \times (K+s)$ *random matrix* $\boldsymbol{O}$.
$\boldsymbol{Y} \leftarrow (\boldsymbol{A}\boldsymbol{A}^T)^q \boldsymbol{A}\boldsymbol{O}$
$\boldsymbol{Q} \leftarrow$ *Orthogonal basis for the range of* $\boldsymbol{Y}$
$\boldsymbol{B} \leftarrow \boldsymbol{Q}^T \boldsymbol{A}$
Compute the SVD $\boldsymbol{B} = \boldsymbol{W}\boldsymbol{S}\boldsymbol{V}^T$.
$\boldsymbol{U} \leftarrow \boldsymbol{Q}\boldsymbol{W}$

For a large matrix $\boldsymbol{A}$, it quickly becomes infeasible to compute orthogonal bases for $\boldsymbol{Y}$ or to compute the SVD of $\boldsymbol{B}$. This is the main motivation for doing all steps of Algorithm 5.1 in MPO-form. We therefore assume that all matrices in Algorithm 5.1 can be represented by an MPO with relatively small MPO-ranks. In what follows, we derive how each of the steps of Algorithm 5.1 can be implemented in MPO-form, resulting in the TNrSVD as described in Algorithm 5.5. For a matrix $\boldsymbol{A} \in \mathbb{R}^{I_1 \cdots I_d \times J_1 \cdots J_d}$ with d MPO-tensors $\boldsymbol{\mathcal{A}}^{(i)} \in \mathbb{R}^{R_i \times I_i \times J_i \times R_{i+1}}$ $(i = 1, \ldots, d)$, TNrSVD computes a rank-K factorization that consists of d MPO-tensors $\boldsymbol{\mathcal{U}}^{(1)}, \ldots, \boldsymbol{\mathcal{U}}^{(d)}$ and $\boldsymbol{\mathcal{V}}^{(1)}, \ldots, \boldsymbol{\mathcal{V}}^{(d)}$ and the $K \times K$ diagonal and nonnegative $\boldsymbol{S}$ matrix.

5.2. Random matrix as a rank-1 MPO. The rSVD algorithm relies on multiplying the original matrix $\boldsymbol{A}$ with a random matrix $\boldsymbol{O}$. Fortunately, it is possible to directly construct a random matrix into MPO form.

LEMMA 5.1. *A particular random* $J_1 J_2 \cdots J_d \times K$ *matrix* $\boldsymbol{O}$ *with rank*$(\boldsymbol{O}) = K$ *and* $K \leq J_1$ *can be represented by a unit-rank MPO with the following random MPO-tensors:*

$$\boldsymbol{\mathcal{O}}^{(1)} \in \mathbb{R}^{1 \times J_1 \times K \times 1},$$
$$\boldsymbol{\mathcal{O}}^{(i)} \in \mathbb{R}^{1 \times J_i \times 1 \times 1} \qquad (2 \leq i \leq d).$$

Proof. All MPO-ranks being equal to one implies that Theorem 3.1 applies. The random matrix $\boldsymbol{O}$ is constructed from the Kronecker product of $d-1$ random column vectors $\boldsymbol{O}^{(i)} \in \mathbb{R}^{J_i}$, $(i = 2, \ldots, d)$ with the matrix $\boldsymbol{O}^{(1)} \in \mathbb{R}^{J_1 \times K}$. The Kronecker product has the property that

$$\operatorname{rank}(\boldsymbol{O}) = \operatorname{rank}(\boldsymbol{O}^{(d)}) \cdots \operatorname{rank}(\boldsymbol{O}^{(1)}).$$

The fact that $\boldsymbol{O}^{(1)}$ is a random matrix then ensures that $\operatorname{rank}(\boldsymbol{O}) = K$. ∎

Probabilistic error bounds for Algorithm 5.1 are typically performed for random Gaussian matrices $\boldsymbol{O}$ [16, p. 273]. Another type of test matrices $\boldsymbol{O}$ are subsampled random Fourier transform matrices [16, p. 277]. The random matrix in MPO form from Lemma 5.1 will not be Gaussian, as the multiplication of Gaussian random variables is not Gaussian. This prevents the straightforward determination of error bounds for the MPO-implementation of Algorithm 5.1 that we propose. In spite of the lack of any probabilistic bounds on the error, all numerical experiments that we performed demonstrate that the orthogonal basis that we obtain for the range of $\boldsymbol{A}$ can capture the action of $\boldsymbol{A}$ sufficiently. Once the matrix $\boldsymbol{A}$ has been converted into

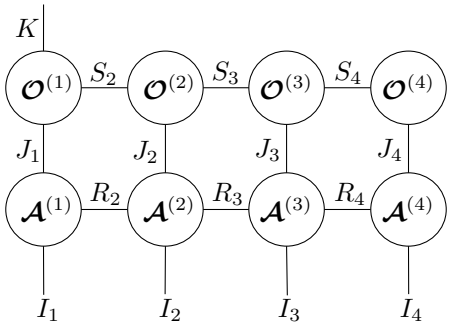

FIG. 5.1. *The matrix multiplication* $\boldsymbol{A}\,\boldsymbol{O}$ *as contraction of a tensor network.*

an MPO using Algorithm 4.1 and a random MPO has been constructed using Lemma 5.1, what remains are matrix multiplications and computing low-rank QR and SVD factorizations. We will now explain how these steps can be done efficiently using MPOs.

5.3. Matrix multiplication. Matrix multiplication is quite straightforward. Suppose the matrices $\boldsymbol{A} \in \mathbb{R}^{I_1 I_2 \cdots I_d \times J_1 J_2 \cdots J_d}, \boldsymbol{O} \in \mathbb{R}^{J_1 J_2 \cdots J_d \times K}$ have MPO representations of 4 tensors. This implies that the rows and columns of $\boldsymbol{A}$ are indexed by the multi-indices $[i_1 i_2 i_3 i_4]$ and $[j_1 j_2 j_3 j_4]$, respectively. The matrix multiplication $\boldsymbol{A}\,\boldsymbol{O}$ then corresponds with the summation of the column indices of $\boldsymbol{A}$

$$\boldsymbol{A}\,\boldsymbol{O} = \sum_{j_1, j_2, j_3, j_4} \boldsymbol{A}(:, [j_1 j_2 j_3 j_4])\, \boldsymbol{O}([j_1 j_2 j_3 j_4], :)$$

and is visualized as contractions of two MPOs meshed into one tensor network in Figure 5.1. All edges with a dimension of one are not shown but one has to keep in mind that all tensors are 4-way. The contraction $\sum_{j_i} \boldsymbol{\mathcal{A}}^{(i)}(:,:,j_i,:)\boldsymbol{\mathcal{O}}^{(i)}(:,j_i,:,:)$ for each of the four MPO-tensors results in a new MPO that represents the matrix multiplication $\boldsymbol{A}\,\boldsymbol{O}$. If $\boldsymbol{\mathcal{A}}^{(i)} \in \mathbb{R}^{R_i \times I_i \times J_i \times R_{i+1}}$ and $\boldsymbol{\mathcal{O}}^{(i)} \in \mathbb{R}^{S_i \times J_i \times 1 \times S_{i+1}}$, then the summation over the index J_i results in an MPO-tensor with dimensions $R_i S_i \times I_i \times 1 \times R_{i+1} S_{i+1}$ with a computational complexity of $O(R_i S_i I_i J_i R_{i+1} S_{i+1})$ flops. Corresponding MPO-ranks R_i, S_i and R_{i+1}, S_{i+1} are multiplied with one another, which necessitates a rounding step in order to reduce the dimensions of the resulting MPO-tensors. Note, however, that the random matrix $\boldsymbol{O}$ constructed via Lemma 5.1 has a unit-rank MPO, which implies that $S_1 = S_2 = \cdots = S_{d+1} = 1$ such that the MPO corresponding with the matrix $\boldsymbol{AO}$ will retain the MPO-ranks of $\boldsymbol{A}$.

5.4. Thin QR and economical SVD in MPO-form. An orthogonal basis for the range of $\boldsymbol{Y} \in \mathbb{R}^{I \times K}$ can be computed through a thin QR decomposition $\boldsymbol{Y} = \boldsymbol{Q}\,\boldsymbol{R}$, where $\boldsymbol{Q} \in \mathbb{R}^{I \times K}$ has orthogonal columns and $\boldsymbol{R} \in \mathbb{R}^{K \times K}$. The algorithm to compute a thin QR decomposition from a matrix in MPO-form is given in pseudocode in Algorithm 5.2 and its MATLAB implementation in the TNrSVD pacakge is qrTN.m. The thin QR is computed by an orthogonalization sweep from right-to-left, which means we start with the orthogonalization of $\boldsymbol{\mathcal{A}}^{(d)}$, and absorbing the R factor matrix into the preceding MPO-tensor. The main operations in the orthogonalization sweep are tensor reshaping and the matrix QR decomposition. The final computation is the orthogonalization of $\boldsymbol{\mathcal{A}}^{(1)}$, which is orthogonalized in a slightly different way such that the $K \times K$ $\boldsymbol{R}$ matrix is obtained. This last computation distinguishes Algorithm 5.2

ALGORITHM 5.2. *MPO-QR algorithm* [25, p. 2302]
***Input**: rank-K matrix $\boldsymbol{A} \in \mathbb{R}^{I \times K}$ in MPO-form with $\boldsymbol{\mathcal{A}}^{(1)} \in \mathbb{R}^{1 \times I_1 \times K \times R_2}$, $I_1 \geq K$.*
***Output**: d MPO-tensors of $\boldsymbol{Q} \in \mathbb{R}^{I \times K}$ with $\boldsymbol{\mathcal{Q}}^{(1)} \in \mathbb{R}^{1 \times I_1 \times K \times R_2}$, $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_K$, and $\boldsymbol{R} \in \mathbb{R}^{K \times K}$.*

```
for i=d:-1 : 2 do
  Reshape A^(i) into R_i × I_i R_{i+1} matrix A_i.
  A_i = R_i Q_i with R_i ∈ R^{R_i × R_i} and Q_i Q_i^T = I_{R_i}.
  Q^(i) ← reshape Q_i into R_i × I_i × 1 × R_{i+1} tensor.
  A^(i-1) ← A^(i-1) ×_4 R_i.
end for
Permute A^(1) into K × 1 × I_1 × R_2 tensor.
Reshape A^(1) into K × I_1 R_2 matrix A_1.
A_1 = R Q_1 with R ∈ R^{K × K} and Q_1 Q_1^T = I_K.
Reshape Q_1 into K × 1 × I_1 × R_2 tensor Q^(1).
Permute Q^(1) into 1 × I_1 × K × R_2 tensor.
```

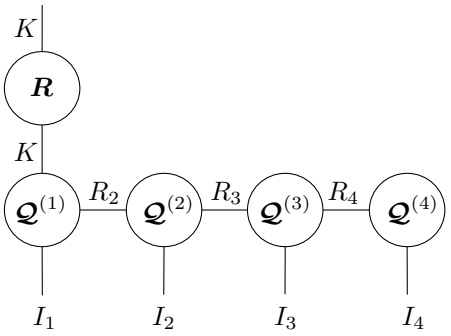

FIG. 5.2. *Thin QR decomposition $\boldsymbol{QR}$ as a tensor network with $\boldsymbol{Q} \in \mathbb{R}^{I_1 I_2 I_3 I_4 \times K}$ and $\boldsymbol{R} \in \mathbb{R}^{K \times K}$.*

from the standard orthogonalization algorithm. The computational cost of Algorithm 5.2 is dominated by the QR computation of the first MPO-tensor, as it normally has the largest dimensions. Using Householder transformations to compute this QR decomposition costs approximately $O(I_1 R_2 K^2)$ flops. The proof of the procedure can be found in [25, p. 2302]. The thin QR decomposition in MPO-form is illustrated for an MPO of four tensors in Figure 5.2. Again, all indices of dimension one are not shown.

The TNrSVD algorithm also requires an economical SVD computation of the $K \times J_1 \cdots J_d$ matrix $\boldsymbol{B} = \boldsymbol{W}\,\boldsymbol{S}\,\boldsymbol{V}^T$, where both $\boldsymbol{W}, \boldsymbol{S}$ are $K \times K$ matrices, $\boldsymbol{W}$ is orthogonal, and $\boldsymbol{S}$ is diagonal and nonnegative. The matrix $\boldsymbol{V}$ is stored in MPO-form. Only a slight modification of Algorithm 5.2 is required to obtain the desired matrices. Indeed, the only difference with Algorithm 5.2 is that now the SVD of $\boldsymbol{A}_1$ needs to be computed. From this SVD we obtain the desired $\boldsymbol{W}, \boldsymbol{S}$ matrices and can reshape and permute the right singular vectors into the desired $\boldsymbol{\mathcal{V}}^{(1)}$ MPO-tensor. Again, the overall computational cost will be dominated by this SVD step, which costs approximately $O(J_1 R_2 K^2)$ flops. The economical SVD of a matrix in MPO-form is given in pseudocode in Algorithm 5.3 and its MATLAB implementation in the TNrSVD package is svdTN.m. A graphical representation of the corresponding tensor network for a simple example of four MPO-tensors is depicted in Figure 5.3.

ALGORITHM 5.3. *MPO-SVD algorithm*
***Input**: d MPO-tensors of* $\boldsymbol{A} \in \mathbb{R}^{K\times J}$ *with* $\boldsymbol{\mathcal{A}}^{(1)} \in \mathbb{R}^{1\times K\times J_1\times R_2}$ *and* $J_1 \geq K$.
***Output**: d MPO-tensors of* $\boldsymbol{V} \in \mathbb{R}^{K\times J}$ *with* $\boldsymbol{\mathcal{V}}^{(1)} \in \mathbb{R}^{1\times K\times J_1\times R_2}$, $\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}_K$ *and* $\boldsymbol{W} \in \mathbb{R}^{K\times K}$, $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$, *and* $\boldsymbol{S} \in \mathbb{R}^{K\times K}$ *diagonal and nonnegative.*
for *i=d:*-1 : 2 ***do***
Reshape $\boldsymbol{\mathcal{A}}^{(i)}$ *into* $R_i \times I_i R_{i+1}$ *matrix* $\boldsymbol{A}_i$.
$\boldsymbol{A}_i = \boldsymbol{R}_i\boldsymbol{Q}_i$ *with* $\boldsymbol{R}_i \in \mathbb{R}^{R_i\times R_i}$ *and* $\boldsymbol{Q}_i\boldsymbol{Q}_i^T = \boldsymbol{I}_{R_i}$.
$\boldsymbol{\mathcal{V}}^{(i)} \leftarrow$ *reshape* $\boldsymbol{Q}_i$ *into* $R_i \times I_i \times 1 \times R_{i+1}$ *tensor.*
$\boldsymbol{\mathcal{A}}^{(i-1)} \leftarrow \boldsymbol{\mathcal{A}}^{(i-1)} \times_4 R_i$.
end for
Permute $\boldsymbol{\mathcal{A}}^{(1)}$ *into* $K \times 1 \times J_1 \times R_2$ *tensor.*
Reshape $\boldsymbol{\mathcal{A}}^{(1)}$ *into* $K \times J_1R_2$ *matrix* $\boldsymbol{A}_1$.
Compute SVD of $\boldsymbol{A}_1 = \boldsymbol{W}\boldsymbol{S}\boldsymbol{Q}_1^T$.
Reshape $\boldsymbol{Q}_1$ *into* $K \times 1 \times J_1 \times R_2$ *tensor* $\boldsymbol{\mathcal{V}}^{(1)}$.
Permute $\boldsymbol{\mathcal{V}}^{(1)}$ *into* $1 \times K \times J_1 \times R_2$ *tensor.*

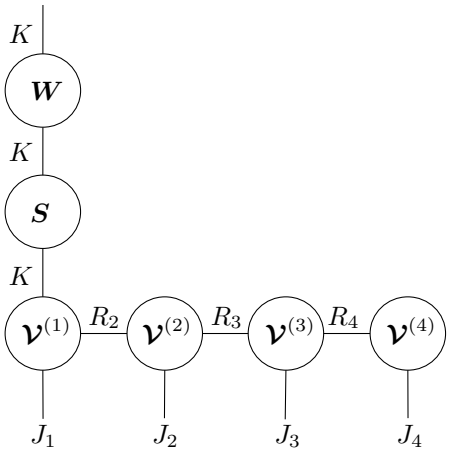

FIG. 5.3. *Economical SVD* $\boldsymbol{W}\boldsymbol{S}\boldsymbol{V}^T$ *as a tensor network with* $\boldsymbol{V} \in \mathbb{R}^{J_1J_2J_3J_4\times K}$ *and* $\boldsymbol{W}, \boldsymbol{S} \in \mathbb{R}^{K\times K}$.

Both the thin QR and economical SVD of the matrix in MPO-form are computed for each tensor of the MPO separately, which reduces the computational complexity significantly. Unlike the ALS-SVD and MALS-SVD, no iterative sweeping over the different MPO-tensors is required.

5.5. Randomized subspace iteration. The computation of the matrix $\boldsymbol{Y} = (\boldsymbol{A}\boldsymbol{A}^T)^q\,\boldsymbol{A}\boldsymbol{O}$ is vulnerable to round-off errors and an additional orthogonalization step is required between each application of $\boldsymbol{A}$ and $\boldsymbol{A}^T$. Indeed, the repeated multiplication with $\boldsymbol{A}$ and $\boldsymbol{A}^T$ results in the loss of information on singular values smaller than $\epsilon^{1/(2q+1)}||\boldsymbol{A}||$, where ϵ is the machine precision [16, p. 244]. Instead of computing $\boldsymbol{Y}$ and applying Algorithm 5.2, the randomized MPO-subspace iteration of Algorithm 5.4 is proposed. First, the random matrix $\boldsymbol{O}$ is multiplied onto $\boldsymbol{A}$, after which a rounding step is performed to reduce the MPO-ranks. Algorithm 5.2 is then applied to obtain an orthogonal basis $\boldsymbol{Q}$ for the range of $\boldsymbol{Y}$. One now proceeds with the multiplication $\boldsymbol{A}^T\boldsymbol{Q}$, after which another rounding step and orthogonalization through Algorithm 5.2 are performed. These steps are repeated until the desired number of multiplications

with $\boldsymbol{A}$ and $\boldsymbol{A}^T$ has been done. The SVD-based rounding and orthogonalization steps can actually be integrated into one another. Indeed, one can apply a left-to-right rounding sweep first, followed by the right-to-left sweep of Algorithm 5.2. This prevents performing the right-to-left sweep twice. Similarly, one can integrate the rounding step after the multiplication $\boldsymbol{A}^T\boldsymbol{Q}$ with the computation of the economical SVD $\boldsymbol{W}\boldsymbol{S}\boldsymbol{V}^T$. Note that the computational complexity and accuracy of the final result will depend on by how much the ranks are chosen to be truncated during the rounding step.

ALGORITHM 5.4. *Randomized MPO-subspace iteration*
Input: *exponent* q, $\boldsymbol{A} \in \mathbb{R}^{I\times J}$ *and random matrix* $\boldsymbol{O} \in \mathbb{R}^{J\times K}$ *in MPO-form.*
Output: *MPO-tensors of* $\boldsymbol{Q} \in \mathbb{R}^{I\times K}$ *with* $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_K$.

```
Y ← AO
Q ← Use Algorithm 5.2 on Y
for i=1:q do
  Y ← A^T Q
  Q ← Use Algorithm 5.2 on Y with rounding
  Y ← A Q
  Q ← Use Algorithm 5.2 on Y with rounding
end for
```

5.6. TNrSVD algorithm. All ingredients to perform each of the steps of Algorithm 5.1 in MPO-form are now available. The pseudocode of the TNrSVD algorithm is given as Algorithm 5.5 and its MATLAB implementation in the TNrSVD package is TNrSVD.m. First, a random matrix $\boldsymbol{O}$ is created in rank-1 MPO-form using Lemma 5.1. The MPO-subspace iteration algorithm is then used to generate an orthogonal basis $\boldsymbol{Q}$ in MPO-form for the range of $(\boldsymbol{A}\boldsymbol{A}^T)^q\,\boldsymbol{A}\,\boldsymbol{O}$. The matrix multiplication $\boldsymbol{B} = \boldsymbol{Q}^T\,\boldsymbol{A}$ is then performed as the contraction of the corresponding tensor networks as explained in subsection 5.3. Using Algorithm 5.3, we then compute the economical SVD of $\boldsymbol{B}$, which provides us with a $(K+s)\times(K+s)$ orthogonal matrix $\boldsymbol{W}$, diagonal $\boldsymbol{S}$, and orthogonal $\boldsymbol{V}$ in MPO-form. Since $\boldsymbol{W} \in \mathbb{R}^{(K+s)\times(K+s)}$, the multiplication $\boldsymbol{Q}\boldsymbol{W}$ in MPO-form is obtained from $\boldsymbol{\mathcal{Q}}^{(1)} \times_3 \boldsymbol{W}^T$, which results in the desired orthogonal matrix $\boldsymbol{U}$ in MPO-form.

ALGORITHM 5.5. *Tensor network randomized SVD*
Input: *matrix* $\boldsymbol{A} \in \mathbb{R}^{I\times J}$ *in MPO-form, target number* K*, oversampling parameter* s *and exponent* q.
Output: *approximate rank-*K *factorization* $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$*, where* $\boldsymbol{U}, \boldsymbol{V}$ *are orthogonal and in MPO-form,* $\boldsymbol{S}$ *is diagonal and nonnegative.*

```
Generate an J × (K + s) random matrix O in MPO-form using Lemma 5.1
Q ← Orthogonal basis for the range of (AA^T)^q AO using Algorithm 5.4
Compute B = Q^T A according to subsection 5.3
Compute the economical SVD B = WSV^T using Algorithm 5.3
Compute U = QW as Q^(1) ×_3 W^T
```

The computational complexities for each of the steps of the TNrSVD algorithm are listed in Table 5.1. These computational complexities are always dominated by the computation on $\boldsymbol{\mathcal{A}}^{(1)}$, as this tensor usually has the largest size. The R_2^2 factor

TABLE 5.1
Computational complexity of each step in Algorithm 5.5.

	Computational complexity
Algorithm 5.2	$O(I_1 R_2 (K+s)^2)$
Algorithm 5.3	$O(J_1 R_2 (K+s)^2)$
SVD-based rounding	$O(I_1 (K+s) R_2^2)$
$\boldsymbol{A}^T\boldsymbol{Q}$, $\boldsymbol{AQ}$	$O(R_2 I_1 J_1 (K+s) S_2)$
$\boldsymbol{QW}$	$O(J_1 R_2 (K+s)^2)$

in the SVD-based rounding algorithm is due to a different reshaping of $\boldsymbol{\mathcal{A}}^{(1)}$ being orthogonalized. The matrix multiplication $\boldsymbol{AQ}$ step in the subspace iterations are the most expensive step. Note that S_2 denotes the MPO-rank of $\boldsymbol{\mathcal{Q}}^{(1)}$. Comparing Table 5.1 with Table 1 in [19, p. 1005], which lists the computational complexity of the computations in the ALS-SVD and MALS-SVD algorithms, we see that the TNrSVD algorithm has lower computational complexities consistently. The three main computations of the ALS-SVD algorithm have complexities $O(KIR_A R^3 + KI^2 R_A^2 R^2), O(KI^2 R^3)$, and $O(IR_A R^3 + I^2 R_A^2 R^2)$, where R denotes the maximal MPO-rank for both the orthogonal $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices, and R_A denotes the maximal MPO-rank of $\boldsymbol{A}$.

Algorithm 5.5 requires the user to fix the values of K, s, and q in advance. In practice, one will run Algorithm 5.5 for incremental values of q until more power iterations are no longer useful. Alternatively, one can use the q-adaptive algorithm shown in Algorithm 5.6, which will use additional power iterations until some termination criterion is met. In [20], both the ALS-SVD and MALS-SVD algorithms run for a fixed number of iterations or until the relative residual $||\boldsymbol{A}^T\boldsymbol{U} - \boldsymbol{VS}||_F / ||\boldsymbol{S}||_F$ decreases below a given tolerance parameter ϵ. However, we have that

$$||\boldsymbol{A}^T\boldsymbol{U} - \boldsymbol{VS}||_F = ||\boldsymbol{A} - \boldsymbol{USV}^T||_F < \epsilon ||\boldsymbol{S}||_F,$$

ALGORITHM 5.6. *q-adaptive TNrSVD*
***Input**: matrix $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ in MPO-form, target number K, oversampling parameter s.*
***Output**: approximate rank-K factorization $\boldsymbol{USV}^T$, where $\boldsymbol{U}, \boldsymbol{V}$ are orthogonal and in MPO-form, $\boldsymbol{S}$ is diagonal and nonnegative.*
Generate an $J \times (K+s)$ random matrix $\boldsymbol{O}$ in MPO-form using Lemma 5.1
$\boldsymbol{Q} \leftarrow$ *Orthogonal basis for the range of $\boldsymbol{AO}$ using Algorithm* 5.2
Compute $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$ according to subsection 5.3
Compute the economical SVD $\boldsymbol{B} = \boldsymbol{WSV}^T$ using Algorithm 5.3
Compute $\boldsymbol{U} = \boldsymbol{QW}$ as $\boldsymbol{\mathcal{Q}}^{(1)} \times_3 \boldsymbol{W}^T$
while** Termination criterion not met **do
  $\boldsymbol{Q} \leftarrow$ *Orthogonal basis for the range of $(\boldsymbol{AA}^T)\boldsymbol{Q}$*
  Compute $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$ according to subsection 5.3
  Compute the economical SVD $\boldsymbol{B} = \boldsymbol{WSV}^T$ using Algorithm 5.3
  Compute $\boldsymbol{U} = \boldsymbol{QW}$ as $\boldsymbol{\mathcal{Q}}^{(1)} \times_3 \boldsymbol{W}^T$
end while

which implies that the user needs to know the relative approximation error of the best rank-K approximation before even using the (M)ALS-SVD algorithms. There is therefore a possibility that these algorithms never terminate, i.e., when ϵ is chosen

smaller than the optimal rank-K approximation error. An alternative stopping criterion that we propose is

$$\gamma := \max_{1 \le i \le K} \frac{|{\sigma_i^{(k)}}^2 - {\sigma_i^{(k-1)}}^2|}{{\sigma_1^{(k)}}^2}, \tag{5.1}$$

where $\sigma_i^{(k)}$ denotes the ith singular value computed at the kth iteration. Experiments in section 6 demonstrate that this heuristical criterion provides an estimate of the correct number of estimated digits of the obtained singular values. Note that the criterion (5.1) does not require any a priori knowledge of the best rank-K approximation error. Furthermore, as the estimates for the singular values and vectors are guaranteed to improve with increasing q values [23], it is less likely for Algorithm 5.6 to get stuck in an endless loop compared to the (M)ALS-SVD algorithms.

6. Numerical experiments. In this section we demonstrate the effectiveness of Algorithms 4.1 and 5.6, discussed in this article. Algorithms 4.1–5.6 were implemented in MATLAB and all experiments were done on a desktop computer with an 8-core Intel i7-6700 cpu @ 3.4 GHz and 64 GB RAM. These implementations can be freely downloaded from https://github.com/kbatseli/TNrSVD.

6.1. Matrix permutation prior to MPO conversion. Applying a permutation prior to the conversion can effectively reduce the maximal MPO-rank. Consider the 150102×150102 AMD-G2-circuit matrix from the SuiteSparse Matrix Collection [8] (formerly known as the University of Florida Sparse Matrix Collection), with a bandwidth of 93719 and sparsity pattern shown in Figure 6.1. The sparsity pattern after applying the Cuthill–Mckee algorithm is shown in Figure 6.2 and the bandwidth is reduced to 1962. We can factor 150102 as $2 \times 3 \times 3 \times 31 \times 269$, which sets the maximal number of tensors in the MPO to 6. The total number of nonzero entries is 726674, which makes an MPO representation of 6 tensors infeasible as for this case all $R_k = 726674$ and there is insufficient memory to store the MPO-tensors. Table 6.1 lists the number of MPO-cores d, the obtained MPO-rank for both the original matrix and after applying the Cuthill–Mckee algorithm

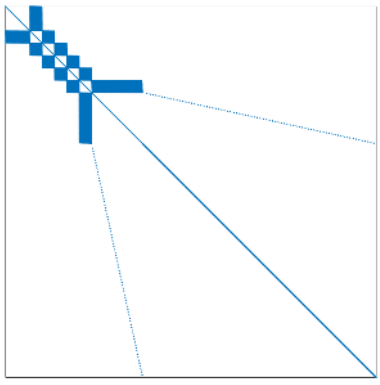

FIG. 6.1. *Original matrixsparsity pattern.*

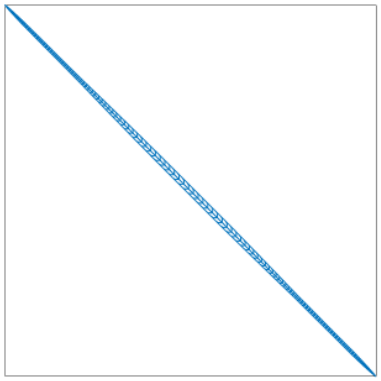

FIG. 6.2. *Sparsity pattern after the Cuthill–Mckee permutation.*

TABLE 6.1
Maximal MPO-ranks for varying block matrix sizes.

Matrix block size	d	Maximal rank		Runtime
		Original	Permuted	[seconds]
269×269	5	9352	4382	NA
538×538	4	3039	1753	5.47
807×807	4	1686	1018	4.67
1614×1614	3	665	347	3.92

and the runtime for applying Algorithm 4.1 on the permuted matrix. Applying the permutation effectively reduces the MPO-rank approximately by half so we only consider the permuted matrix. First, we order the prime factors in a descending fashion $269, 31, 3, 3, 2$, which would result in an MPO that consists of the following five tensors $\boldsymbol{\mathcal{A}}^{(1)} \in \mathbb{R}^{1\times 269\times 269\times 4382}, \boldsymbol{\mathcal{A}}^{(2)} \in \mathbb{R}^{4382\times 31\times 31\times 4382}, \boldsymbol{\mathcal{A}}^{(3)} \in \mathbb{R}^{4382\times 3\times 3\times 4382}, \boldsymbol{\mathcal{A}}^{(4)} \in \mathbb{R}^{4382\times 3\times 3\times 4381}$, and $\boldsymbol{\mathcal{A}}^{(5)} \in \mathbb{R}^{4382\times 2\times 2\times 4382}$. Due to the high MPO-rank, however, it is not possible to construct this MPO. We can now try to absorb the prime factor 2 into 269 and construct the corresponding MPO that consists of four tensors with $\boldsymbol{\mathcal{A}}^{(1)} \in \mathbb{R}^{1\times 538\times 538\times 1753}$. This takes about 5 seconds. As Table 6.1 shows, increasing the dimensions of $\boldsymbol{\mathcal{A}}^{(1)}$ by absorbing it with more prime factors further reduces the MPO-rank and runtimes. Note that if MATLAB supported sparse tensors by default, then it would be possible to use Algorithm 4.1 for both the original and permuted matrices as all MPO-tensors are sparse. SVD-based rounding on the $d = 3$ MPO with a tolerance of 10^{-10} reduces the MPO-rank from 347 down to 7 and takes about 61 seconds. Using the alternative parallel vector rounding from [17] truncates the MPO-rank also down to 7 in about 5 seconds.

6.2. Fast matrix-to-MPO conversion. In this experiment, Algorithm 4.1 is compared with the TT-SVD algorithm [27, p. 2135] and the TT-cross algorithm [30, p. 82], [31], both state-of-the-art methods for converting a matrix into an MPO. Both the TT-SVD and TT-cross implementations from the TT-Toolbox [26] were used. The TT-cross method was tested with the DMRG_CROSS function and was run for 10 sweeps and with an accuracy of 10^{-10}. We test the three algorithms on five matrices from the SuiteSparse Matrix Collection [8], which are listed in Table 6.2 together with their size and the decomposition of both the row and column dimensions. For example, the g7jac100 matrix has a dimension factorization of $329 \times 5 \times 3^2 \times 2$, which implies that the MPO consists of five tensors with $I_1 = 329, I_2 = 5, I_3 = 3, I_4 = 3$, and $I_5 = 2$. Table 6.3 lists the runtimes and relative errors when converting these matrices into the MPO format for the three considered methods. The relative errors are obtained by contracting the obtained MPO back into a matrix $\hat{\boldsymbol{A}}$ and computing $||\boldsymbol{A} - \hat{\boldsymbol{A}}||_F / ||\boldsymbol{A}||_F$. Algorithm 4.1 always results in an exact representation while the TT-SVD obtains a result that is accurate up to machine precision. The TT-cross

TABLE 6.2
Test matrices from the SuiteSparse Matrix Collection [8].

Matrix	Dimensions	Dimension factorization
Erdos972	5488×5488	$7^3 \times 2^4$
lhr10c	10672×10672	$29 \times 23 \times 2^4$
delaunay_n14	16384×16384	128×2^7
g7jac100	29610×29610	$329 \times 5 \times 3^2 \times 2$
venkat01	62424×62424	$289 \times 3^3 \times 2^3$

TABLE 6.3
Runtimes and relative errors for three different matrix-to-MPO methods.

Matrix	Runtime [seconds]			Relative error		
	Alg. 4.1	TT-SVD	TT-cross	Alg. 4.1	TT-SVD	TT-cross
Erdos972	0.649	12.25	14.92	0	8.11e-15	1.016
lhr10c	1.075	22.32	135.37	0	4.07e-15	0.926
delaunay_n14	0.171	1302.2	18.77	0	8.45e-15	1.141
g7jac100	0.903	422.34	716.31	0	4.67e-11	0.791
venkat01	1.025	NA	NA	0	NA	NA

method fails to find a sufficiently accurate MPO. Applying the TT-SVD and TT-cross methods on the venkat01 matrix was not possible due to insufficient memory.

6.3. Influence of q on accuracy. In this experiment, we revisit the matrices Erdos972 and lhr10c and illustrate how the value of q influences the obtained accuracy of the results. For both matrices a rank-100 approximation $\hat{\boldsymbol{U}}, \hat{\boldsymbol{V}}, \hat{\boldsymbol{S}}$ is computed with oversampling parameter $s = 100$. The Erdos972 matrix is converted into an MPO of five tensors with dimensions $I_1 = 343$ and $I_2 = \cdots = I_5 = 2$ and the lhr10c matrix into an MPO of three tensors with dimensions $I_1 = 232, I_2 = 23, I_3 = 2$. Given the size of these two matrices it is possible to compute the first 100 singular vectors and values using the svds command as a reference. Tables 6.4 and 6.5 list the relative error on the singular values $||\boldsymbol{S} - \hat{\boldsymbol{S}}||_F/||\boldsymbol{S}||_F$, the stopping criterion used in [19, p. 1006], the runtime of Algorithm 5.5, and the maximal and minimal number of correct digits of the computed singular values. The relative error on the singular values is seen to consistently decrease as q increases, indicating that the estimates improve for each additional subspace iteration. As expected, there is a large spread in number of

TABLE 6.4
*TNrSVD results on the Erdos*972 *matrix for various q values.*

q	0	1	2	3	4	5
$\frac{\|\|\boldsymbol{S}-\hat{\boldsymbol{S}}\|\|_F}{\|\|\boldsymbol{S}\|\|_F}$	0.1815	0.0569	0.0168	0.0063	0.0021	4.2e-4
$\frac{\|\|\boldsymbol{A}^T\hat{\boldsymbol{U}}-\hat{\boldsymbol{V}}\hat{\boldsymbol{S}}\|\|_F}{\|\|\hat{\boldsymbol{S}}\|\|_F}$	8.0e-15	5.9e-15	6.5e-15	6.7e-15	6.5e-15	7.0e-15
$\max_{1\le i\le K} \frac{\|\sigma_i(k)^2-\sigma_i(k-1)^2\|}{\sigma_1(k)^2}$	1	0.1388	0.0163	0.0038	0.0048	8.8e-4
Runtime [seconds]	5.9	17.6	29.2	40.9	52.8	64.9
Max digits correct	1	2	4	6	8	11
Min digits correct	1	1	1	2	2	3

TABLE 6.5
*TNrSVD results on the lhr*10*c matrix for various q values.*

q	0	1	2	3	4	5
$\frac{\|\|\boldsymbol{S}-\hat{\boldsymbol{S}}\|\|_F}{\|\|\boldsymbol{S}\|\|_F}$	0.4515	0.0643	0.0175	0.0104	0.0035	0.0021
$\frac{\|\|\boldsymbol{A}^T\hat{\boldsymbol{U}}-\hat{\boldsymbol{V}}\hat{\boldsymbol{S}}\|\|_F}{\|\|\hat{\boldsymbol{S}}\|\|_F}$	4.7e-15	6.2e-15	5.1e-15	4.9e-15	4.9e-15	4.8e-15
$\max_{1\le i\le K} \frac{\|\sigma_i(k)^2-\sigma_i(k-1)^2\|}{\sigma_1(k)^2}$	1	0.2438	0.0732	0.0316	0.0097	0.0016
Runtime [seconds]	5.5	18.1	28.7	40.9	52.7	64.7
Max digits correct	1	1	4	9	10	13
Min digits correct	0	1	1	1	2	2

correct digits. The largest singular values quickly converge to estimates that have over 10 correct digits while the smallest singular values only attain 2 or 3 correct digits. Each additional subspace iteration adds approximately 10 seconds to the total runtime for both matrices. The stopping criterion proposed in [19] is numerically zero and independent from the value of q for the Erdos972 matrix, which implies that this criterion is not suitable to be used in Algorithm 5.6. For the lhr10c matrix, both this stopping criterion and the relative errors have the same orders of magnitude. We see that $-\log_{10}(\gamma)$, where γ is the stopping criterion (5.1), corresponds approximately with the minimal number of correct digits.

6.4. Comparison with ALS-SVD and MALS-SVD. The tensor network-method described in [19] uses ALS and MALS methods to compute low-rank approximations of a given matrix in MPO-form. Three numerical experiments are considered in [19], of which two deal with finding a low-rank approximation to a given matrix. Here, we revisit these two experiments and compare the performance of our proposed TNrSVD algorithm with both the ALS-SVD and MALS-SVD methods. In order to do a fair comparison, the reported runtimes of the ALS-SVD and MALS-SVD algorithms are for a number of sweeps where either the desired accuracy was obtained or an additional sweep did not decrease the residual in the stopping criterion anymore.

6.4.1. Erdos972 and lhr10c matrices. We revisit the Erdos972 and lhr10c matrices and now apply the ALS-SVD algorithm. The MPO-ranks of both the orthogonal $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices are set to 600 and 200 for the Erdos972 and lhr10c matrix, respectively. It turns out that for these matrices neither $||\boldsymbol{S}-\hat{\boldsymbol{S}}||_F/||\boldsymbol{S}||_F$ nor $||\boldsymbol{A}^T\hat{\boldsymbol{U}}-\hat{\boldsymbol{V}}\hat{\boldsymbol{S}}||_F/||\hat{\boldsymbol{S}}||_F$ improves with additional ALS-sweeps. The ALS-SVD algorithm never terminates if the tolerance ϵ is chosen smaller than 6.8e-4 for the Erdos972 matrix and smaller than 2.2e-5 for lhr10c matrix. Table 6.6 lists the aforementioned residuals, together with the total runtime and maximal and minimal number of correct digits for the Erdos972 and lhr10c matrix. As expected, the number of correctly estimated digits of the singular values is more uniform compared to the TNrSVD method. Both the TNrSVD and ALS-SVD algorithms are able to find a rank-100 approximation of the Erdos972 matrix where at least 3 digits are estimated correctly in about 1 minute. Note that the dominant singular values are estimated correctly by the TNrSVD algorithm up to 11 digits. For the lhr10c matrix, the ALS-SVD algorithm requires almost 3 minutes and obtains estimates of the singular values that are accurate up to 6 digits. The TNrSVD algorithm obtains estimates for the singular values that are correct up to 13 digits in about 1 minute.

TABLE 6.6
*ALS-SVD results on the Erdos*972 *and lhr*10c *matrix.*

	Erdos972 ($R = 600$)	lhr10c ($R = 200$)
$\frac{\|\|\boldsymbol{S}-\hat{\boldsymbol{S}}\|\|_F}{\|\|\boldsymbol{S}\|\|_F}$	6.8e-4	2.2e-5
$\frac{\|\|\boldsymbol{A}^T\hat{\boldsymbol{U}}-\hat{\boldsymbol{V}}\hat{\boldsymbol{S}}\|\|_F}{\|\|\hat{\boldsymbol{S}}\|\|_F}$	0.0257	0.0054
Runtime [seconds]	65.7	173.4
Max digits correct	3	6
Min digits correct	3	4

6.4.2. Hilbert matrix. The first matrix that is considered in [19] is a rectangular submatrix of the Hilbert matrix. The Hilbert matrix $\boldsymbol{H} \in \mathbb{R}^{2^N \times 2^N}$ is a symmetric matrix with entries

$$\boldsymbol{H}(i,j) = (i+j-1)^{-1}, i,j = 1,2,\ldots,2^N.$$

The submatrix $\boldsymbol{A} := \boldsymbol{H}(:, 1:2^{N-1})$ is considered with $10 \leq N \leq 50$. Following [19], the corresponding MPO is constructed using the FUNCRS2 function of the TT-Toolbox [26], which applies a TT-cross approximation method using a functional description of the matrix entries, and consists of N MPO-tensors. The obtained MPO approximates the Hilbert matrix with a relative error of 10^{-11}. Maximal MPO-ranks for all values of N were bounded between 18 and 24. A tolerance of 10^{-8} was chosen for the residual $||\boldsymbol{A}^T\hat{\boldsymbol{U}} - \hat{\boldsymbol{V}}\hat{\boldsymbol{S}}||_F / ||\hat{\boldsymbol{S}}||_F$ when computing rank-16 approximations with both the ALS and MALS algorithms. As no information on the real singular values is available, we therefore use the singular values obtained from the ALS-SVD method as reference values. We use the q-adaptive TNrSVD method, Algorithm 5.6, to compute a rank-16 approximation such that the obtained singular values are accurate up to at least 8 digits. The tolerance for the stopping criterion is set to 10^{-3} and the rounding tolerance is set to 10^{-9}, which ensured that at least 8 digits of each of the obtained singular values are identical with the reference singular values. In order to be able to apply Algorithm 5.1, we first need to make sure that the MPO-tensor $\boldsymbol{\mathcal{Y}}^{(1)}$ of the matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{O}$ has dimensions $1 \times I_1 \times (K+s) \times R_2$ with $I_1 \geq (K+s)$. We set the oversampling parameter s equal to K. By contracting the first 5 MPO-tensors $\boldsymbol{\mathcal{A}}^{(1)}, \boldsymbol{\mathcal{A}}^{(2)}, \boldsymbol{\mathcal{A}}^{(3)}, \boldsymbol{\mathcal{A}}^{(4)}, \boldsymbol{\mathcal{A}}^{(5)}$ into a tensor with dimensions $1 \times 32 \times 32 \times R_6$, we obtain a new MPO of $N-5+1$ tensors that satisfies the $I_1 \geq K$ condition. Figure 6.3 shows the runtimes of the ALS-SVD, MALS-SVD, and TNrSVD methods as a function of N. The MALS-SVD method solves larger optimization problems than the ALS-SVD method at each iteration and is therefore considerably slower. The TNrSVD method is up to 5 times faster than the ALS-SVD method and 11 times faster than the MALS-SVD method for this particular example. Using a standard matrix implementation of Algorithm 5.1 we could compute low-rank approximations only for the $N = 10$ and $N = 15$ cases, with respective runtimes of 0.04 and 9.79 seconds. From this we

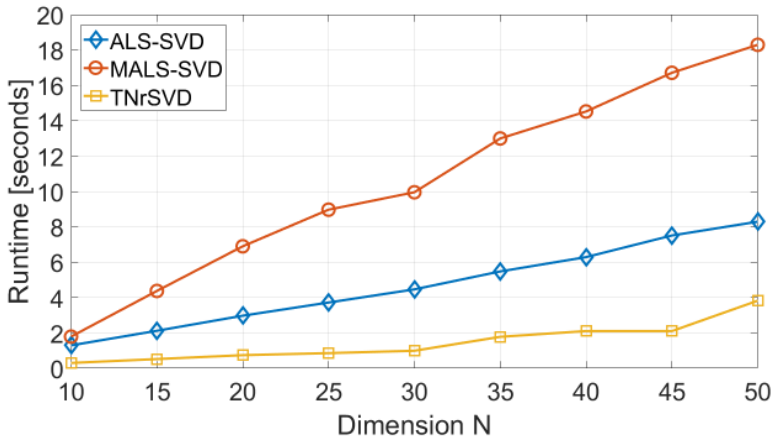

FIG. 6.3. *Runtimes for computing rank-16 approximations of* $2^N \times 2^{N-1}$ *Hilbert matrices for* $10 \leq N \leq 50$.

can conclude that all three tensor-based methods outperform the standard matrix implementation of Algorithm 5.1 when $N \geq 15$ for this particular example.

6.4.3. Random matrix with prescribed singular values. The second and final matrix that is considered in [19] is one with 50 prescribed singular values 0.5^k, $(k = 0, \ldots, 49)$ and random left and right singular vectors $\boldsymbol{U} \in \mathbb{R}^{2^N \times 50}, \boldsymbol{V} \in \mathbb{R}^{2^N \times 50}$. As with the Hilbert matrices, N ranges from 10–50 and equals the number of tensors in the MPO. The maximal MPO-rank of all constructed MPOs was 25. The orthogonal $\boldsymbol{U}, \boldsymbol{V}$ matrices were generated in MPO-form using the TT_RAND function of the TT-Toolbox. The MPO-representation of the matrix was then obtained by computing $\boldsymbol{USV}^T$ in MPO-form. A tolerance of 10^{-6} was set for the computation of rank-50 approximations with both the ALS and MALS algorithms, all singular values obtained from either the ALS or MALS method are accurate up to 3 digits. Additional sweeps of the ALS and MALS algorithms did not improve the accuracy any further and therefore only one sweep was used for each experiment. The q-adaptive TNrSVD algorithm, Algorithm 5.6, was run with a tolerance of 10^{-1} and the rounding tolerances were set to $10^{-5}, 10^{-6}, 10^{-8}, 10^{-8}, 10^{-9}, 10^{-10}, 10^{-11}, 10^{-12}, 10^{-13}$ for $N = 10, 15, 20, \ldots, 50$, respectively. This ensured that the result of the TNrSVD method had a relative error $||\boldsymbol{S} - \hat{\boldsymbol{S}}||_F / ||\boldsymbol{S}||_F$ on the estimated 50 dominant singular values below 10^{-6}, implying that the obtained singular values were accurate up to 6 digits. Computing a rank-50 approximation implies that $K = 100$ and the first 7 MPO tensors need to be contracted prior to running the TNrSVD algorithm. These contractions result in a new MPO where the first tensor has dimensions $1 \times 128 \times 128 \times R_8$ such that $I_1 = 128 \geq K = 100$ is satisfied. Figure 6.4 shows the runtimes of the ALS, MALS, and TNrSVD methods as a function of N. Just like with the Hilbert matrices, the MALS algorithm takes considerately longer to finish one sweep. The TNrSVD algorithm is up to 4 times faster than ALS and 13 times faster than MALS for this particular example. Using a standard matrix implementation of Algorithm 5.1 we could compute low-rank approximations only for the $N = 10$ and $N = 15$ cases, with respective runtimes of 0.04 and 7.59 seconds. For $N = 15$, the runtimes for the ALS, MALS, and TNrSVD methods were 6.4, 16.9, and 2.9 seconds, respectively.

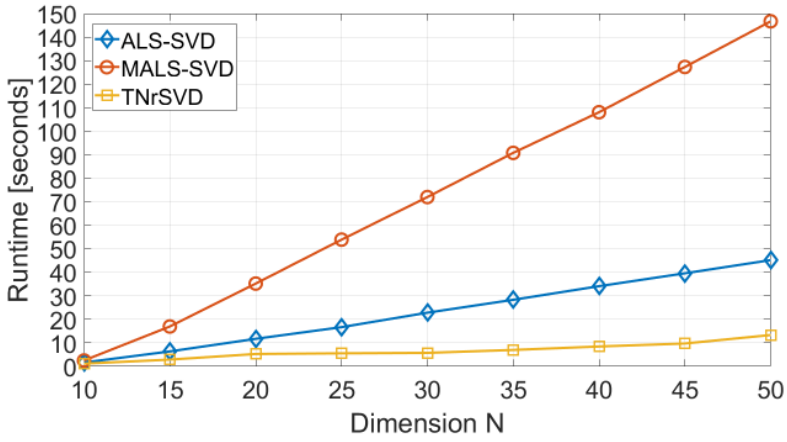

FIG. 6.4. *Runtimes for computing* rank-50 *decompositions of* $2^N \times 2^N$ *random matrices with prescribed singular values for* $10 \leq N \leq 50$.

7. Conclusion. We have proposed a new algorithm to convert a sparse matrix into an MPO form and a new randomized algorithm to compute a low-rank approximation of a matrix in MPO form. Our matrix to MPO conversion algorithm is able to generate MPO representations of a given matrix with machine precision accuracy up to 509 times faster than the standard TT-SVD algorithm. Compared with the state-of-the-art ALS-SVD and MALS-SVD algorithms, our TNrSVD is able to find low-rank approximations with the same accuracy up to 6 and 13 times faster, respectively. Future work includes the investigation of finding permutations, other than the Cuthill–Mckee permutation, that can reduce the MPO-rank of a given matrix.

REFERENCES

[1] K. Batselier, Z. Chen, and N. Wong, *Tensor network alternating linear scheme for MIMO Volterra system identification*, Automatica, 84 (2017), pp. 26–35.

[2] E. Beltrami, *Sulle Funzioni Bilineari*, Giornale di Mathematiche, 11 (1873), pp. 98–106.

[3] A. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[4] Z. Chen, K. Batselier, J. A. K. Suykens, and N. Wong, *Parallelized tensor train learning of polynomial classifiers*, IEEE Trans. Neural Netw. Learn. Syst., (2017), pp. 1–12, https://dx.doi.org/10.1109/TNNLS.2017.2771264.

[5] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, *Tensor networks for dimensionality reduction and large-scale optimization: Part* 1 *low-rank tensor decompositions*, Found. Trends Mach. Learn., 9 (2016), pp. 249–429.

[6] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, and D. P. Mandic, *Tensor networks for dimensionality reduction and large-scale optimization: Part* 2 *applications and future perspectives*, Found. Trends Mach. Learn., 9 (2017), pp. 431–673.

[7] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 1969 24th National Conference, ACM '69, ACM, New York, NY, 1969, pp. 157–172.

[8] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25.

[9] J. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[10] S. Dolgov, B. Khoromskij, I. Oseledets, and D. Savostyanov, *Computation of extreme eigenvalues in higher dimensions using block tensor train format*, Comput. Phys. Commun., 185 (2014), pp. 1207–1216.

[11] S. Dolgov and D. Savostyanov, *Alternating minimal energy methods for linear systems in higher dimensions*, SIAM J. Sci. Comput., 36 (2014), pp. A2248–A2271.

[12] L. Eldén, *Matrix Methods in Data Mining and Pattern Recognition*, SIAM, Philadelphia, 2007.

[13] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J. Soc. Ind. Appl. Math. Ser. B Numer. Anal., 2 (1965), pp. 205–224.

[14] G. H. Golub and C. Reinsch, *Singular value decomposition and least squares solutions*, Numer. Math., 14 (1970), pp. 403–420.

[15] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[16] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[17] C. Hubig, I. P. McCulloch, and U. Schollwöck, *Generic construction of efficient matrix product operators*, Phys. Rev. B, 95 (2017), p. 035129.

[18] T. Kolda and B. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.

[19] N. Lee and A. Cichocki, *Estimating a few extreme singular values and vectors for large-scale matrices in tensor train format*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 994–1014.

[20] N. Lee and A. Cichocki, *Regularized computation of approximate pseudoinverse of large matrices using low-rank tensor train decompositions*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 598–623.

[21] E. Liberty, F. Woolfe, P. G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci., 104 (2007), pp. 20167–20172.

[22] M. Moonen and B. De Moor, *SVD and Signal Processing,* III: *Algorithms, Architectures and Applications*, Elsevier Science, Amsterdam, The Netherlands, 1995.
[23] C. Musco and C. Musco, *Randomized block Krylov methods for stronger and faster approximate singular value decomposition*, in Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15, MIT Press, Cambridge, MA, 2015, pp. 1396–1404.
[24] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, *Tensorizing neural networks*, in Proceedings of the 28th International Conference on Neural Information Processing Systems, NIPS'15, Cambridge, MA, USA, 2015, MIT Press, pp. 442–450.
[25] I. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317.
[26] I. Oseledets, S. Dolgov, et al., *MATLAB TT-Toolbox Version* 2.3, available online, 2014.
[27] I. V. Oseledets, *Approximation of* $2^d \times 2^d$ *matrices using tensor decomposition*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2130–2145.
[28] I. V. Oseledets, *DMRG approach to fast linear algebra in the TT–format*, Comput. Methods Appl. Math., 11 (2011), pp. 382–393.
[29] I. V. Oseledets and S. V. Dolgov, *Solution of linear systems and matrix inversion in the tt-format*, SIAM J. Sci. Comput., 34 (2012), pp. A2718–A2739.
[30] I. V. Oseledets and E. Tyrtyshnikov, *TT-cross approximation for multidimensional arrays*, Linear Algebra Appl., 422 (2010), pp. 70–88.
[31] D. Savostyanov and I. Oseledets, *Fast adaptive interpolation of multi-dimensional arrays in tensor train format*, in The 2011 International Workshop on Multidimensional (ND) Systems, IEEE, 2011, pp. 1–8.
[32] U. Schollwöck, *The density-matrix renormalization group in the age of matrix product states*, Ann. Physics, 326 (2011), pp. 96–192.
[33] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 335–366.
[34] W. Yu, Y. Gu, J. Li, S. Liu, and Y. Li, *Single-pass PCA of Large High-Dimensional Data*, CoRR, abs/1704.07669, 2017.
[35] W. Yu, Y. Gu, and Y. Li, *Efficient Randomized Algorithms for the Fixed-Precision Low-Rank Matrix Approximation*, CoRR, abs/1606.09402, 2018.