FASTER TENSOR TRAIN DECOMPOSITION FOR SPARSE DATA

LINGJIE LI†, WENJIAN YU†, AND KIM BATSELIER‡

Abstract. In recent years, the application of tensors has become more widespread in fields that involve data analytics and numerical computation. Due to the explosive growth of data, low-rank tensor decompositions have become a powerful tool to harness the notorious curse of dimensionality. The main forms of tensor decomposition include CP decomposition, Tucker decomposition, tensor train (TT) decomposition, etc. Each of the existing TT decomposition algorithms, including the TT-SVD and randomized TT-SVD, is successful in the field, but neither can both accurately and efficiently decompose large-scale sparse tensors. Based on previous research, this paper proposes a new quasi-best fast TT decomposition algorithm for large-scale sparse tensors with proven correctness and for which the upper bound of its complexity derived. In numerical experiments, we verify that the proposed algorithm can decompose sparse tensors faster than the TT-SVD, and have more speed, precision and versatility than randomized TT-SVD, and it can be used to decomposes arbitrary high-dimensional tensor without losing efficiency when the number of non-zero elements is limited. The new algorithm implements a large-scale sparse matrix TT decomposition that was previously unachievable, enabling tensor decomposition based algorithms to be applied in larger-scale scenarios.

Key words. tensor train decomposition, sparse data, TT-rounding, parallel-vector rounding

AMS subject classifications. 15A23, 15A69, 65F99

1. Introduction. In the fields of physics, data analytics, scientific computing, digital circuit design, machine learning, etc., data are often organized into a matrix or tensor so that various sophisticated data processing techniques can be applied. One example of such a technique is the low-rank matrix decomposition [20, 21]. It is often implemented through the well-known singular value decomposition (SVD).

\[ A = U \Sigma V^T, \]

where \( A \in \mathbb{R}^{n \times m}, \Sigma \in \mathbb{R}^{n \times n} \), \( \Sigma \) is a diagonal matrix whose diagonal elements (a.k.a. singular values) \( \sigma_i, (1 \leq i \leq \min(m, n)) \) are non-negative and non-ascending.

In recent years, tensors, as a high-dimensional extension of matrices, have also been applied as a powerful and universal tool. In order to overcome the curse of dimensionality (the data size of a tensor increases exponentially with the increase of the dimensionality of the tensor), people have extended the notion of a low-rank matrix decomposition to tensors, proposing tensor decompositions such as the CP decomposition [7], the Tucker decomposition [18] and the tensor train (TT) decomposition [16]. Among them, the TT decomposition transforms the storage complexity of an \( n^d \) tensor into \( O(dnr^2) \), where \( r \) is the maximal TT rank, effectively removing the exponential dependence on \( d \). The TT decomposition is advantageous for processing large data sets and has been applied to problems like linear equation solution [17], electronic design automation (EDA) [13, 22], system identification [1, 2], large-scale matrix processing [3, 15] and image/video inpainting [11, 19].

To realize the TT decomposition, the TT-SVD algorithm [16] was proposed. It involves a sequence of SVD computations on reshaped matrices. For a large-scale

\*This work is supported by the NSFC under grant 61872206.
†BNRist, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (li-lj18@mails.tsinghua.edu.cn, yu-wj@tsinghua.edu.cn).
‡The Delft Center for Systems and Control, Delft University of Technology, Delft, Netherlands (k.batselier@tudelft.nl).
sparse tensor, the TT-SVD consumes excessive computing time and memory usage. Recently, a randomized TT-SVD algorithm [8] was proposed, which incorporates the randomized SVD algorithm [6] into the TT-SVD algorithm so as to reduce the runtime for converting a sparse tensor. However, due to the inaccuracy of the randomized SVD, the randomized TT-SVD algorithm usually results in the TT with exaggerated TT ranks or insufficient accuracy. This largely limits its application.

In this work, we propose a fast and effective TT decomposition algorithm specifically for large sparse data tensors. It includes the steps of constructing an exact TT with nonzero $p$-subvectors, parallel-vector rounding and TT-rounding. The new algorithm, called FastTT, produces the same compact TT representation as the TT-SVD algorithm [16], but exhibits a significant runtime advantage for large sparse data. We have also extended the algorithm to convert a matrix into the “matrix in TT-format”, also known as a matrix product operator (MPO). In addition, dynamic approaches are proposed to choose the parameters $p$ and $\delta_k$ in the FastTT algorithm. Experiments are carried out on sparse data in problems of image/video inpainting, linear equation solution, and data analysis. The results show that the proposed algorithm is several times to several hundreds times faster than the TT-SVD algorithm without loss of accuracy or an increase of the TT ranks. The speedup ratios are up to 9.6X for the image/video inpainting, 240X for the linear equation and 35X for the sparse data processing, respectively. The experimental results also reveal the effectiveness of the proposed dynamic approaches for choosing the parameters in the FastTT algorithm, and the advantages of FastTT over the randomized TT-SVD algorithm [8].

2. Notations and Preliminaries. In this article we use boldface capital calligraphic letters (e.g. $\mathcal{A}$) to denote tensors, boldface capital letters (e.g. $A$) to denote matrices, boldface letters (e.g. $a$) to denote vectors, and roman (e.g. $a$) or Greek (e.g. $\alpha$) letters to denote scalars.

2.1. Tensor. Tensors are a high-dimensional generalization of matrices and vectors. A one-dimensional array $a \in \mathbb{R}^n$ is called a vector, and a two-dimensional array $A \in \mathbb{R}^{n_1 \times n_2}$ is called a matrix. When the dimensionality is extended to $d \geq 3$, the $d$-dimensional array $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ is called a $d$-way tensor. The positive integer $d$ is defined as the order of the tensor. $(n_1, n_2, \cdots, n_d)$ are the dimensions of the tensor, where each $n_k$ is the dimension of a particular mode. Vectors and matrices can be considered as 1-way and 2-way tensors, respectively.

2.2. Basic Tensor Arithmetic.

Definition 2.1. Vectorization [22]. If we reorder the entries of $A \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ into a vector $b \in \mathbb{R}^{\prod_{k=1}^d n_k}$, where

$$a_{i_1, i_2, \cdots, i_d} = \frac{b}{\sum_{k=1}^{d}[(i_k-1)\prod_{l=k+1}^{d} n_l]+i_d},$$

then the vector $b$ is called the vectorization of the tensor $A$, represented as vec($A$).

Definition 2.2. Reshaping [22]. Like vectorization, if we reorder the entries of $A$ into another tensor $B$ satisfying vec($A$) = vec($B$), then the tensor $B$ is called the reshaping of $A$, represented as reshape($A, Dims$), where Dims denotes the dimensions of $B$. In fact, vectorization is a special kind of reshaping.

Definition 2.3. Splitting [16, unfolding]. Splitting is also a kind of reshaping. If we reshape $A \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ into a matrix $B \in \mathbb{R}^{n_1 \times n_2}$, then $B$ is called the $k$-splitting of $A$, represented as split$_k(A)$. 

Definition 2.4. **Contraction** [2]. Contraction is the tensor generalization of matrix product. For two tensors $\mathbf{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ and $\mathbf{B} \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_d}$ satisfying $n_k = m_k$, their $(k_1, k_2)$-contraction $\mathbf{C} = \mathbf{A} \circ_{k_1}^{k_2} \mathbf{B}$ is defined as

$$
c_{i_1 \cdots i_{k_1} j_{k_1} \cdots j_{k_2} 1} j_{k_2+1} \cdots j_{d} k_{i_1+1} \cdots i_{d_1} = \sum_{l=1}^{n_{k_1}} a_{i_1 \cdots i_{k_1} - l(i_{k_1}+1) \cdots i_{d_1}} b_{j_{k_1} \cdots j_{k_2} - l(j_{k_2+1}+1) \cdots j_{d_2}},
$$

where $\mathbf{C} \in \mathbb{R}^{n_1 \times \cdots \times n_{k_1-1} \times m_{k_1} \times \cdots \times m_{k_2-1} \times m_{k_2+1} \times \cdots \times m_{d_2} \times n_{k_1+1} \times \cdots \times n_{d_1}}$. If $k_1$ and $k_2$ are not specified, $\mathbf{A} \circ \mathbf{B}$ means the $(d_1, 1)$-contraction of $\mathbf{A}$ and $\mathbf{B}$.

Definition 2.5. **Tensor-matrix product** [22]. The $k$-product of a tensor $\mathbf{A}$ and a matrix $\mathbf{B}$ can be defined as tensor contraction if the matrix is treated as a 2-way tensor.

$$\mathbf{A} \times_k \mathbf{B} = \mathbf{A} \circ_k \mathbf{B}.$$

Definition 2.6. **Rank-1 tensor** [22]. A rank-1 $d$-way tensor can be written as the outer product

$$\mathbf{A} = u^{(1)} \circ u^{(2)} \circ \cdots \circ u^{(d)},$$

of $d$ column vectors $u^{(1)} \in \mathbb{R}^{n_1}, \ldots, u^{(d)} \in \mathbb{R}^{n_d}$. The entries of $\mathbf{A}$ can be computed as $a_{i_1i_2\cdots i_d} = u^{(1)}_{i_1} u^{(2)}_{i_2} \cdots u^{(d)}_{i_d}$.

2.3. Tensor Train Decomposition. A tensor train decomposition [16], shown in Figure 1(a), represents a $d$-way tensor $\mathbf{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ with two 2-way tensors and $(d-2)$ 3-way tensors:

$$\mathbf{A} = \mathcal{G}^{(1)} \circ \mathcal{G}^{(2)} \circ \cdots \circ \mathcal{G}^{(d)},$$

where $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ is the $k$-th core tensor. Per definition, $r_0 = r_d = 1$ such that $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(d)}$ are actually matrices. The dimensions $r_0, r_1, \ldots, r_d$ of the auxiliary indices are called the tensor-train (TT) ranks. When all the TT ranks have the same value, then we can just call it the **TT rank**.

![Graphical illustrations of the tensor train (TT) decomposition](image)

(a) A 3-way tensor and its TT decomposition

(b) TT diagram of a 3-way tensor

**Fig. 1.** Graphical illustrations of the tensor train (TT) decomposition, where a 3-way tensor $\mathbf{A}$ is decomposed into two 2-way tensors $\mathcal{G}^{(1)}, \mathcal{G}^{(3)}$ and a 3-way tensor $\mathcal{G}^{(2)}$.

Figure 1(b) shows a very convenient graphical representation [2] of a tensor train. In this diagram, each circle represents a tensor where each “leg” attached to it denotes a particular mode of the tensor. The connected line between two circles represents the contraction of two tensors. The dimension is labeled besides each “leg”. Figure 1(b) also illustrates a simple tensor network, which is a collection of tensors that are interconnected through contractions. By fixing the second index of $\mathcal{G}^{(k)}$ to $i_k$, we
obtain a matrix $G^{(k)}_{i_k}$ (actually a vector if $k = 1$ or $k = d$). Then the entries of $A$ can be computed as

$$a_{i_1,i_2,...,i_d} = G^{(1)}_{i_1} G^{(2)}_{i_2} \cdots G^{(d)}_{i_d}.$$

The tensor train decomposition can be computed with the TT-SVD algorithm [16], which consists of doing $d-1$ consecutive reshapings and matrix SVD computations. It is described as Algorithm 2.1. The expression rank$_{\delta}(C)$ denotes the number of remaining singular values after the $\delta$-truncated SVD. An advantage of TT-SVD is that a quasi-optimal approximation can be obtained with a given error bound and an automatic rank determination.

**Algorithm 2.1 TT-SVD [16, p. 2301]**

**Input:** a tensor $A \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, desired accuracy tolerance $\varepsilon$.

**Output:** Core tensors $G^{(1)}, \ldots, G^{(d)}$ of the TT-approximation $B$ to $A$ with TT ranks $r_k (k = 0, 1, \cdots, d)$ satisfying

$$\|A - B\|_F \leq \varepsilon \|A\|_F.$$

1: Compute truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}} \|A\|_F$.
2: $C := A, r_0 := 1$.
3: for $k = 1$ to $d - 1$ do
4: $C := \text{reshape}(C, [r_k - 1, n_k, \prod_{j=k+1}^d n_j])$.
5: Compute $\delta$-truncated SVD: $C = U \Sigma V^T + E, \|E\|_F \leq \delta, \ r_k := \text{rank}_\delta(C)$.
6: $G^{(k)} := \text{reshape}(U, [r_k - 1, n_k, r_k])$.
7: $C := \Sigma V^T$.
8: end for
9: $G^{(d)} := C$.
10: Return tensor $B$ in TT-format with cores $G^{(1)}, \ldots, G^{(d)}$.

We define the FLOP count of the TT-SVD algorithm as $f_{\text{TT-SVD}}$. Then

$$f_{\text{TT-SVD}} \approx \sum_{i=1}^{d-1} f_{\text{SVD}} \left( r_i - 1, n_i, \prod_{j=i+1}^d n_j \right),$$

where $f_{\text{SVD}}(m, n) = C_{\text{SVD}} m n \min(m, n)$ is the FLOP count of performing the economic SVD for an $m \times n$ dense matrix.

**3. Faster tensor train decomposition of sparse tensor.** The TT-SVD algorithm does not take advantage of the possible sparsity of data since the $\delta$-truncated SVD is used. In this section, we propose a new algorithm for computing the TT decomposition of a sparse tensor whereby the sparsity is explicitly exploited. The key idea is to rearrange the data in such a way that the desired TT decomposition can be written down explicitly, followed by a parallel-vector rounding step.

**3.1. Constructing TT with nonzero $p$-subvectors.** For a tensor $A \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and a given integer $p$ that satisfies $1 \leq p \leq d$, we define a $p$-subvector of a tensor as a fiber of the tensor in the direction $e_p$. We denote the $p$-subvector of $A$ by

$$v_{i_1,\ldots,i_{p-1},i_{p+1},\ldots,i_d} := A(i_1,\ldots,i_{p-1},i_p,\ldots,i_d).$$
Suppose we have \( R \) nonzero \( p \)-subvectors of \( \mathcal{A} \) with indices \((i_1, \ldots, i_p-1, i_p+1, \ldots, i_d)\) forming a set \( S_p \) with \(|S_p| = R\). Then, \( \mathcal{A} \) can be represented as the sum of \( R \) rank-1 tensors.

\[
(3.2) \quad \mathcal{A} = \sum_{(i_1, \ldots, i_p-1, i_p+1, \ldots, i_d) \in S_p} e_{i_1} \circ \cdots \circ e_{i_p-1} \circ \mathbf{v}_{i_1, \ldots, i_p-1, i_p+1, \ldots, i_d} \circ e_{i_p+1} \circ \cdots \circ e_{i_d},
\]

where \( e_{i_k} \in \mathbb{R}^{n_k} \) is the standard basis vector.

**Lemma 3.1.** Any rank-1 tensor is equivalent to a tensor train whose TT rank is 1.

\[
(3.3) \quad \mathbf{v}_1 \circ \mathbf{v}_2 \circ \cdots \circ \mathbf{v}_d = \mathbf{V}^{(1)} \circ \mathbf{V}^{(2)} \circ \cdots \circ \mathbf{V}^{(d)},
\]

where \( \mathbf{v}_k \in \mathbb{R}^{n_k} \), \((k = 1, \ldots, d)\), \( \mathbf{V}^{(1)} = \text{reshape}(\mathbf{v}_1, [n_1, 1]) \), \( \mathbf{V}^{(d)} = \text{reshape}(\mathbf{v}_d, [1, n_d]) \), and \( \mathbf{V}^{(k)} = \text{reshape}(\mathbf{v}_k, [1, n_k]) \) for \( 1 < k < d \).

**Lemma 3.2.** [16, p. 2308] Suppose we have two tensors \( \mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d} \) and \( \mathcal{B} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d} \) in the TT format,

\[
\begin{align*}
\alpha_{i_1 i_2 \ldots i_d} &= \mathcal{A}^{(1)}_{i_1} \mathcal{A}^{(2)}_{i_2} \cdots \mathcal{A}^{(d)}_{i_d}, \\
b_{i_1 i_2 \ldots i_d} &= \mathcal{B}^{(1)}_{i_1} \mathcal{B}^{(2)}_{i_2} \cdots \mathcal{B}^{(d)}_{i_d}.
\end{align*}
\]

The TT cores of the sum \( \mathcal{C} = \mathcal{A} + \mathcal{B} \) in the TT format then satisfy

\[
(3.4) \quad C^{(k)}_{i_k} = \begin{bmatrix} A^{(k)}_{i_k} & O \\ O & B^{(k)}_{i_k} \end{bmatrix}, \quad k = 2, \ldots, d - 1,
\]

\[
C^{(1)}_{i_1} = \begin{bmatrix} A^{(1)}_{i_1} \\ B^{(1)}_{i_1} \end{bmatrix}, \quad \quad C^{(d)}_{i_d} = \begin{bmatrix} A^{(d)}_{i_d} \\ B^{(d)}_{i_d} \end{bmatrix},
\]

where \( O \) denotes a zero matrix of appropriate dimensions.

The proof of Lemmas 3.1 and 3.2 can be easily derived from Definitions 2.4 to 2.6 and the definition of the tensor train decomposition.

Based on (3.2) and Lemmas 3.1 and 3.2, we have the following theorem.

**Theorem 3.3.** A sparse tensor \( \mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d} \) can be transformed into an equivalent tensor train with TT rank \( R \), where \( R \) is the number of nonzero \( p \)-subvectors in \( \mathcal{A} \) \((1 \leq p \leq d)\). If \( p \neq 1 \) or \( d \),

\[
(3.5) \quad \mathcal{A} = \mathcal{P}^{(1)} \circ \cdots \circ \mathcal{P}^{(p-1)} \circ \mathbf{V} \circ \mathcal{P}^{(p+1)} \circ \cdots \circ \mathcal{P}^{(d)},
\]

where \( \mathcal{P}^{(k)} \in \{0, 1\}^{R \times n_k \times R}, (2 \leq k \leq d, k \neq p) \), \( \mathcal{P}^{(1)} \in \{0, 1\}^{1 \times n_1 \times R} \), \( \mathcal{P}^{(d)} \in \{0, 1\}^{R \times n_d \times 1} \), and \( \mathbf{V} \in \mathbb{R}^{R \times n_p \times R} \). Similar expressions hold for the situations with \( p = 1 \) or \( d \).

The TT cores \( \mathcal{P}^{(k)} \) and \( \mathbf{V} \) in Theorem 3.3 are sparse tensors, whose nonzero distributions are illustrated in Figure 2. Each horizontal bar depicted in Figure 2 is a standard basis vector \( e_{i_k} \) for \( \mathcal{P}^{(k)} \) or a \( p \)-subvector \( \mathbf{v} \) for \( \mathbf{V} \). The derived matrices \((\mathcal{P}^{(k)}_{i_k} \text{ and } \mathbf{V}_{i_p})\) from these TT cores are all diagonal matrices. Furthermore, each of the \( \mathcal{P}^{(k)} \) cores is very sparse, as the nonzero elements consist of only \( R \) 1’s.
3.2. Efficient parallel-vector rounding. With Theorem 3.3, we can convert a sparse tensor into an exact TT representation. However, the TT rank \( R \) is usually large. A parallel-vector rounding technique [10] can be applied to reduce the TT ranks. Its idea is to remove duplicate vectors. The following example illustrates how this can reduce the dimensions of a matrix.

Example 3.4. With the parallel-vector rounding, the following matrix can be represented as the product of two smaller matrices.

\[
\begin{bmatrix}
  a_1 & b_1 & a_1 & a_1 \\
  a_2 & b_2 & a_2 & a_2 \\
  a_3 & b_3 & a_3 & a_3 \\
  a_4 & b_4 & a_4 & a_4 \\
\end{bmatrix}
= \begin{bmatrix}
  a_1 & b_1 \\
  a_2 & b_2 \\
  a_3 & b_3 \\
  a_4 & b_4 \\
\end{bmatrix}
\begin{bmatrix}
  1 & 0 & 1 & 1 \\
  0 & 1 & 0 & 0 \\
\end{bmatrix}.
\]

Applying this technique to the TT format obtained in Theorem 3.3, we obtain a lossless sparse tensor to TT conversion algorithm, described as Algorithm 3.1, where the function Deduplication refers to this parallel-vector rounding.

The correctness of Algorithm 3.1 is due to Theorem 3.3 and the associative property of matrix/tensor multiplications. The graphical representations of the decomposition forms during the algorithm execution are shown in Figure 3 for a 4-way TT, where each of the \( Q^{(k)} \) matrices is a column selection matrix as in the right-hand side of equation (3.6).

An important observation is that each matrix \( Q \) in Algorithm 3.1 is a quasi-permutation matrix. This will enable fast execution of the parallel-vector rounding (function Deduplication).

**Definition 3.5. Quasi-permutation matrix.** If each column of a matrix has only one nonzero element with a value of 1, then the matrix is called a quasi-permutation matrix.

Obviously, a permutation or identity matrix belongs to the class of quasi-permutation matrices. It turns out that the matrices \( \text{split}_1(\mathcal{P}^{(k)}) \) and \( \text{split}_2(\mathcal{P}^{(k)}) \) \( (k \neq p) \) derived from the TT cores \( \mathcal{P}^{(k)} \) in (3.5) are also quasi-permutation matrices.

**Theorem 3.6.** In Algorithm 3.1, each input matrix of the function Deduplication is a quasi-permutation matrix.
Algorithm 3.1: Sparse TT conversion with parallel-vector rounding

Input: A sparse tensor \( \mathbf{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d} \), an integer \( p (1 \leq p \leq d) \).

Output: Core tensors \( \mathbf{G}^{(1)}, \ldots, \mathbf{G}^{(d)} \) of TT-format tensor \( \mathbf{B} \) which is equivalent to \( \mathbf{A} \) with TT ranks \( \tilde{r}_k \) \( (k = 0, 1, \ldots, d) \).

1. Initialize empty cores \( \mathbf{G}^{(1)}, \ldots, \mathbf{G}^{(d)} \) for TT-format tensor \( \mathbf{B} \).
2. for every \( v \in \{ \) all \( R \) nonzero \( p \)-subvectors of \( \mathbf{A} \) \} do
3. Determine \( (d-1) \) \( e \), vectors in (3.2).
4. Construct rank-1 TT \( \mathbf{T} \) with \( v \) and \( e \) vectors as Lemma 3.1.
5. \( \mathbf{B} := \mathbf{B} + \mathbf{T} \), which means \( \mathbf{G}^{(1)}, \ldots, \mathbf{G}^{(d)} \) are update with (3.4).
6. end for
7. \( \tilde{r}_0 := 1. \)
8. for \( k = 1, \ldots, p-1 \) do
9. \( [\mathbf{N}, \mathbf{Q}] := \text{Deduplication(split}_2(\mathbf{G}^{(k)})) \), where \( \mathbf{N} \in \mathbb{R}^{\tilde{r}_{k-1} \times n_k \times \tilde{r}_k}, \mathbf{Q} \in \mathbb{R}^{\tilde{r}_k \times R} \).
10. \( \mathbf{G}^{(k)} := \text{reshape}(\mathbf{N}, [\tilde{r}_{k-1}, n_k, \tilde{r}_k]) \).
11. \( \mathbf{G}^{(k+1)} := \mathbf{G}^{(k+1)} \times_1 Q^T. \)
12. end for
13. \( \tilde{r}_d := 1. \)
14. for \( k = d, \ldots, p+1 \) do
15. \( [\mathbf{N}, \mathbf{Q}] := \text{Deduplication(split}_1(\mathbf{G}^{(k)})) \), where \( \mathbf{N} \in \mathbb{R}^{\tilde{r}_k \times n_k \times \tilde{r}_{k-1}}, \mathbf{Q} \in \mathbb{R}^{\tilde{r}_{k-1} \times R} \).
16. \( \mathbf{G}^{(k)} := \text{reshape}(\mathbf{N}^T, [\tilde{r}_{k-1}, n_k, \tilde{r}_k]) \).
17. \( \mathbf{G}^{(k-1)} := \mathbf{G}^{(k-1)} \times_3 Q^T. \)
18. end for
19. Return tensor \( \mathbf{B} \) in TT-format with cores \( \mathbf{G}^{(1)}, \ldots, \mathbf{G}^{(d)} \).

Fig. 3. The graphical representations of the decomposition forms during the Algorithm 3.1 execution for a 4-way TT \((p = 3)\)

Proof. \( \mathbf{G}^{(1)} \) is definitely a quasi-permutation matrix according to (3.3) and (3.4). For \( 2 \leq k < p \), a new \( \mathbf{G}^{(k)} \) is computed as \( \mathbf{G}^{(k)} \times_1 Q^T \) \((\text{split}_1(\mathbf{G}^{(k)}) = Q \times \text{split}_1(\mathbf{G}^{(k)}))\).

of reasoning can be used to prove the theorem for $k > p$

From this it follows that split_2 has the structure

\[
\begin{bmatrix}
  x_{11} & 0 & \cdots & 0 & x_{12} & 0 & \cdots & 0 & x_{1n_k} & 0 & \cdots & 0 \\
  0 & x_{21} & \cdots & 0 & 0 & x_{22} & \cdots & 0 & 0 & x_{2n_k} & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & x_{R1} & 0 & 0 & \cdots & x_{R2} & 0 & 0 & \cdots & x_{Rn_k} \\
\end{bmatrix},
\]

where $\forall j = 1, 2, \ldots, R$, vector $[x_{j1}, x_{j2}, \cdots, x_{jn_k}]^T$ is a particular standard basis vector. Then if $Q = [e_{i_1}, e_{i_2}, \cdots, e_{i_R}]$, where $e_{i_k}$ is the standard basis vector, split_2($\mathcal{G}'(k)$) = $Q \times$ split_1($\mathcal{G}(k)$) will have the structure

\[
\begin{bmatrix}
  x_{11} e_{i_1} & x_{21} e_{i_2} & \cdots & x_{R1} e_{i_R} & x_{1n} e_{i_1} & x_{2n} e_{i_2} & \cdots & x_{Rn} e_{i_R} \\
\end{bmatrix},
\]

and the structure of split_2($\mathcal{G}(k)$) will be

\[
\begin{bmatrix}
  x_{11} e_{i_1} & x_{21} e_{i_2} & \cdots & x_{R1} e_{i_R} \\
  x_{12} e_{i_1} & x_{22} e_{i_2} & \cdots & x_{R2} e_{i_R} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{1n} e_{i_1} & x_{2n} e_{i_2} & \cdots & x_{Rn} e_{i_R} \\
\end{bmatrix}.
\]

From this it follows that split_2($\mathcal{G}(k)$) is a quasi-permutation matrix. The same line of reasoning can be used to prove the theorem for $k > p$. □

Now, we consider how to perform parallel-vector rounding for a quasi-permutation matrix. Our aim is to express a matrix $M$ as the product of two smaller matrices: $M = NQ$. For a quasi-permutation matrix, we need to remove the duplicate columns in $M$. As shown in Example 3.4, the result $Q$ itself is a quasi-permutation matrix. So, $N = I$ and $Q = M$, where $I$ is an identity matrix, can be regarded as the result of performing parallel-vector rounding on a quasi-permutation matrix, except that the duplicate columns in $M$ have not yet been removed. What remains to be done is the removal of zero rows of $Q$ and the corresponding columns in $N$. This is described as Algorithm 3.2.

**Algorithm 3.2** Deduplication for a quasi-permutation matrix

**Input:** A quasi-permutation matrix $M \in \mathbb{R}^{n_1 \times n_2}$.

**Output:** Matrices $N \in \mathbb{R}^{n_1 \times \beta}$, $Q \in \mathbb{R}^{\beta \times n_2}$ so that $M = NQ$, and $N$ includes nonduplicate columns of $M$.

1. $\beta := 0$.
2. Let $N \in \mathbb{R}^{n_1 \times \beta}$, $Q \in \mathbb{R}^{\beta \times n_2}$ be two dynamically resized transfer matrices.
3. for $i = 1, 2, \cdots, n_1$ do
4. if $M_{i,:}$ is not a zero row then
5. $\beta := \beta + 1$ \hspace{1cm} \# append matrix $N$ and $Q$
6. $Q_{i,\beta} := M_{i,:}$.
7. Set $N_{i,\beta}$ a zero column except $N_{i,\beta} = 1$
8. end if
9. end for
10. Return $N$ and $Q$.

For a quasi-permutation matrix, each column can be represented by the position of 1 in it. Thus, Algorithm 3.2 has a time complexity of $O(n_1 + n_2)$, where $n_1$ and
are the dimensions of $M$. It can be executed much more efficiently than a general parallel-vector removing algorithm. From Algorithm 3.2 we can also observe, that the resulting matrix size $\beta$ must be no more than $n_1$, even if $n_2 \gg n_1$.

According to Theorem 3.6 and the above analysis, with Algorithm 3.1 the TT ranks will be reduced to $\tilde{r}$ satisfying the following upper bounds

$$\tilde{r}_k \leq \bar{r}_k = \begin{cases} \min\left(R, \prod_{i=1}^{k} n_i\right) & \text{if } 1 \leq k < p, \\ \min\left(R, \prod_{i=k+1}^{d} n_i\right) & \text{if } p \leq k < d. \end{cases}$$ (3.7)

where $R$ is the number of nonzero $p$-subvectors in the original tensor $A$.

### 3.3. TT-rounding and the FastTT algorithm.

The TT-rounding algorithm \cite{16} is similar to the TT-SVD algorithm and can reduce the TT ranks of a tensor train for a given accuracy tolerance. The algorithm consists of $(d-1)$ QR decompositions and SVDs. As shown in Algorithm 3.3, we modify the TT-rounding algorithm to adapt it to the tensor train produced by Algorithm 3.1. The truncation parameters in Algorithm 3.3 satisfy

$$\delta_k := \frac{\varepsilon}{\sqrt{p - 1} + \sqrt{d - p}} \|A\|_F, \quad k = 1 \ldots d - 1.$$ (3.8)

The correctness of Algorithm 3.3 is explained as follows.

**Lemma 3.7.** A quasi-permutation matrix with no duplicate columns is an orthonormal matrix.

**Lemma 3.7** can be easily proved based on the Definition 3.5 and the definition of an orthonormal matrix.

In Steps 9 and 15 of Algorithm 3.1, the duplicate columns in the input quasi-permutation matrix (according to Theorem 3.6) are removed. Then, based on Lemma 3.7, we have the following statement.

**Corollary 3.8.** Supposes the TT cores $G^{(k)}$ are obtained with Algorithm 3.1. Then the matrices $\text{split}_2(G^{(k)})$, $k < p$ and $\text{split}_1(G^{(k)})$, $k > p$ are all orthonormal matrices.

**Lemma 3.9.** Suppose $U^{(i)}$, $i = 1, \ldots, d$ are the cores of a tensor train. If matrix $\text{split}_2(U^{(i)})$ is an orthonormal matrix for all $i = 1, \ldots, k$ ($1 \leq k \leq d$), then the matrix $\text{split}_j(U^{(1)} \circ \cdots \circ U^{(j)})$ is an orthonormal matrix for all $j = 1, \ldots, k$.

The proof of Lemma 3.9 can be found in \cite[Appendix B]{19}. We can now derive the following theorem.

**Theorem 3.10.** (Correctness of Algorithm 3.3) The approximation $B$ obtained in Algorithm 3.3 always satisfies $\|A - B\|_F \leq \varepsilon \|A\|_F$.

**Proof.** Let $C$ denote the contraction of the cores $(G^{(1)} \circ \cdots \circ G^{(d)})$ after the first loop of Algorithm 3.3 and $D$ denote the contraction after the second loop. Obviously $C$ is equal to $D$ because there are mainly QR decompositions in the second loop. Then we have

$$\|A - B\|_F = \|A - C + D - B\|_F \leq \|A - C\|_F + \|D - B\|_F.$$
Algorithm 3.3 The TT-rounding for the sparse TT conversion

Input: Cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(d)}$ of the TT-format tensor $\mathcal{A}$ with TT-ranks $r_1, \ldots, r_{d-1}$, desired accuracy tolerance $\varepsilon$, an integer $p (1 \leq p \leq d)$.

Output: Cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(d)}$ of the TT-approximation $\mathcal{B}$ to $\mathcal{A}$ in the TT-format with TT-ranks $r_1, \ldots, r_{d-1}$. The computed approximation satisfies

$$||\mathcal{A} - \mathcal{B}||_F \leq \varepsilon ||\mathcal{A}||_F.$$ 

1: Set truncation parameters according to (3.8).
2: for $k = p, \ldots, d-1$ do
3: $G := \text{split}_2(\mathcal{G}^{(k)})$.
4: Compute $\delta_k$-truncated SVD: $G = U \Sigma V^T + E_k$, $||E_k||_F \leq \delta_k$, $r_k := \text{rank}_{\delta_k}(G)$.
5: $\mathcal{G}^{(k)} := \text{reshape}(U,[r_{k-1},n_k,r_k])$.
6: $\mathcal{G}^{(k+1)} := \mathcal{G}^{(k+1)} \times_1 (V \Sigma)$.
7: end for
8: for $k = d, \ldots, p+1$ do
9: $G := \text{split}_1(\mathcal{G}^{(k)})$.
10: Compute economic QR decomposition: $G = QR$.
11: $\mathcal{G}^{(k)} := \text{reshape}(Q^T,[r_{k-1},n_k,r_k])$.
12: $\mathcal{G}^{(k-1)} := \mathcal{G}^{(k-1)} \times_3 R^T$.
13: end for
14: for $k = p, \ldots, 2$ do
15: $G := \text{split}_1(\mathcal{G}^{(k)})$.
16: Compute $\delta_{k-1}$-truncated SVD: $G = U \Sigma V^T + E_{k-1}$, $||E_{k-1}||_F \leq \delta_{k-1}$, $r_{k-1} := \text{rank}_{\delta_{k-1}}(G)$.
17: $\mathcal{G}^{(k)} := \text{reshape}(U^T,[r_{k-1},n_k,r_k])$.
18: $\mathcal{G}^{(k-1)} := \mathcal{G}^{(k-1)} \times_3 (V \Sigma)$.
19: end for
20: Return $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(d)}$ as cores of $\mathcal{B}$.

$\mathcal{C}$ with $\mathcal{A} = \mathcal{L} \circ \mathcal{A'}$ and $\mathcal{C} = \mathcal{L} \circ \mathcal{C'}$. According to Corollary 3.8 and Lemma 3.9, $L = \text{split}_{p-1}(\mathcal{L})$ is an orthonormal matrix. Thus

$$||\mathcal{A} - \mathcal{C}||_F = ||\mathcal{L} \circ \mathcal{A'} - \mathcal{L} \circ \mathcal{C'}||_F = ||L \mathcal{A'} - L \mathcal{C'}||_F = ||\mathcal{A'} - \mathcal{C'}||_F = ||\mathcal{A'} - \mathcal{C'}||_F,$$

where $\mathcal{A'} = \text{split}_1(\mathcal{A'})$ and $\mathcal{C'} = \text{split}_1(\mathcal{C'})$.

Focus on the first iteration $(k = p)$ of the first loop. Let $\mathcal{G}$ be the $i$-th core of $\mathcal{A}$, $\mathcal{U}$ be the $k$-th core of $\mathcal{C}$, $\mathcal{R}$ be $\mathcal{G}^{(k+1)} \circ \cdots \circ \mathcal{G}^{(d)}$ before the first iteration, $\mathcal{A}_1$ be $\mathcal{G}^{(k+1)} \circ \cdots \circ \mathcal{G}^{(d)}$ after the first iteration and $\mathcal{C}_1$ be $\mathcal{G}^{(k+1)} \circ \cdots \circ \mathcal{G}^{(d)}$ after the whole loop. Let $A = \text{split}_2(\mathcal{A'})$, $C = \text{split}_2(\mathcal{C'})$, $G = \text{split}_2(\mathcal{G})$, $U = \text{split}_2(\mathcal{U})$, $R = \text{split}_1(\mathcal{R})$, $A_1 = \text{split}_1(\mathcal{A}_1)$ and $C_1 = \text{split}_1(\mathcal{C}_1)$. According to Corollary 3.8 and Lemma 3.9, $R^T$ is an orthonormal matrix. The SVD in the first iteration can be written as $G = U \Sigma V^T + E_k$, where $||E_k||_F \leq \delta_k$. From Line 6 of Algorithm 3.3 we know $A_1 = \Sigma V^T R$. From the properties of the SVD it follows that $U^T E_k = 0$, $V^T E_k = 0$, and $R^T E_k = 0$. The desired accuracy tolerance $\varepsilon$ can be achieved in the first iteration. In all other iterations, the error is reduced by a factor of at least $\delta_k$, which is sufficient to reach the desired accuracy.
which means $(U(A_1 - C_1))^T(E_k R) = 0$. Thus

$$\|A' - C'\|_F^2 = \|A - C\|_F^2$$

$$= \|GR - UC_1\|_F^2$$

$$= \|\langle U\Sigma V^T + E_k \rangle R - UC_1\|_F^2$$

$$= \|U\Sigma V^T R + E_k R - UC_1\|_F^2$$

$$= \|UA_1 + E_k R - UC_1\|_F^2$$

$$= \|E_k R\|_F^2 + \|U(A_1 - C_1)\|_F^2$$

Note that for the new core $G^{(p)}$ after the first iteration split,$G^{(p)} = U$ is also an orthonormal matrix, which makes the matrix $L$ in the next iteration ($k = p + 1$) still orthonormal. Proceeding by induction, we have

$$\|A - C\|_F^2 = \sum_{k=p}^{d-1} \|E_k\|_F^2.$$ 

Similarly we have

$$\|D - B\|_F^2 = \sum_{k=2}^{p} \|E_{k-1}\|_F^2.$$ 

Thus

$$\|A - B\|_F \leq \|A - C\|_F + \|D - B\|_F = \sqrt{\sum_{k=p}^{d-1} \|E_k\|_F^2 + \sum_{k=2}^{p} \|E_{k-1}\|_F^2} \leq \sqrt{\sum_{k=p}^{d-1} \delta_k^2 + \sum_{k=2}^{p} \delta_{k-1}^2}.$$ 

According to (3.8) and (3.9), we have

$$\|A - B\|_F \leq \sqrt{\sum_{k=p}^{d-1} \delta_k^2 + \sum_{k=2}^{p} \delta_{k-1}^2} \leq \varepsilon \|A\|_F.$$

Now, we are ready to describe the whole algorithm for the conversion of a sparse tensor into a TT, presented as Algorithm 3.4.

It should be pointed out that if the accuracy tolerance $\varepsilon$ is set to 0\(^1\), the obtained TT ranks with Algorithm 3.4 will be maximal and equal to the TT-ranks obtained from the TT-SVD algorithm. We take $p = 1$ as an example to discuss the TT-rank $r_1$ case. In the TT-SVD algorithm, $r_1$ is obtained by computing the SVD of $C = \text{split}_1(A)$. $A$ can also represented as the contraction of the TT cores obtained by Algorithm 3.1.

$$A = G^{(1)} \circ G^{(2)} \circ \cdots \circ G^{(d)}.$$

\(^1\)In practice, $\varepsilon$ is usually set to a small value like $10^{-14}$ due to the inevitable round-off error.
Algorithm 3.4 Tensor train decomposition of sparse tensor (FastTT)

Input: A sparse tensor \( \mathbf{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d} \), desired accuracy tolerance \( \varepsilon \), an integer \( p \) \( (1 \leq p \leq d) \).

Output: Cores \( \mathbf{G}^{(1)}, \ldots, \mathbf{G}^{(d)} \) of the TT-approximation \( \mathbf{B} \) to \( \mathbf{A} \) in the TT-format with TT-ranks \( r_k \).

The computed approximation satisfies

\[ \| \mathbf{A} - \mathbf{B} \|_F \leq \varepsilon \| \mathbf{A} \|_F. \]

1: Use Algorithm 3.1 to obtain \( \mathbf{B} \) in the TT-format with cores \( \mathbf{G}^{(1)}, \ldots, \mathbf{G}^{(d)} \).
2: Use Algorithm 3.3 to reduce the TT ranks of \( \mathbf{B} \).
3: Return the TT-approximation \( \mathbf{B} \).

Thus,

\[ \mathbf{C} = \text{split}_2(\mathbf{G}^{(1)}) \text{split}_1(\mathbf{G}^{(2)} \circ \cdots \circ \mathbf{G}^{(d)}). \]

According to Corollary 3.8 and Lemma 3.9, \( \mathbf{L} = \text{split}_T(\mathbf{G}^{(2)} \cdots \mathbf{G}^{(d)}) \) is an orthonormal matrix, i.e. \( \mathbf{L}^T \mathbf{L} = \mathbf{I} \).

\[ \mathbf{C} \mathbf{C}^T = \text{split}_2(\mathbf{G}^{(1)}) \mathbf{L}^T \mathbf{L} \text{split}_2(\mathbf{G}^{(1)}) = \text{split}_2(\mathbf{G}^{(1)}) \text{split}_2(\mathbf{G}^{(1)}). \]

This means matrix \( \text{split}_2(\mathbf{G}^{(1)}) \) has the same singular values as \( \mathbf{C} \). For Algorithm 2.1, \( r_1 \) equals \( \text{rank}_3(\mathbf{C}) \), while the \( r_1 \) obtained with Algorithm 3.4 is \( \text{rank}_3(\text{split}_2(\mathbf{G}^{(1)})) \) (see Lines 3 and 4 of Algorithm 3.3). These numerical ranks are therefore equal when \( \delta = \delta_1 \). Similar results for the other TT ranks and for \( p \neq 1 \) can be derived.

For a sparse tensor the runtime of Algorithm 3.4 may be smaller than the TT-SVD algorithm, as the SVD is performed on smaller matrices.

3.4. Fixed-rank TT approximations and matrices in TT-format. Sometimes we need a TT approximation of a tensor with given TT-ranks. We can slightly modify Algorithm 3.3 to fit this scenario. Specifically, the desired accuracy tolerance \( \varepsilon \) is not needed and thus substituted with the desired TT-ranks. The truncation parameters \( \delta_i \) will not be computed either. In the truncated SVD computation we simply truncate the matrices with the given ranks instead of truncating them according to the accuracy tolerance. This technique could be useful in applications like tensor completion [11].

Some other applications require matrix-vector multiplications, which are convenient if both the matrix and the vector is in TT-format (as shown in Figure 4). A vector \( \mathbf{v} \in \mathbb{R}^N \) can be transformed into TT-format if we first reshape it into a tensor \( \mathbf{V} \in \mathbb{R}^{n_1 \times \cdots \times n_d} \), where \( N = n_1 \cdots n_d \), and then decompose it into a TT. A “matrix in TT-format” [16, pp. 2311-2313], also known as a matrix product operator (MPO), is similar but more complicated. The elements of matrix \( \mathbf{M} \in \mathbb{R}^{M \times N} \) are rearranged into a tensor \( \mathbf{M} \in \mathbb{R}^{m_1 \times n_1 \times \cdots \times m_d \times n_d} \), where \( M = m_1 \cdots m_d, N = n_1 \cdots n_d \). The cores \( \mathbf{M}^{(i)}(i = 1, \ldots, d) \) of the MPO satisfy

\[ \mathbf{M}^{(i)}(i_1, j_1, \ldots, i_d, j_d) = \mathbf{M}^{(1)}(:, i_1, j_1, :) \cdots \mathbf{M}^{(d)}(:, i_d, j_d, :), \]

where \( \mathbf{M}^{(i)} \in \mathbb{R}^{r_{i-1} \times m_i \times n_i \times r_i}(i = 1, \ldots, d), r_0 = r_d = 1 \). The matrix-to-MPO algorithm is basically computing a TT-decomposition of the \( d \)-way tensor \( \mathbf{M}' \in \mathbb{R}^{M \times N} \) such that

\[ \| \mathbf{M} - \mathbf{M}' \|_F \leq \varepsilon \| \mathbf{M} \|_F. \]
\[ \mathbb{R}^{m_1 n_1 \times \cdots \times m_d n_d} \], along with a few necessary reshapings, which can also be done with Algorithm 3.4.

\[ \begin{array}{c}
M^{(1)} \\
M^{(2)} \\
M^{(3)} \\
\vdots
\end{array} \quad \begin{array}{c}
V^{(1)} \\
V^{(2)} \\
V^{(3)} \\
\vdots
\end{array} \quad \begin{array}{c}
r_0 \\
r_1 \\
r_2 \\
r_3 \\
\vdots
\end{array} \quad \begin{array}{c}
r_0' \\
r_1' \\
r_2' \\
r_3' \\
\vdots
\end{array}
\]

**Fig. 4.** Diagram of matrix-vector-multiplication in the TT-format.

### 3.5. A dynamic method to choose the truncation parameters.

The actual relative error of the truncated SVD is usually not very close to the truncation parameter \( \delta_k \), which implies that if the truncation parameters are set statically at the beginning with (3.8), some of the desired accuracy tolerant \( \varepsilon \) will be “wasted”. The main idea of the dynamic method is to compute the truncation parameters dynamically to make use of those “wastes”. One of the possible approaches is shown in Algorithm 3.5. In each step of the truncated SVD, an expected error is calculated with the current “total error remainder” and used as the truncation parameter. After each truncated SVD, the “total error remainder” is decreased according to the actual error. Such an adaptive approach to setting the truncation parameters can lead to lower TT ranks while keeping the relative error smaller than \( \varepsilon \).

**Algorithm 3.5** The revised TT-rounding for the sparse TT conversion

**Input:** (same as Algorithm 3.3)  
**Output:** (same as Algorithm 3.3)

1. \( \delta_{\text{right}} := \sqrt{d - p + \sqrt{1 + \frac{1}{d - p}}} \varepsilon \|A\|_F \), \( \delta_{\text{left}} := \sqrt{p - 1} \sqrt{d - p + \sqrt{1 + \frac{1}{d - p}}} \varepsilon \|A\|_F \).
2. for \( k = p, \ldots, d - 1 \) do
   3. \( \delta_k := \frac{\delta_{\text{right}}}{\sqrt{d - k}} \)
   4. Steps 3-6 of Algorithm 3.3.
   5. \( \delta_{\text{right}} := \sqrt{\delta_{\text{right}}^2 - \|E_k\|_F^2} \).
3. end for
4. for \( k = p, \ldots, 2 \) do
   5. \( \delta_{k - 1} := \frac{\delta_{\text{left}}}{\sqrt{k - 1}} \).
   6. Steps 15-18 of Algorithm 3.3.
   7. \( \delta_{\text{left}} := \sqrt{\delta_{\text{left}}^2 - \|E_{k - 1}\|_F^2} \).
4. end for
8. Return \( G^{(1)}, \ldots, G^{(d)} \) as cores of \( B \).

**Theorem 3.11.** (Correctness of Algorithm 3.5) The approximation \( B \) obtained in Algorithm 3.5 always satisfies \( \|A - B\|_F \leq \varepsilon \|A\|_F \).

**Proof.** According to (3.9), we have that

\[
\|A - B\|_F \leq \left( \sum_{k=p}^{d-1} \|E_k\|_F^2 \right)^{\frac{1}{2}} + \left( \sum_{k=2}^{p} \|E_{k-1}\|_F^2 \right)^{\frac{1}{2}}.
\]
Now we are going to use a loop invariant [4, pp. 18-19] to prove the correctness of Algorithm 3.5. The loop invariant for Loop 2-6 is

$$\delta^2_{\text{right}} + \sum_{i=p}^{k-1} \|E_i\|_F^2 = \frac{d - p}{\sqrt{d - p} + \sqrt{p - 1}^{2}} \varepsilon \|A\|_F^2.$$  

**Initialization:** Before the first iteration $k = p$, $\sum_{i=p}^{k-1} \|E_i\|_F^2 = 0$ and $\delta_{\text{right}} = \frac{\sqrt{d - p}}{\sqrt{d - p} + \sqrt{p - 1}}^{2} \varepsilon \|A\|_F$. Thus the invariant is satisfied.

**Maintenance:** After each iteration, $\delta_{\text{right}}$ is decreased by $\|E_k\|_F^2$ and $\sum_{i=p}^{k-1} \|E_i\|_F^2$ is increased by $\|E_k\|_F^2$. Thus the invariant remains satisfied.

**Termination:** When the loop terminates at $k = d$. Again the loop invariant is satisfied. This means that

$$\delta^2_{\text{right}} + \sum_{i=p}^{d-1} \|E_i\|_F^2 = \frac{d - p}{\sqrt{d - p} + \sqrt{p - 1}}^{2} \varepsilon \|A\|_F^2.$$  

Consequently we can prove that

$$\sqrt{\sum_{k=2}^{p} \|E_{k-1}\|_F^2} \leq \frac{\sqrt{p - 1}}{\sqrt{d - p} + \sqrt{p - 1}}^{2} \varepsilon \|A\|_F,$$

is satisfied after Loop 8-12.

Thus

$$\|A - B\|_F \leq \sqrt{\sum_{k=p}^{d-1} \|E_k\|_F^2 + \sum_{k=2}^{p} \|E_{k-1}\|_F^2}.$$  

\[\leq \frac{\sqrt{d - p}}{\sqrt{d - p} + \sqrt{p - 1}}^{2} \varepsilon \|A\|_F + \frac{\sqrt{p - 1}}{\sqrt{d - p} + \sqrt{p - 1}}^{2} \varepsilon \|A\|_F\]

$$= \varepsilon \|A\|_F. \quad \square$$

### 3.6. Complexity Analysis

Finding nonzero $p$-subvectors can be accelerated by employing balanced binary search trees or hash tables, while parallel-vector rounding will be efficient if deduplication is implemented in Algorithm 3.2. Notice that, there is no floating point operation in these procedures. Therefore, the time complexity of Algorithm 3.4 mainly depends on Algorithm 3.3, where the cost of SVD is of major concern. The FLOP count $f_{\text{fasttt}}$ can thus be estimated as

\[ (3.10) \quad f_{\text{fasttt}} \approx f_{\text{SVD}}(\tilde{r}_{p-1}n_p, \tilde{r}_p) + \sum_{i=p+1}^{d-1} f_{\text{SVD}}(r_{i-1}n_i, \tilde{r}_i) + \sum_{i=2}^{p} f_{\text{SVD}}(\tilde{r}_{i-1}, n_ir_i), \]

where $\{\tilde{r}_k\}$ and $\{r_k\}$ are the TT-ranks before and after executing Algorithm 3.3. According to (3.7), where the upper bound of $\tilde{r}_k$, i.e., $\tilde{r}_k$, is given, we can estimate the
Algorithm 3.6 Automatically select $p$

**Input:** A sparse tensor $\mathbf{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$.

**Output:** Selected $\bar{p}$ for best estimated performance in Algorithm 3.4.

1: for $p = 1, \ldots, d$ do
2: $R :=$ the number of nonzero $p$-subvectors of $\mathbf{A}$.
3: $\{\bar{r}_k\} := \{\tilde{r}_k\}$ given in (3.7).
4: $\{r_k\} := \{\bar{r}_k\}$, or specified by users.
5: $f_p :=$ the estimated FLOP count in (3.10).
6: end for
7: Return $\bar{p} = \arg\min_p f_p$.

upper bound of the FLOP count before any actual computation. With this estimation, $p$ can be automatically selected as described in Algorithm 3.6. In Line 3, $\{\tilde{r}_k\}$ can be obtained alternatively by actually executing deduplication for a more precise estimation since it will not take much time after all.

For a more intuitive view of the time complexity, we analyze the FLOP counts for an example from Section 4.1. Suppose we are computing a fixed rank-10 TT-approximation of a sparse 7-way tensor $\mathbf{A} \in \mathbb{R}^{10 \times 20 \times 20 \times 10 \times 15 \times 20 \times 3}$ with density $\sigma = 0.001$. According to (2.1), the approximate FLOP count of TT-SVD is

$$f_{\text{TTSVD}} \approx f_{\text{SVD}}(10, 20 \times 20 \times 10 \times 15 \times 20 \times 3) + f_{\text{SVD}}(20r, 20 \times 10 \times 15 \times 20 \times 3)$$
$$\approx (3.6 \times 10^8 + 7.6r^2 \times 10^7)C_{\text{SVD}}$$
$$\approx (8 \times 10^9)C_{\text{SVD}}.$$

As for $f_{\text{fasttt}}$, we let $p = 7$. Since the elements of $\mathbf{A}$ are grouped in triples stored in the last dimension, the number of nonzero 7-subvectors $R$ satisfies $R \leq \text{nnz}(\mathbf{A})/3 = 12000$, which means $\{\tilde{r}_k\}$ given in (3.7) is no more than $\{10, 200, 4000, 12000, 12000, 12000\}$. According to (3.10), we have

$$f_{\text{fasttt}} \approx f_{\text{SVD}}(3, 120000) + f_{\text{SVD}}(20r, 120000) + f_{\text{SVD}}(15r, 12000)$$
$$+ f_{\text{SVD}}(10r, 4000) + f_{\text{SVD}}(20r, 200)$$
$$\approx (1.08 \times 10^6 + 8r^2 \times 10^6)C_{\text{SVD}}$$
$$\approx (8 \times 10^8)C_{\text{SVD}}.$$

In this case, Algorithm 3.4 is about 10X faster than TT-SVD. The actual speedup will be a bit lower due to the uncounted operations such as those in Algorithm 3.1. If we increase the density $\sigma$ to 0.01, $f_{\text{TTSVD}}$ will remain the same and $f_{\text{fasttt}}$ will change into

$$f_{\text{fasttt}} \approx f_{\text{SVD}}(3, 120000) + f_{\text{SVD}}(20r, 120000) + f_{\text{SVD}}(15r, 40000)$$
$$+ f_{\text{SVD}}(10r, 4000) + f_{\text{SVD}}(20r, 200)$$
$$\approx (1.08 \times 10^6 + 5.7r^2 \times 10^7)C_{\text{SVD}}$$
$$\approx (5.7 \times 10^9)C_{\text{SVD}},$$

and the speedup drops to 1.4. The actual speedup will be a bit higher because we overestimate $\{\tilde{r}_k\}$ and the uncounted operations become insignificant with the
increasing SVD cost. For matrix-to-MPO applications, the main operation is the TT-decomposition of $\mathbf{M}^{\prime} \in \mathbb{R}^{m_{1}n_{1} \times \cdots \times m_{d}n_{d}}$, so (3.10) also works here. We just need to replace $n_{i}$ with $m_{i}n_{i}$.

We now look at the memory cost of the FastTT algorithm. Before the TT-rounding step, all data are stored in a sparse format. So, the extra memory cost occurs in the TT-rounding which is similar to the TT-SVD algorithm and is of similar size to the cores in the obtained TT.

4. Numerical Experiments. In order to provide empirical proof of the performance of the developed FastTT algorithm, we conduct several numerical experiments. The algorithm is implemented in C++ based on the xerus C++ toolbox [9] and Intel Math Kernel Library\(^2\). The xerus library also contains an implementation of TT-SVD and is at least 2X faster than TT-toolbox\(^3\) in MATLAB. As no C++ implementation of the TT-cross method [14] is available we did not compare TT-cross with FastTT. All experiments were carried out on a x86-64 Linux server with 32 CPU cores and 512G RAM. The desired accuracy tolerance $\varepsilon$ of both TT-SVD and our FastTT algorithms is $10^{-14}$, unless otherwise stated. In all experiments, the CPU time is reported.

4.1. Image/video inpainting. Applications like tensor completion [11] require a fixed-rank TT-approximate of the given tensors. The tensors used in this section are a large color image Dolphin\(^4\) which have been reshaped into a $10 \times 20 \times 20 \times 10 \times 15 \times 20 \times 3$ tensor and a color video Mariano Rivera Ultimate Career Highlights\(^5\) which have been reshaped into a $20 \times 18 \times 20 \times 32 \times 12 \times 12 \times 3$ tensor. Most pixels of the image/video are not observed and are regarded as zeros whereas the observed pixels are chosen randomly. The observation ratio $\sigma$ is the ratio of observed pixels to the total number of pixels. Table 1 shows the results for different specified TT-ranks and observation ratios.

| data | TT-rank | $\sigma$ | time (s) | speedup |
|------|---------|----------|----------|---------|
|      |         |          | TT-SVD   | rTTSVD  | FastTT  |         |
| image | 10      | 0.001    | 32.9     | 24.1    | 3.43    | 9.6X    |
|       | 10      | 0.005    | 32.3     | 23.9    | 10.9    | 3.0X    |
|       | 10      | 0.01     | 32.8     | 26.0    | 14.2    | 2.3X    |
|       | 30      | 0.001    | 42.7     | 38.1    | 12.2    | 3.5X    |
|       | 30      | 0.005    | 42.9     | 33.8    | 20.5    | 2.1X    |
|       | 100     | 0.001    | 67.3     | 91.5    | 23.7    | 2.8X    |
| video | 10      | 0.001    | 66.2     | 56.0    | 10.4    | 6.4X    |
|       | 10      | 0.005    | 66.6     | 60.5    | 26.2    | 2.5X    |
|       | 10      | 0.01     | 66.9     | 62.7    | 33.3    | 2.0X    |
|       | 30      | 0.001    | 103      | 108     | 26.5    | 3.9X    |
|       | 30      | 0.005    | 110      | 94.2    | 47.6    | 2.3X    |
|       | 100     | 0.001    | 232      | 221     | 107     | 2.2X    |

\(^2\)https://software.intel.com/en-us/mkl
\(^3\)https://github.com/oseledets/TT-Toolbox
\(^4\)http://absfreepic.com/absolutely_free_photos/original_photos/dolphin-4000x3000_21859.jpg
\(^5\)https://www.youtube.com/watch?v=UPtDJuIMyhc
It can be seen from Table 1 that our algorithm can greatly speed up the calculation of a TT-approximation when the observation ratio is small. We have also tested the randomized TT-SVD (rTT-SVD) algorithm\(^6\) which also speeds up the calculation in some cases. However, the speedup of the rTT-SVD algorithm is not as great as ours, and in cases where the preset TT-rank is high we observe that the rTT-SVD algorithm is even slower than the TT-SVD algorithm. In addition, the quality of the TT-approximation calculated by the rTT-SVD algorithm is not as good as ours. For example, in the image inpainting task where the TT-rank is 100 and the observation ratio \(\sigma = 0.001\), the mean square error (MSE) of both TT-SVD algorithm and our algorithm is 22.3, while the MSE of rTT-SVD algorithm is 23.5.

For each of the experiments the integer \(p\) was selected automatically by the FLOP estimation in Algorithm 3.6. Now, we validate this FLOP estimation. For the parameters TT-rank = 100, \(\sigma = 0.001\) in the image experiment we run Algorithm 3.4 several times while manually setting different integer \(p\) and plot the CPU time for each \(p\) along with the estimated FLOP count. The results are shown in Figure 5, where we can see that the trend of the two curves is basically consistent. The integer \(p\) selected by Algorithm 3.6 is \(p = 7\), with which the exact CPU time is only slightly more than the best selection at \(p = 6\). Algorithm 3.6 does not always produce the best \(p\), it certainly avoids bad values like \(p = 3\) in this case.

\begin{center}
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5.png}
\caption{The CPU time and estimated FLOPs with (3.10) of the FastTT algorithm for different \(p\) values.}
\end{figure}
\end{center}

4.2. Linear equation in finite difference method. The finite difference method (FDM) is widely used for solving partial differential equations, in which finite differences approximate the partial derivatives. With FDM, a linear equation system with sparse coefficient matrix is solved. We consider simulating a three-dimensional rectangular domain with FDM. The resulted linear equation system can be transformed into the matrix TT format (i.e. MPO) and then solved with an alternating least squares (ALS) method [17].

For a domain partitioned into \(n \times m \times k\) grids, FDM produces a coefficient matrix \(A \in \mathbb{R}^{N \times N}\), where \(N = n \times m \times k\). For example, the sparsity pattern of the coefficient

\[\text{The oversampling parameter of rTT-SVD algorithm is set to 10 in all experiments.}\]
matrix $A$ for FDM with $20 \times 20 \times 20$ grids is shown in Figure 6. Naturally, the $A$ matrix can be regarded as a 6-way tensor $A \in \mathbb{R}^{n \times n \times m \times m \times k \times k}$, which is then converted into an MPO. Since the tensor $A$ is very sparse, replacing the TT-SVD with FastTT will speed up the procedure of computing its TT-decomposition. In this experiment we construct the coefficient matrix with different grid partition, while the coefficients either follow a particular pattern, or are randomly generated. The results for converting the matrix to an MPO are shown in Table 2.

![Fig. 6. The sparsity pattern of the coefficient matrix for FDM with $20 \times 20 \times 20$ grids.](image)

**Table 2.** Experimental results on the coefficient matrices for the FDM with $n \times n \times n$ grids.

| $n$ | coefficients | method   | time(s) | speedup | $\varepsilon_{\text{actual}}$ | TT-ranks** |
|-----|---------------|----------|---------|---------|-----------------------------|-------------|
| 20  | pattern       | TT-SVD   | 43.6    |    -    | $4.0 \times 10^{-16}$      | $r:2,2$     |
|     |               | rTT-SVD  | 1.29    | 34X     | $1.2 \times 10^{-15}$       | $r:2,2$     |
|     |               | FastTT   | 0.788   | 55X     | $9.6 \times 10^{-16}$       | $R:1920; \tilde{r}:58,58; r:2,2$ |
|     | pattern       | TT-SVD   | 690     |    -    | $2.0 \times 10^{-15}$       | $r:2,2$     |
|     |               | rTT-SVD  | 19.3    | 36X     | $1.6 \times 10^{-15}$       | $r:2,2$     |
|     |               | FastTT   | 2.88    | 240X    | $1.1 \times 10^{-15}$       | $R:4380; \tilde{r}:88,88; r:2,2$ |
| 30  | pattern       | TT-SVD   | 53.4    |    -    | $2.5 \times 10^{-15}$       | $r:58,58$   |
|     |               | rTT-SVD  | 23.4    | 2.3X    | $4.8 \times 10^{-15}$       | $r:58,58$   |
|     |               | FastTT   | 1.67    | 32X     | $2.4 \times 10^{-15}$       | $R:1920; \tilde{r}:58,58; r:58,58$ |
| 20  | random        | TT-SVD   | 762     |    -    | $3.4 \times 10^{-15}$       | $r:88,88$   |
|     |               | rTT-SVD  | 67.0    | 11X     | $4.2 \times 10^{-15}$       | $r:88,88$   |
|     |               | FastTT   | 12.4    | 61X     | $3.3 \times 10^{-15}$       | $R:4380; \tilde{r}:88,88; r:88,88$ |
| 30  | random        | TT-SVD   | NA      |    -    | NA                          | NA          |
|     |               | rTT-SVD  | 597     |    -    | $5.0 \times 10^{-15}$       | $r:118,118$ |
|     |               | FastTT   | 57.5    |    -    | $2.6 \times 10^{-15}$       | $R:7840; \tilde{r}:118,118; r:118,118$ |

* $\varepsilon_{\text{actual}} = \| A - B \|_F / \| A \|_F$. The same below.

** R is the number of nonzero subvectors. $\tilde{r}$ is the TT-ranks after deduplication. $r$ is the final TT-ranks after TT-rounding.
As seen from Table 2, FastTT can convert large sparse matrices much faster than the TT-SVD with up to 240X speedup. These experiments also prove that the deduplication procedure can greatly reduce the TT-rank and thus simplify the computation of the TT-rounding procedure. The results of rTTSVD are obtained by setting the TT-ranks same as those obtained by TT-SVD and FastTT. From the result we can see the rTTSVD algorithm is not as fast as FastTT even if we know the proper TT-ranks.

If we set $n = 40$ with random coefficients, the TT-SVD algorithm cannot produce any result in a reasonable time, while FastTT finishes in 57.5 seconds with a resulting TT-rank of $r = 118$.

4.3. Data of road network. A directed/undirected graph with $N$ nodes is equivalent to its adjacency matrix $A \in \mathbb{R}^{N \times N}$, which can also decomposed into an MPO if we properly factorize its order $N = n_1 \times \cdots \times n_d$. This may benefit some data mining applications. In this experiment we use the undirected graph roadNet-PA\(^7\) from SNAP [12]. Since the graph is fairly large, we only take the subgraph of the first $N$ nodes as our data and preprocess its adjacency matrix by performing reverse Cuthill-McKee ordering [5]. Additionally, different desired accuracy tolerances $\varepsilon$ and the actual relative error are tested in this experiment. The truncation parameters in Algorithm 3.3 is either set as $\delta_k = \frac{\varepsilon}{\sqrt{p-k+1}+\sqrt{d-p}} \|A\|_F$ or determined by Algorithm 3.5. The results are shown in Table 3.

**Table 3**
Experimental results on converting the data of roadNet-PA.

| $N$   | $\varepsilon$   | method* | time (s) | speedup | TT-ranks | $\varepsilon_{actual}$ |
|-------|-----------------|---------|----------|---------|----------|-----------------------|
| 20\(^3\) | $1 \times 10^{-14}$ | TT-SVD  | 75.4     | –       | 58, 400  | $3.7 \times 10^{-15}$  |
|       |                 | FastTT  | 14.1     | 5.3X    | 58, 400  | $3.3 \times 10^{-15}$  |
| 20\(^3\) | $5 \times 10^{-1}$  | TT-SVD  | 62.2     | –       | 31, 281  | $4.8 \times 10^{-1}$   |
|       |                 | FastTT  | 11.8     | 5.3X    | 31, 281  | $4.8 \times 10^{-1}$   |
|       |                 | FastTT\(^+\) | 10.4  | 6.0X    | 55, 209  | $5.0 \times 10^{-1}$   |
| 10\(^4\) | $1 \times 10^{-14}$ | TT-SVD  | 833      | –       | 28, 1407, 70 | $3.9 \times 10^{-15}$  |
|       |                 | FastTT  | 23.3     | 34X     | 28, 1407, 70 | $4.3 \times 10^{-15}$  |
| 10\(^4\) | $1 \times 10^{-2}$ | TT-SVD  | 839      | –       | 28, 1390, 70 | $5.5 \times 10^{-3}$   |
|       |                 | FastTT  | 24.4     | 34X     | 28, 1395, 70 | $3.8 \times 10^{-3}$   |
|       |                 | FastTT\(^+\) | 24.2  | 35X     | 28, 1377, 70 | $9.8 \times 10^{-3}$   |

* FastTT: use Algorithm 3.3 for TT-rounding; FastTT\(^+\): use Algorithm 3.5 for TT-rounding.

Again, for sparse graphs our FastTT algorithm is much faster than TT-SVD. Also, the actual relative errors are shown to be less than the given $\varepsilon$. If $\varepsilon$ is small enough, the TT-rank obtained by FastTT is the same as those obtained by TT-SVD. Otherwise Algorithm 3.5 will usually produce lower TT-ranks and higher relative error than Algorithm 3.3 which sets unified truncation parameters.

5. Conclusions. This paper analyzes several state-of-the-art algorithms for the computation of the TT decomposition and proposes a faster TT decomposition algorithm for sparse tensors. We prove the correctness and complexity of the algorithm and demonstrate the advantages and disadvantages of each algorithm.

In the subsequent experiments, we verified the actual performance of each al-

\(^7\)Road network of Pennsylvania. [http://snap.stanford.edu/data/roadNet-PA.html](http://snap.stanford.edu/data/roadNet-PA.html)
algorithm and confirmed our theoretical analysis. The experimental results also show that our proposed FastTT algorithm for sparse tensors is indeed an algorithm with excellent efficiency and versatility. Previous state-of-the-art algorithms were mainly limited by the tensor size whereas our proposed algorithm is mainly limited by the number of non-zero elements. As a result, the TT decomposition can be computed quickly regardless of the number of dimensions. This algorithm therefore is very promising to tackle tensor applications that were previously unimaginable, just like the large-scale use of previous sparse matrix algorithms.

REFERENCES

[1] K. Batselier, Z. Chen, and N. Wong, A tensor network Kalman filter with an application in recursive MIMO Volterra system identification, Automatica, 84 (2017), pp. 17–25.
[2] K. Batselier, Z. Chen, and N. Wong, Tensor network alternating linear scheme for MIMO Volterra system identification, Automatica, 84 (2017), pp. 26–35.
[3] K. Batselier, W. Yu, L. Daniel, and N. Wong, Computing low-rank approximations of large-scale matrices with the tensor network randomized SVD, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 1221–1244.
[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, MIT press, 2009.
[5] A. George and J. W. Liu, Computer Solution of Large Sparse Positive Definite, Prentice Hall Professional Technical Reference, 1981.
[6] N. Halko, P.-G. Martinsson, and J. A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM review, 53 (2011), pp. 217–288.
[7] F. L. Hitchcock, The expression of a tensor or a polyadic as a sum of products, Studies in Applied Mathematics, 6 (1927), pp. 164–189.
[8] B. Huber, R. Schneider, and S. Wolf, A randomized tensor train singular value decomposition, in Compressed Sensing and its Applications, Springer, Cham, 2017, pp. 281–290.
[9] B. Huber and S. Wolf, Xerus - A general purpose tensor library, 2014–2017, https://libxerus.org/.
[10] C. Hubig, I. McCulloch, and U. Schollwöck, Generic construction of efficient matrix product operators, Physical Review B, 95 (2017), p. 035129.
[11] C.-Y. Ko, K. Batselier, W. Yu, and N. Wong, Fast and accurate tensor completion with total variation regularized tensor trains, arXiv preprint arXiv:1804.06128, (2018).
[12] J. Leskovec and A. Krevl, SNAP Datasets: Stanford large network dataset collection, June 2015, http://snap.stanford.edu/data.
[13] H. Liu, L. Daniel, and N. Wong, Model reduction and simulation of nonlinear circuits via tensor decomposition, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 34 (2015), pp. 1059–1069.
[14] I. Oseledets and E. Tyrtyshnikov, TT-cross approximation for multidimensional arrays, Linear Algebra and its Applications, 432 (2010), pp. 70–88.
[15] I. V. Oseledets, Approximation of 2d×2d matrices using tensor decomposition, SIAM Journal on Matrix Analysis and Applications, 31 (2010), pp. 2130–2145.
[16] I. V. Oseledets, Tensor-train decomposition, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317.
[17] I. V. Oseledets and S. Dolgov, Solution of linear systems and matrix inversion in the TT-format, SIAM Journal on Scientific Computing, 34 (2012), pp. A2718–A2739.
[18] L. R. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika, 31 (1966), pp. 279–311.
[19] W. Wang, V. Aggarwal, and S. Aeron, Efficient low rank tensor ring completion, in IEEE International Conference on Computer Vision (ICCV), 2017, pp. 5698–5706.
[20] W. Yu, Y. Gu, and J. Li, Single-pass PCA of large high-dimensional data, in Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017, pp. 3350–3356.
[21] W. Yu, Y. Gu, and Y. Li, Efficient randomized algorithms for the fixed-precision low-rank matrix approximation, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 1339–1359.
[22] Z. Zhang, K. Batselier, H. Liu, L. Daniel, and N. Wong, Tensor computation: A new framework for high-dimensional problems in EDA, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36 (2017), pp. 521–536.