Faster tensor train decomposition for sparse data

Li, Lingjie; Yu, Wenjian; Batselier, Kim

DOI
10.1016/j.cam.2021.113972

Publication date
2022

Document Version
Final published version

Published in
Journal of Computational and Applied Mathematics

Journal of Computational and Applied Mathematics 405 (2022) 113972

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Faster tensor train decomposition for sparse data[☆]

Lingjie Li[a], Wenjian Yu[a,*], Kim Batselier[b]

[a] Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing 100084, China
[b] The Delft Center for Systems and Control, Delft University of Technology, Delft, Netherlands

ARTICLE INFO

Article history:
Received 7 September 2020
Received in revised form 17 August 2021

Keywords:
Tensor train decomposition
Sparse data
TT-rounding
Parallel-vector rounding

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-optimal fast TT decomposition algorithm for large-scale sparse tensors with proven correctness and the upper bound of computational complexity derived. It can also efficiently produce sparse TT with no numerical error and slightly larger TT-ranks on demand. In numerical experiments, we verify that the proposed algorithm can decompose sparse tensors in a much faster speed than the TT-SVD, and have advantages on speed, precision and versatility over the randomized TT-SVD and TT-cross. And, with it we can realize large-scale sparse matrix TT decomposition that was previously unachievable, enabling the tensor decomposition based algorithms to be applied in more scenarios.

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. It is often implemented through the well-known singular value decomposition (SVD).

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top},$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times m}$, $\boldsymbol{U} \in \mathbb{R}^{n\times n}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times m}$, $\boldsymbol{V} \in \mathbb{R}^{m\times m}$. $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices, and $\boldsymbol{\Sigma}$ is a diagonal matrix whose diagonal elements (a.k.a. singular values) σ_i, $(1 \le i \le \min(m, n))$ 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

[☆] This work is partially supported by the National Natural Science Foundation of China (grant number 61872206) and Tsinghua University Initiative Scientific Research Program.
[*] Corresponding author.
E-mail addresses: li-lj18@mails.tsinghua.edu.cn (L. Li), yu-wj@tsinghua.edu.cn (W. Yu), k.batselier@tudelft.nl (K. Batselier).

https://doi.org/10.1016/j.cam.2021.113972
0377-0427/© 2021 Elsevier B.V. All rights reserved.

to tensors, proposing tensor decompositions such as the CP decomposition [1], the Tucker decomposition [2] and the tensor train (TT) decomposition [3]. 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 [4], electronic design automation (EDA) [5–7], system identification [8], large-scale matrix processing [9–11], image/video inpainting [12,13], data mining [14] and machine learning [15–17].

To realize the TT decomposition, the TT-SVD algorithm [3] was proposed. It involves a sequence of SVD computations on reshaped matrices. For a large-scale sparse tensor, the TT-SVD consumes excessive computing time and memory usage. Another method employs "cross approximation" to perform low-rank TT-approximations [18,19], but it still needs too many calculations to find a good representation. Recently, a randomized TT-SVD algorithm [20] was proposed, which incorporates the randomized SVD algorithm [21] 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 fibers, more efficient parallel-vector rounding and revised TT-rounding. The new algorithm, called *FastTT*, produces the same compact TT representation as the TT-SVD algorithm [3] given the same error bound, but exhibits a significant runtime advantage for large sparse data. The algorithm can also skip the revised TT-rounding procedure on demand, resulting in sparse output with no numerical error and much less time consumption at the price of slightly larger TT-ranks. 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 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. In cases where only sparse output is demanded, FastTT can process the adjacency matrix of a sparse undirected graph with 10^6 nodes in only 39.9s. 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 TT-cross and the randomized TT-SVD algorithm [20]. For reproducibility, we have shared the C++ codes of the proposed algorithms and experimental data on https://github.com/lljbash/FastTT.

2. Notations and preliminaries

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

2.1. Tensor

Tensors are a high-dimensional generalization of matrices and vectors. A one-dimensional array $\boldsymbol{a} \in \mathbb{R}^n$ is called a vector, and a two-dimensional array $\boldsymbol{A} \in \mathbb{R}^{n_1 \times n_2}$ is called a matrix. When the dimensionality is extended to $d \geq 3$, the d-dimensional array $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ is called a d-way tensor. The positive integer d is defined as the **order** of the tensor. $(n_1, n_2, \ldots, n_d)$ 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 1 (*Vectorization*)**.** If we rearrange the entries of $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ into a vector $\boldsymbol{b} \in \mathbb{R}^{\prod_{k=1}^{d} n_k}$, where

$$a_{i_1,i_2,\ldots,i_d} = b_{i_d + \sum_{k=1}^{d-1}\left[(i_k-1)\prod_{l=k+1}^{d} n_l\right]},$$

then the vector $\boldsymbol{b}$ is called the vectorization of the tensor $\boldsymbol{\mathcal{A}}$, represented as $\text{vec}(\boldsymbol{\mathcal{A}})$.

Definition 2 (*Reshaping*)**.** Like vectorization, if we rearrange the entries of $\boldsymbol{\mathcal{A}}$ into anther tensor $\boldsymbol{\mathcal{B}}$ satisfying $\text{vec}(\boldsymbol{\mathcal{A}}) = \text{vec}(\boldsymbol{\mathcal{B}})$, then the tensor $\boldsymbol{\mathcal{B}}$ is called the reshaping of $\boldsymbol{\mathcal{A}}$, represented as $\text{reshape}(\boldsymbol{\mathcal{A}}, \mathit{Dims})$, where *Dims* denotes the dimensions of $\boldsymbol{\mathcal{B}}$. In fact, vectorization is a special kind of reshaping.

Definition 3 (*Unfolding [3]*)**.** Unfolding is also a kind of reshaping. If we reshape $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ into a matrix $\boldsymbol{B} \in \mathbb{R}^{m_1 \times m_2}$ where $m_1 = \prod_{j=1}^{k} n_j$, $m_2 = \prod_{j=k+1}^{d} n_j$, then $\boldsymbol{B}$ is called the k-unfolding of $\boldsymbol{\mathcal{A}}$, represented as $\text{unfold}_k(\boldsymbol{\mathcal{A}})$.

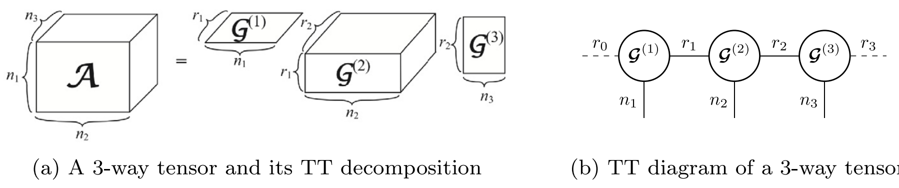

Fig. 1. Graphical illustrations of the tensor train (TT) decomposition, where a 3-way tensor $\mathcal{A}$ is decomposed into two 2-way tensors $\mathcal{G}^{(1)}$, $\mathcal{G}^{(3)}$ and a 3-way tensor $\mathcal{G}^{(2)}$.

Definition 4 (*Contraction*)**.** Contraction is the tensor generalization of matrix product. For two tensors $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_{d_1}}$ and $\mathcal{B} \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_{d_2}}$ satisfying $n_{k_1} = m_{k_2}$, their (k_1, k_2)-contraction $\mathcal{C} = \mathcal{A} \circ_{k_1}^{k_2} \mathcal{B}$ is defined as

$$c_{i_1 \cdots i_{k_1-1} j_1 \cdots j_{k_2-1} j_{k_2+1} \cdots j_{d_2} i_{k_1+1} \cdots i_{d_1}} = \sum_{l=1}^{n_{k_1}} a_{i_1 \cdots i_{k_1-1} l i_{k_1+1} \cdots i_{d_1}} b_{j_1 \cdots j_{k_2-1} l j_{k_2+1} \cdots j_{d_2}},$$

where $\mathcal{C} \in \mathbb{R}^{n_1 \times \cdots \times n_{k_1-1} \times m_1 \times \cdots \times m_{k_2-1} \times m_{k_2+1} \times \cdots \times m_{d_2} \times n_{k_1+1} \times \cdots \times n_{d_1}}$. If k_1 and k_2 are not specified, $\mathcal{A} \circ \mathcal{B}$ means the $(d_1, 1)$-contraction of $\mathcal{A}$ and $\mathcal{B}$.

Definition 5 (*Tensor–Matrix Product*)**.** The k-product of a tensor $\mathcal{A}$ and a matrix $\boldsymbol{B}$ can be defined as tensor contraction if the matrix is treated as a 2-way tensor $\mathcal{B}$.

$$\mathcal{A} \times_k \boldsymbol{B} = \mathcal{A} \circ_k^1 \mathcal{B}.$$

Definition 6 (*Rank-1 Tensor*)**.** A rank-1 d-way tensor can be written as the outer product

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

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

2.3. Tensor train decomposition

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

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

where $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ is the kth 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*.

Fig. 1(b) shows a very convenient graphical representation [8] 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". Fig. 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 $\mathcal{G}_{i_k}^{(k)}$ (actually a vector if $k = 1$ or $k = d$). Then the entries of $\mathcal{A}$ can be computed as

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

The tensor train decomposition can be computed with the TT-SVD algorithm [3], which consists of doing $d-1$ consecutive reshapings and matrix SVD computations. It is described as Algorithm 1. The expression $\text{rank}_\delta(\boldsymbol{C})$ denotes the number of remaining singular values after the δ-truncated SVD. An advantage of TT-SVD is that a quasi-optimal[1] approximation can be obtained with a given error bound and an automatic rank determination [7].

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

$$f_{\text{TTSVD}} \approx \sum_{i=1}^{d-1} \left[f_{\text{SVD}} \left(r_{i-1} n_i, \prod_{j=i+1}^{d} n_j \right) \right], \tag{1}$$

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

$$f_{\text{SVD}}(m, n) = C_{\text{SVD}} \cdot mn \cdot \min(m, n),$$

where C_{SVD} is a constant.

[1] Here, quasi-optimal means that the resulted TT-ranks are as small as possible in the given error bound.

Algorithm 1 TT-SVD [3, p. 2301].

Input: a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$, desired accuracy tolerance ε.
Output: Core tensors $\boldsymbol{\mathcal{G}}^{(1)}, \dots, \boldsymbol{\mathcal{G}}^{(d)}$ of the TT-approximation $\boldsymbol{\mathcal{B}}$ to $\boldsymbol{\mathcal{A}}$ with TT ranks r_k $(k = 0, 1, \cdots, d)$ satisfying

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

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

A big problem with Algorithm 1 is the large computation cost of δ-truncated SVD on large-scale unfolded matrices when the dimensions grow. A possible solution is to replace SVD with more economic decomposition like pseudo-skeleton decomposition [22]. TT-cross [18,19] is a multidimensional generalization of the skeleton decomposition to the tensor case. The algorithm uses a sweep strategy and can produce TT-approximates with given accuracy or maximal ranks. The time complexity of TT-cross depends on d linearly.

As for decomposing large-scale sparse tensors, a simple idea is to employ the truncated SVD algorithm based on Krylov subspace iterative method, e.g. the built-in function `svds` in Matlab. Another common approach is to utilize matrix-free methods like SLEPc [23]. However, both of them requests a truncation rank as input, and thus cannot be directly applied here. Moreover, the sparsity can only be taken advantage of at the first iteration step of Algorithm 1. After that, the matrix of right singular vectors is processed, which is definitely a dense matrix, and thus no more sparsity can be leveraged.

2.4. Rounding

Sometimes one is given tensor data already in the TT-format but with suboptimal TT-ranks. In order to save storage and speed up the following computation, one can reduce the TT-ranks while maintaining accuracy, through a procedure called *rounding*. It is realized with the TT-rounding algorithm [3] (Algorithm 2). The algorithm is based on the same principle as TT-SVD and also produces quasi-optimal TT-ranks with a given error bound. TT-rounding can be of great use in cases where a large tensor is represented in TT-format.

Algorithm 2 TT-rounding [3, p. 2305].

Input: Cores $\boldsymbol{\mathcal{G}}^{(1)}, \dots, \boldsymbol{\mathcal{G}}^{(d)}$ of the TT-format tensor $\boldsymbol{\mathcal{A}}$ with TT-ranks $r_1, \dots, r_{d-1}$, desired accuracy tolerance ε.
Output: Cores $\boldsymbol{\mathcal{G}}^{(1)}, \dots, \boldsymbol{\mathcal{G}}^{(d)}$ of the TT-approximation $\boldsymbol{\mathcal{B}}$ to $\boldsymbol{\mathcal{A}}$ in the TT-format with TT-ranks $r_1, \dots, r_{d-1}$. The computed approximation satisfies

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

```
1: Compute truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}} \|\boldsymbol{\mathcal{A}}\|_F$.
2: for $k = d, \dots, 2$ do
3:     $\boldsymbol{G} :=$ unfold$_1^T(\boldsymbol{\mathcal{G}}^{(k)})$.
4:     Compute economic QR decomposition $\boldsymbol{G} = \boldsymbol{QR}$,   $r_{k-1} :=$ rank$(\boldsymbol{G})$.
5:     $\boldsymbol{\mathcal{G}}^{(k)} :=$ reshape($\boldsymbol{Q^T}, [r_{k-1}, n_k, r_k]$),   $\boldsymbol{\mathcal{G}}^{(k-1)} := \boldsymbol{\mathcal{G}}^{(k-1)} \times_3 \boldsymbol{R}^T$.
6: end for
7: for $k = 1, \dots, d-1$ do
8:     $\boldsymbol{G} :=$ unfold$_2(\boldsymbol{\mathcal{G}}^{(k)})$.
9:     Compute $\delta$-truncated SVD: $\boldsymbol{G} = \boldsymbol{U\Sigma V}^T + \boldsymbol{E}$, $\|\boldsymbol{E}\|_F \leq \delta$,   $r_k :=$ rank$_\delta(\boldsymbol{C})$.
10:    $\boldsymbol{\mathcal{G}}^{(k)} :=$ reshape($\boldsymbol{U}, [r_{k-1}, n_k, r_k]$),   $\boldsymbol{\mathcal{G}}^{(k+1)} := \boldsymbol{\mathcal{G}}^{(k+1)} \times_1 (\boldsymbol{\Sigma V}^T)$.
11: end for
12: $\boldsymbol{\mathcal{G}}^{(d)} := \boldsymbol{\mathcal{C}}$
13: Return tensor $\boldsymbol{\mathcal{B}}$ in TT-format with cores $\boldsymbol{\mathcal{G}}^{(1)}, \dots, \boldsymbol{\mathcal{G}}^{(d)}$.
```

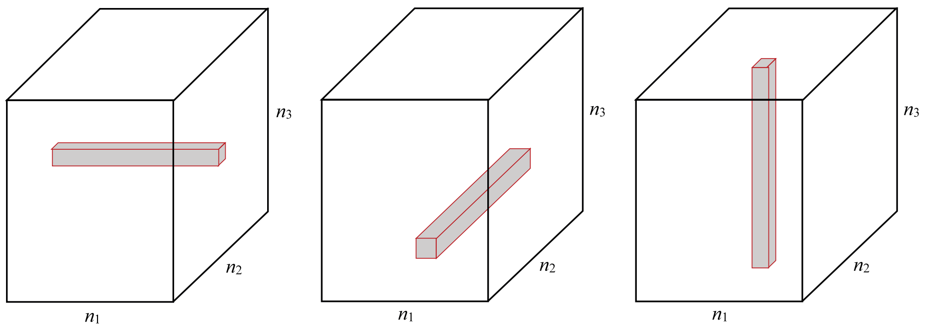

Fig. 2. A 1-fiber, a 2-fiber and a 3-fiber of tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

Parallel-vector rounding [24] is another rounding method which replaces the truncated SVD with Deparallelisation (Algorithm 3). It removes the paralleled columns of an $a \times b$ matrix in $O(ab\alpha)$ time, where α is the number of non-parallel columns. Parallel-vector rounding is lossless, runs much faster than TT-rounding and can preserve the sparsity of TT. However, it usually cannot reduce the TT-ranks much; its effectiveness highly depends on the parallelism in TT-cores. Therefore, the parallel-vector rounding is suitable for a constructed sparse TT, rather than the construction of a TT.

Algorithm 3 Deparallelisation [24, Appendix B].

```
Input: Matrix $\boldsymbol{M} \in \mathbb{R}^{a \times b}$.
Output: Matrix $\boldsymbol{N} \in \mathbb{R}^{a \times \beta}$, $\boldsymbol{T} \in \mathbb{R}^{\beta \times b}$ s.t. $\boldsymbol{M} = \tilde{\boldsymbol{M}} \times \boldsymbol{T}$ and $\boldsymbol{N}$ has at most as many columns as $\boldsymbol{M}$ and no two columns
    which are parallel to each other.
1: Let $K$ be the set of kept columns, empty initially.
2: Let $\boldsymbol{T}$ be the dynamically-resized transfer matrix.
3: for every column index $j \in [1, b]$ do
4:   for every kept index $i \in [1, |K|]$ do
5:     if the $j$-th column $\boldsymbol{M}_{:j}$ is parallel to column $K_i$ then
6:       Set $T_{i,j}$ to the prefactor between the two columns.
7:     else
8:       add $\boldsymbol{M}_{:j}$ to $K$, set $T_{|K|,j} = 1$.
9:     end if
10:  end for
11: end for
12: Construct $\boldsymbol{N}$ by horizontally concatenating the columns stored in $K$.
13: Return $\boldsymbol{N}$ and $\boldsymbol{T}$.
```

3. Faster tensor train decomposition of sparse tensor

The TT-SVD algorithm does not take advantage of the possible sparsity of data since the δ-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-fibers

For a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and a given integer p that satisfies $1 \leq p \leq d$, we define a p-fiber of a tensor as a fiber of the tensor in the direction $\boldsymbol{e}_p$. Fig. 2 shows a 1-fiber, a 2-fiber and a 3-fiber of a 3-d tensor. We can specify a p-fiber of $\mathcal{A}$ by fixing the indices except the pth dimension $\boldsymbol{i}_p = (i_1, \ldots, i_{p-1}, i_{p+1}, \ldots, i_d)$,

$$\boldsymbol{v}_p(\boldsymbol{i}_p) := \mathcal{A}(i_1, \ldots, i_{p-1}, :, i_{p+1}, \ldots, i_d). \tag{2}$$

Then a tensor $\mathcal{A}_1$ with only one non-zero p-fiber $\boldsymbol{v}_p(\boldsymbol{i}_p)$ can be easily formed as the outer product of the p-fiber and $d-1$ standard basis vector and is thus a rank-1 tensor,

$$\mathcal{A}_1 = \boldsymbol{e}_{i_1} \circ \cdots \circ \boldsymbol{e}_{i_{p-1}} \circ \boldsymbol{v}_p(\boldsymbol{i}_p) \circ \boldsymbol{e}_{i_{p+1}} \circ \cdots \circ \boldsymbol{e}_{i_d}. \tag{3}$$

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

$$\boldsymbol{\mathcal{A}} = \sum_{(i_1,\ldots,i_{p-1},i_{p+1},\ldots,i_d)\in S_p} \boldsymbol{e}_{i_1} \circ \cdots \circ \boldsymbol{e}_{i_{p-1}} \circ \boldsymbol{v}_{i_1,\ldots,i_{p-1},i_{p+1},\ldots,i_d} \circ \boldsymbol{e}_{i_{p+1}} \circ \cdots \circ \boldsymbol{e}_{i_d}, \tag{4}$$

where $\boldsymbol{e}_{i_k} \in \mathbb{R}^{n_k}$ is the standard basis vector. Next, we are going to construct a tensor train based on this representation.

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

$$\boldsymbol{v}_1 \circ \boldsymbol{v}_2 \circ \ldots \circ \boldsymbol{v}_d = \boldsymbol{\mathcal{V}}^{(1)} \circ \boldsymbol{\mathcal{V}}^{(2)} \circ \ldots \circ \boldsymbol{\mathcal{V}}^{(d)}, \tag{5}$$

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

Lemma 2 ([3, p. 2308]). *Suppose we have two tensors $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ in the TT format,*

$$a_{i_1 i_2\cdots i_d} = \boldsymbol{\mathcal{A}}^{(1)}_{i_1}\boldsymbol{\mathcal{A}}^{(2)}_{i_2}\cdots\boldsymbol{\mathcal{A}}^{(d)}_{i_d},$$
$$b_{i_1 i_2\cdots i_d} = \boldsymbol{\mathcal{B}}^{(1)}_{i_1}\boldsymbol{\mathcal{B}}^{(2)}_{i_2}\cdots\boldsymbol{\mathcal{B}}^{(d)}_{i_d}.$$

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

$$\begin{aligned} \boldsymbol{\mathcal{C}}^{(k)}_{i_k} &= \begin{bmatrix} \boldsymbol{\mathcal{A}}^{(k)}_{i_k} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{\mathcal{B}}^{(k)}_{i_k} \end{bmatrix}, \quad k = 2,\ldots,d-1, \\ \boldsymbol{\mathcal{C}}^{(1)}_{i_1} &= \begin{bmatrix} \boldsymbol{\mathcal{A}}^{(1)}_{i_1} & \boldsymbol{\mathcal{B}}^{(1)}_{i_1} \end{bmatrix}, \quad \boldsymbol{\mathcal{C}}^{(d)}_{i_d} = \begin{bmatrix} \boldsymbol{\mathcal{A}}^{(d)}_{i_d} \\ \boldsymbol{\mathcal{B}}^{(d)}_{i_d} \end{bmatrix}, \end{aligned} \tag{6}$$

where $\boldsymbol{O}$ denotes a zero matrix of appropriate dimensions.

The proof of Lemmas 1 and 2 can be easily derived from Definitions 4–6 and the definition of the tensor train decomposition.

Based on (4) and Lemmas 1 and 2, we have the following theorem.

Theorem 3. *A sparse tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ can be transformed into an equivalent tensor train with TT rank R, where R is the number of nonzero p-fiber in $\boldsymbol{\mathcal{A}}$ $(1 \le p \le d)$. If $p \ne 1$ or d,*

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{P}}^{(1)} \circ \ldots \circ \boldsymbol{\mathcal{P}}^{(p-1)} \circ \boldsymbol{\mathcal{V}} \circ \boldsymbol{\mathcal{P}}^{(p+1)} \circ \ldots \circ \boldsymbol{\mathcal{P}}^{(d)}, \tag{7}$$

where $\boldsymbol{\mathcal{P}}^{(k)} \in \{0,1\}^{R\times n_j\times R}$, $(2 \le k \le d, k \ne p)$, $\boldsymbol{\mathcal{P}}^{(1)} \in \{0,1\}^{1\times n_1\times R}$, $\boldsymbol{\mathcal{P}}^{(d)} \in \{0,1\}^{R\times n_d\times 1}$, and $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{R\times n_p\times R}$. Similar expressions hold for the situations with $p = 1$ or d.

The TT cores $\boldsymbol{\mathcal{P}}^{(k)}$ and $\boldsymbol{\mathcal{V}}$ in Theorem 3 are sparse tensors, whose nonzero distributions are illustrated in Fig. 3. Each horizontal bar depicted in Fig. 3 is a standard basis vector $\boldsymbol{e}_{i_k}$ for $\boldsymbol{\mathcal{P}}^{(k)}$ or a p-fiber $\boldsymbol{v}$ for $\boldsymbol{\mathcal{V}}$. The derived matrices ($\boldsymbol{\mathcal{P}}^{(k)}_{i_k}$ and $\boldsymbol{\mathcal{V}}_{i_p}$) from these TT cores are all diagonal matrices. Furthermore, each of the $\boldsymbol{\mathcal{P}}^{(k)}$ cores is very sparse, as the nonzero elements consist of only R 1's.

3.2. More efficient parallel-vector rounding

From Fig. 3, it is obvious that $\boldsymbol{\mathcal{P}}^{(k)}$ are very sparse and the elements are arranged regularly. Hence parallel-vector rounding should be very effective on the TT obtained in Theorem 3. In order to maximize the effect of Deparallelisation, we modify the original algorithm so that the decomposition on the less regular core $\boldsymbol{\mathcal{V}}$ is avoided. By combining this modified algorithm with Theorem 3, we obtain a lossless sparse tensor to TT conversion algorithm, described as Algorithm 4, where *Depar* refers to the Deparallelisation algorithm.

The correctness of Algorithm 4 is due to Theorem 3 and the associative property of matrix multiplications. The graphical representations of the decomposition forms during the algorithm execution are shown in Fig. 4 for a 4-way TT.

The original Deparallelisation algorithm (Algorithm 3) does not consider the special pattern of the core tensors. An important observation from Fig. 3 is that each $\boldsymbol{\mathcal{P}}^{(k)}$ obtained by Theorem 3 is consisted of R "diagonally" 2-fiber with only one "1". This means all unfoldings $\text{unfold}_2(\boldsymbol{\mathcal{P}}^{(k)})$ for $k < p$ and $\text{unfold}_1^T(\boldsymbol{\mathcal{P}}^{(k)})$ for $k > p$ are so-called *quasi-permutation matrices*.

Definition 7 (*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. A quasi-permutation matrix $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ can be represented as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{e}_{i_1} & \boldsymbol{e}_{i_2} & \cdots & \boldsymbol{e}_{i_m} \end{bmatrix}, \tag{8}$$

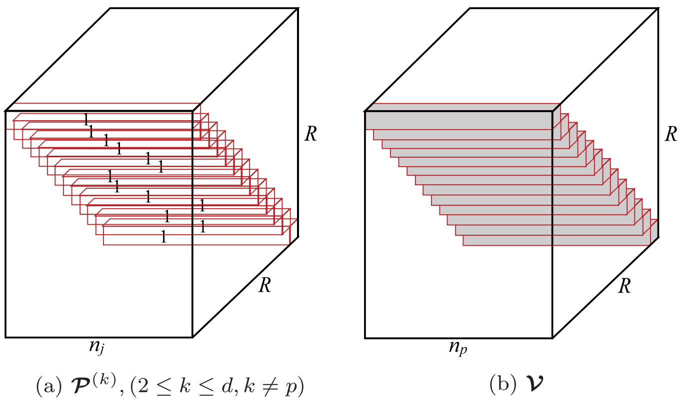

Fig. 3. The nonzero distributions of $\mathcal{P}^{(k)}, (2 \le k \le d, k \neq p)$ and $\mathcal{V}$.

Algorithm 4 Sparse TT conversion with modified parallel-vector rounding.

```
Input: A sparse tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$, an integer $p$ $(1 \le p \le d)$.
Output: Core tensors $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(d)}$ of TT-format tensor $\mathcal{B}$ which is equivalent to $\mathcal{A}$ with TT ranks $\tilde{r}_k$ $(k = 0, 1, \cdots, d)$.
1: Initialize empty cores $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(d)}$ for TT-format tensor $\mathcal{B}$.
2: for every $\boldsymbol{v} \in$ {all $R$ nonzero $p$-fibers of $\mathcal{A}$} do
3:     Determine $(d-1)$ $\boldsymbol{e}_i$ vectors in (4).
4:     Construct rank-1 TT $\mathcal{T}$ with $\boldsymbol{v}$ and $\boldsymbol{e}$ vectors as Lemma 1.
5:     $\mathcal{B} := \mathcal{B} + \mathcal{T}$, which means $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(d)}$ are update with (6).
6: end for
7: $\tilde{r}_0 := 1$.
8: for $k = 1, \dots, p-1$ do
9:     $[\boldsymbol{N}, \boldsymbol{T}] :=$ Depar(unfold$_2(\mathcal{G}^{(k)})$), where $\boldsymbol{N} \in \mathbb{R}^{\tilde{r}_{k-1} n_k \times \tilde{r}_k}$, $\boldsymbol{T} \in \mathbb{R}^{\tilde{r}_k \times R}$.
10:    $\mathcal{G}^{(k)} :=$ reshape$(\boldsymbol{N}, [\tilde{r}_{k-1}, n_k, \tilde{r}_k])$.
11:    $\mathcal{G}^{(k+1)} := \mathcal{G}^{(k+1)} \times_1 \boldsymbol{T}^T$.
12: end for
13: $\tilde{r}_d := 1$.
14: for $k = d, \dots, p+1$ do
15:    $[\boldsymbol{N}, \boldsymbol{T}] :=$ Depar(unfold$_1^T(\mathcal{G}^{(k)})$), where $\boldsymbol{N} \in \mathbb{R}^{\tilde{r}_k n_k \times \tilde{r}_{k-1}}$, $\boldsymbol{T} \in \mathbb{R}^{\tilde{r}_{k-1} \times R}$.
16:    $\mathcal{G}^{(k)} :=$ reshape$(\boldsymbol{N}^T, [\tilde{r}_{k-1}, n_k, \tilde{r}_k])$.
17:    $\mathcal{G}^{(k-1)} := \mathcal{G}^{(k-1)} \times_3 \boldsymbol{T}^T$.
18: end for
19: Return tensor $\mathcal{B}$ in TT-format with cores $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(d)}$.
```

where e_k denotes a standard basis vector in the n-dimensional Euclidean space $\mathbb{R}^n$ with a 1 in the kth coordinate and 0's elsewhere.

Example 1. Obviously, a permutation or identity matrix belongs to the class of quasi-permutation matrices.

Corollary 4. *For Algorithm 3, if the input matrix* $\boldsymbol{M}$ *is a quasi-permutation matrix, matrix* $\boldsymbol{N}$ *and the transfer matrix* $\boldsymbol{T}$ *will also be quasi-permutation matrices.*

It turns out that this property of $\mathcal{P}^{(k)}$ is maintained throughout the whole process of parallel-vector rounding, which will enable a more efficient implementation of Deparallelisation.

Theorem 5. *In Algorithm 4, each input matrix of the function **Depar** is a quasi-permutation matrix.*

Proof. $\mathcal{G}^{(1)}$ is definitely a quasi-permutation matrix according to (5) and (6). For $2 \le k < p$, $\mathcal{G}^{(k)}$ changes twice during Algorithm 4 — once in iteration $k-1$ and once in iteration k. Like in Fig. 4, we use $\mathcal{P}^{(k)}$, $\dot{\mathcal{P}}^{(k)}$ and $\ddot{\mathcal{P}}^{(k)}$ to denote the three stages of $\mathcal{G}^{(k)}$. The input matrix of *Depar* in iteration k is unfold$_2(\dot{\mathcal{P}}^{(k)})$. $\dot{\mathcal{P}}^{(k)}$ is computed as $\mathcal{P}^{(k)} \times_1 \boldsymbol{T}^T$ in iteration $k-1$, which is equivalent to

$$\text{unfold}_1(\dot{\mathcal{P}}^{(k)}) = \boldsymbol{T} \times \text{unfold}_1(\mathcal{P}^{(k)}). \tag{9}$$

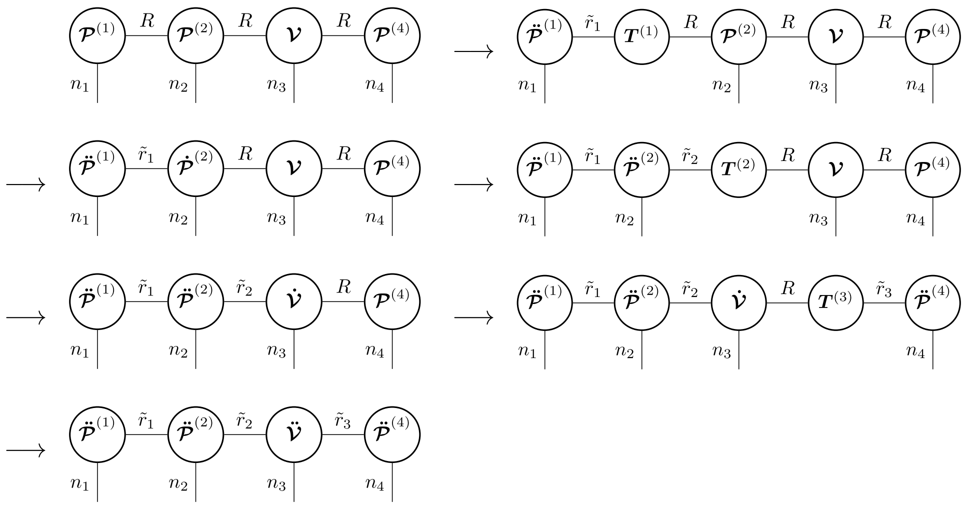
$\mathcal{P}^{(1)}$ R $\mathcal{P}^{(2)}$ R $\mathcal{V}$ R $\mathcal{P}^{(4)}$ n_1 n_2 n_3 n_4
$\ddot{\mathcal{P}}^{(1)}$ $\tilde{r}_1$ $\boldsymbol{T}^{(1)}$
$\dot{\mathcal{P}}^{(2)}$ $\ddot{\mathcal{P}}^{(2)}$ $\tilde{r}_2$ $\boldsymbol{T}^{(2)}$
$\dot{\mathcal{V}}$ $\boldsymbol{T}^{(3)}$ $\tilde{r}_3$ $\ddot{\mathcal{P}}^{(4)}$ $\ddot{\mathcal{V}}$

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

We can deduce from Fig. 3 that unfold$_1(\mathcal{P}^{(k)})$ has the following structure

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

where $\forall j = 1, 2, \ldots, R$, vector $[x_{j1}\ x_{j2}\ \cdots\ x_{jn_k}]^T$ is a particular standard basis vector. Let $\boldsymbol{T} = [\boldsymbol{e}_{i_1}\ \boldsymbol{e}_{i_2}\ \cdots\ \boldsymbol{e}_{i_R}]$ according to Corollary 4. Then unfold$_1(\dot{\mathcal{P}}^{(k)})$ will have the structure

$$\begin{bmatrix} x_{11}\boldsymbol{e}_{i_1} & x_{21}\boldsymbol{e}_{i_2} & \cdots & x_{R1}\boldsymbol{e}_{i_R} & \cdots & x_{1n_d}\boldsymbol{e}_{i_1} & x_{2n_d}\boldsymbol{e}_{i_2} & \cdots & x_{Rn_k}\boldsymbol{e}_{i_R} \end{bmatrix},$$

and thus the structure of unfold$_2(\dot{\mathcal{P}}^{(k)})$ will be

$$\begin{bmatrix} x_{11}\boldsymbol{e}_{i_1} & x_{21}\boldsymbol{e}_{i_2} & \cdots & x_{R1}\boldsymbol{e}_{i_R} \\ x_{12}\boldsymbol{e}_{i_1} & x_{22}\boldsymbol{e}_{i_2} & \cdots & x_{R2}\boldsymbol{e}_{i_R} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1n_k}\boldsymbol{e}_{i_1} & x_{2n_k}\boldsymbol{e}_{i_2} & \cdots & x_{Rn_k}\boldsymbol{e}_{i_R} \end{bmatrix}.$$

From this it follows that unfold$_2(\dot{\mathcal{P}}^{(k)})$ 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 Deparallelisation for a quasi-permutation matrix. Our aim is to express a matrix $\boldsymbol{M}$ as the product of two smaller matrices: $\boldsymbol{M} = \boldsymbol{NT}$. For a quasi-permutation matrix, we need to remove the duplicate columns in $\boldsymbol{M}$. As shown in Corollary 4, the result $\boldsymbol{T}$ itself is a quasi-permutation matrix. Therefore, $\boldsymbol{N} = \boldsymbol{I}$ and $\boldsymbol{T} = \boldsymbol{M}$, where $\boldsymbol{I}$ is an identity matrix, can be regarded as the result of performing Deparallelisation on a quasi-permutation matrix, except that the duplicate columns in $\boldsymbol{M}$ have not yet been removed. What remains to be done is the removal of zero rows of $\boldsymbol{T}$ and the corresponding columns in $\boldsymbol{N}$. This is described as Algorithm 5.

Algorithm 5 Deparallelisation for a quasi-permutation matrix.

```
Input: A quasi-permutation matrix $\boldsymbol{M} \in \mathbb{R}^{n_1 \times n_2}$.
Output: Matrices $\boldsymbol{N} \in \mathbb{R}^{n_1 \times \beta}$, $\boldsymbol{T} \in \mathbb{R}^{\beta \times n_2}$ so that $\boldsymbol{M} = \boldsymbol{NT}$, and $\boldsymbol{N}$ is also a quasi-permutation matrix, $\boldsymbol{T}$ contains all
    nonzero rows of $\boldsymbol{M}$.
1: Let $\boldsymbol{T} \in \mathbb{R}^{\beta \times n_2}$ contains all nonzero rows of $\boldsymbol{M}$, where $\beta$ is the number of nonzero rows.
2: Let $\boldsymbol{N} \in \mathbb{R}^{n_1 \times \beta}$ be a zero-initialized matrix.
3: for $i = 1, 2, \cdots, \beta$ do
4:   $\boldsymbol{N}(j, i) := 1$, where $j$ is the row index of $\boldsymbol{T}(i, :)$ in the original matrix $\boldsymbol{M}$.
5: end for
6: Return $\boldsymbol{N}$ and $\boldsymbol{T}$.
```

For a quasi-permutation matrix, each column can be represented by the position of 1 in it. Thus, Algorithm 5 has a time complexity of $O(n_1 + n_2)$, where n_1 and n_2 are the dimensions of $\boldsymbol{M}$. It can be executed much more efficiently than a general Deparallelisation algorithm. From Algorithm 5 we can also observe, that the resulting matrix size β must be no more than n_1, even if $n_2 \gg n_1$.

According to Theorem 5 and the above analysis, with Algorithm 4 the TT ranks will be reduced to $\tilde{\boldsymbol{r}}$ satisfying the following upper bounds

$$\tilde{r}_k \le \bar{r}_k = \begin{cases} \min\left(R, \prod_{i=1}^{k} n_i\right) & \text{if } 1 \le k < p, \\ \min\left(R, \prod_{i=k+1}^{d} n_i\right) & \text{if } p \le k < d. \end{cases} \tag{10}$$

where R is the number of nonzero p-fibers in the original tensor $\mathcal{A}$. From Fig. 3, it is obvious that the constructed TT contains $Z + (d-1)R$ nonzeros before parallel-vector rounding, where Z is the number of nonzeros in the original tensor $\mathcal{A}$. After the parallel-vector rounding, the number of nonzeros in the TT representation should be reduced to $Z + \sum_{k=1}^{d-1} \tilde{r}_k$.

3.3. More efficient TT-rounding and the FastTT algorithm

Algorithm 4 can already provide rank-reduced TT for sparse tensors while keeping the sparsity and with no precision loss. However, if lower ranks are desired, we can further apply TT-rounding on the TT. This could be useful for applications which do not care about the sparsity. Based on the property of TT obtained in Algorithm 4, we modified the original Algorithm 2 in order to make it more efficient, described as Algorithm 6. Instead of performing a right-to-left QR-sweep and then a left-to-right SVD-sweep, we perform 2 middle-to-edge SVD-sweep from core p. The QR-sweep in our algorithm is proved to be extremely fast due to the orthogonality obtained in Algorithm 4 and former SVD, and that is why our algorithm is more efficient. The truncation parameters in Algorithm 6 is set as (11) to meet the error bound,

$$\delta_k := \frac{\varepsilon}{\sqrt{p-1} + \sqrt{d-p}} \|\mathcal{A}\|_F, \quad k = 1 \dots d-1. \tag{11}$$

The correctness of Algorithm 6 is explained as follows.

Lemma 6. *A quasi-permutation matrix with no duplicate columns is an orthonormal matrix.*

Lemma 6 can be easily proved based by Definition 7 and the definition of an orthonormal matrix.

In Steps 9 and 15 of Algorithm 4, the duplicate columns in the input quasi-permutation matrix (according to Theorem 5) are removed. Then, based on Lemma 6, we have the following statement.

Corollary 7. *Suppose the TT cores $\mathcal{G}^{(k)}$ are obtained with Algorithm 4. Then the matrices* $\text{unfold}_2(\mathcal{G}^{(k)})$, $k < p$ *and* $\text{unfold}_1^T(\mathcal{G}^{(k)})$, $k > p$ *are all orthonormal matrices.*

Lemma 8. *Suppose $\mathcal{U}^{(i)}, i = 1, \dots, d$ are the cores of a tensor train. If matrix* $\text{unfold}_2(\mathcal{U}^{(i)})$ *is an orthonormal matrix for all $i = 1, \dots, k$ $(1 \le k \le d)$, then the matrix* $\text{unfold}_j(\mathcal{U}^{(1)} \circ \cdots \circ \mathcal{U}^{(j)})$ *is an orthonormal matrix for all $j = 1, \dots, k$.*

The proof of Lemma 8 can be found in [14, Appendix B]. We can now derive the following theorem.

Theorem 9 (*Correctness of Algorithm 6*). *The approximation $\mathcal{B}$ obtained in Algorithm 6 always satisfies $\|\mathcal{A} - \mathcal{B}\|_F \le \varepsilon \|\mathcal{A}\|_F$.*

Proof. For simplicity, we let

- $\mathcal{C}$ denote the TT-format tensor after first SVD-sweep,
- $\mathcal{D}$ denote the TT-format tensor before second SVD-sweep,
- $\mathcal{C}_i$ denote the TT-format tensor after the $k = i$ iteration in first SVD-sweep,
- $\mathcal{A}^{(i\dots j)}$ denote the contraction of ith to jth core $\mathcal{G}^{(i)} \circ \cdots \circ \mathcal{G}^{(j)}$ of a TT-format tensor $\mathcal{A}$.

It is obvious that $\mathcal{C} = \mathcal{D}$, hence

$$\|\mathcal{A} - \mathcal{B}\|_F \le \|\mathcal{A} - \mathcal{C}\|_F + \|\mathcal{D} - \mathcal{B}\|_F.$$

Based on the observation that $\mathcal{A}^{(1\dots p-1)} = \mathcal{C}^{(1\dots p-1)}$, we let $\mathcal{A} = \mathcal{A}^{(1\dots p-1)} \circ \mathcal{A}^{(p\dots d)}$ and $\mathcal{C} = \mathcal{A}^{(1\dots p-1)} \circ \mathcal{C}^{(p\dots d)}$. According to Corollary 7 and Lemma 8, $\text{unfold}_{p-1}(\mathcal{A}^{(1\dots p-1)})$ is an orthonormal matrix. Thus

$$\|\mathcal{A} - \mathcal{C}\|_F = \|\mathcal{A}^{(p\dots d)} - \mathcal{C}^{(p\dots d)}\|_F.$$

Let us concentrate on the first iteration ($k = p$) of first SVD-sweep. The truncated-SVD can be rewritten as $\mathcal{A}^{(p)} = \mathcal{C}_1^{(p)} \times_3 \boldsymbol{\Sigma} \boldsymbol{V}^T + \mathcal{E}_p$, where $\|\mathcal{E}_p\|_F \le \delta_p$ and $\mathcal{C}_1^{(p)} \circ_3^3 \mathcal{E}_p = \boldsymbol{0}$. From line 7 we know $\mathcal{C}_1^{(p+1\dots d)} = \mathcal{A}^{(p+1\dots d)} \times_1 \boldsymbol{V} \boldsymbol{\Sigma}$.

Algorithm 6 More efficient TT-rounding for the sparse TT conversion.

```
Input: Cores $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(d)}$ of the TT-format tensor $\mathcal{A}$ with TT-ranks $r_1, \dots, r_{d-1}$, desired accuracy tolerance $\varepsilon$, an integer $p$ $(1 \le p \le d)$.
Output: Cores $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(d)}$ of the TT-approximation $\mathcal{B}$ to $\mathcal{A}$ in the TT-format with TT-ranks $r_1, \dots, r_{d-1}$. The computed approximation satisfies
    $||\mathcal{A} - \mathcal{B}||_F \le \varepsilon ||\mathcal{A}||_F$.

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

Thus,

$$\begin{aligned}
\|\mathcal{A}^{(p\dots d)} - \mathcal{C}^{(p\dots d)}\|_F^2 &= \|\mathcal{A}^{(p)} \circ \mathcal{A}^{(p+1\dots d)} - \mathcal{C}_1^{(p)} \circ \mathcal{C}^{(p+1\dots d)}\|_F^2 \\
&= \|(\mathcal{C}_1^{(p)} \times_3 \boldsymbol{\Sigma}\boldsymbol{V}^T + \boldsymbol{\mathcal{E}}_p) \circ \mathcal{A}^{(p+1\dots d)} - \mathcal{C}_1^{(p)} \circ \mathcal{C}^{(p+1\dots d)}\|_F^2 \\
&= \|\mathcal{C}_1^{(p)} \circ \mathcal{C}_1^{(p+1\dots d)} + \boldsymbol{\mathcal{E}}_p \circ \mathcal{A}^{(p+1\dots d)} - \mathcal{C}_1^{(p)} \circ \mathcal{C}^{(p+1\dots d)}\|_F^2 \\
&= \|\boldsymbol{\mathcal{E}}_p \circ \mathcal{A}^{(p+1\dots d)}\|_F^2 + \|\mathcal{C}_1^{(p)} \circ (\mathcal{C}_1^{(p+1\dots d)} - \mathcal{C}^{(p+1\dots d)})\|_F^2 \\
&= \delta_p^2 + \|\mathcal{C}_1^{(p+1\dots d)} - \mathcal{C}^{(p+1\dots d)}\|_F^2
\end{aligned}$$

Proceeding by induction, we have

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

Similarly we have

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

According to (11),

$$\|\mathcal{A} - \mathcal{B}\|_F \le \sqrt{\sum_{k=p}^{d-1} \delta_k^2} + \sqrt{\sum_{k=2}^{p} \delta_{k-1}^2} \le \varepsilon \|\mathcal{A}\|_F. \quad \square \tag{12}$$

Now, we are ready to describe the whole algorithm for the conversion of a sparse tensor into a TT, presented as Algorithm 7.

Algorithm 7 Tensor train decomposition of sparse tensor (FastTT)

```
Input: A sparse tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, desired accuracy tolerance $\varepsilon$, an integer $p$ $(1 \leq p \leq d)$.
Output: Cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(d)}$ of the TT-approximation $\mathcal{B}$ to $\mathcal{A}$ in the TT-format with TT-ranks $r_k$. The computed
    approximation satisfies
        $||\mathcal{A} - \mathcal{B}||_F \leq \varepsilon ||\mathcal{A}||_F$.
1: Obtain $\mathcal{B}$ in the TT-format with cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(d)}$ by Algorithm 4, where Depar is implemented as Algorithm 3.
2: Reduce the TT ranks of $\mathcal{B}$ with Algorithm 6, or skip this step if sparse output is demanded.
3: Return the TT-approximation $\mathcal{B}$.
```

It should be pointed out that if the accuracy tolerance ε is set to zero,[2] the obtained TT ranks with Algorithm 7 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 $\boldsymbol{C} = \text{unfold}_1(\mathcal{A})$. $\mathcal{A}$ can also represented as the contraction of the TT cores obtained by Algorithm 4.

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

Thus,

$$\boldsymbol{C} = \text{unfold}_2(\mathcal{G}^{(1)})\, \text{unfold}_1(\mathcal{G}^{(2)} \circ \cdots \circ \mathcal{G}^{(d)}).$$

According to Corollary 7 and Lemma 8, $\boldsymbol{L} = \text{unfold}_1^T(\mathcal{G}^{(2)} \cdots \mathcal{G}^{(d)})$ is an orthonormal matrix, i.e. $\boldsymbol{L}^T\boldsymbol{L} = \boldsymbol{I}$.

$$\boldsymbol{C}\boldsymbol{C}^T = \text{unfold}_2(\mathcal{G}^{(1)})\boldsymbol{L}^T\boldsymbol{L}\, \text{unfold}_2^T(\mathcal{G}^{(1)}) = \text{unfold}_2(\mathcal{G}^{(1)})\, \text{unfold}_2^T(\mathcal{G}^{(1)}).$$

This means matrix $\text{unfold}_2(\mathcal{G}^{(1)})$ has the same singular values as $\boldsymbol{C}$. For Algorithm 1, r_1 equals $\text{rank}_\delta(\boldsymbol{C})$, while the r_1 obtained with Algorithm 7 is $\text{rank}_{\delta_1}(\text{unfold}_2(\mathcal{G}^{(1)}))$ (see lines 4 and 5 of Algorithm 6). 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 7 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 6 to fit this scenario. Specifically, the desired accuracy tolerance ε is not needed and thus substituted with the desired TT-ranks. The truncation parameters δ_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 [13].

Some other applications require matrix–vector multiplications, which are convenient if both the matrix and the vector are in TT-format (as shown in Fig. 5). A vector $\boldsymbol{v} \in \mathbb{R}^N$ can be transformed into TT-format if we first reshape it into a tensor $\mathcal{V} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, where $N = n_1 \cdots n_d$, and then decompose it into a TT. A "matrix in TT-format"[3, pp. 2311–2313], also known as a matrix product operator (MPO), is similar but more complicated. The elements of matrix $\boldsymbol{M} \in \mathbb{R}^{M \times N}$ are rearranged into a tensor $\mathcal{M} \in \mathbb{R}^{m_1 \times n_1 \times \cdots \times m_d \times n_d}$, where $M = m_1 \cdots m_d$, $N = n_1 \cdots n_d$. The cores $\mathcal{M}^{(i)} (i = 1, \ldots, d)$ of the MPO satisfy

$$\mathcal{M}(i_1, j_1, \ldots, i_d, j_d) = \mathcal{M}^{(1)}(:, i_1, j_1, :) \cdots \mathcal{M}^{(d)}(:, i_d, j_d, :),$$

where $\mathcal{M}^{(i)} \in \mathbb{R}^{r_{i-1} \times m_i \times n_i \times r_i} (i = 1, \ldots, d)$, $r_0 = r_d = 1$. The matrix-to-MPO algorithm is basically computing a TT-decomposition of the d-way tensor $\mathcal{M}' \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_d n_d}$, along with a few necessary reshapings, which can also be done with Algorithm 7.

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 δ_k, which implies that if the truncation parameters are set statically at the beginning with (11), some of the desired accuracy tolerant ε 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 8. In each step of the truncated SVD, an expected

[2] In practice, ε is usually set to a small value like 10^{-14} due to the inevitable round-off error.

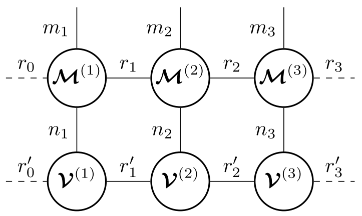

Fig. 5. Diagram of matrix–vector-multiplication in the TT-format.

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 ε.

Algorithm 8 The revised TT-rounding for the sparse TT conversion.

```
Input: (same as Algorithm 6)
Output: (same as Algorithm 6)
1: $\delta_{\text{right}} := \frac{\sqrt{d-p}}{\sqrt{d-p}+\sqrt{p-1}}\varepsilon\|\mathcal{A}\|_F,\ \delta_{\text{left}} := \frac{\sqrt{p-1}}{\sqrt{d-p}+\sqrt{p-1}}\varepsilon\|\mathcal{A}\|_F$.
2: for $k = p, \dots, d-1$ do
3:     $\delta_k := \frac{\delta_{\text{right}}}{\sqrt{d-k}}$.
4:     Steps 4–7 of Algorithm 6.
5:     $\delta_{\text{right}} := \sqrt{\delta_{\text{right}}^2 - \|\boldsymbol{E}_k\|_F^2}$.
6: end for
7: Steps 10–15 of Algorithm 6.
8: for $k = p, \dots, 2$ do
9:     $\delta_{k-1} := \frac{\delta_{\text{left}}}{\sqrt{k-1}}$.
10:    Steps 18–21 of Algorithm 6.
11:    $\delta_{\text{left}} := \sqrt{\delta_{\text{left}}^2 - \|\boldsymbol{E}_{k-1}\|_F^2}$.
12: end for
13: Return $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(d)}$ as cores of $\mathcal{B}$.
```

Theorem 10 (*Correctness of Algorithm 8*)**.** *The approximation $\mathcal{B}$ obtained in Algorithm 8 always satisfies $\|\mathcal{A} - \mathcal{B}\|_F \le \varepsilon\|\mathcal{A}\|_F$.*

The proof of Theorem 10 is based on a loop invariant [25, pp. 18–19] and provided in the Appendix.

3.6. Complexity analysis

Finding nonzero p-fibers can be accelerated by employing balanced binary search trees or hash tables, while parallel-vector rounding will be efficient if *Depar* is implemented as Algorithm 5. Notice that, there is no floating point operation in these procedures. Therefore, the time complexity of Algorithm 7 mainly depends on Algorithm 6, where the cost of SVD is of major concern. The FLOP count f_{fasttt} can thus be estimated as

$$f_{\text{fasttt}} \approx f_{\text{SVD}}(\tilde{r}_{p-1}n_p, \tilde{r}_p) + \sum_{i=p+1}^{d-1} f_{\text{SVD}}(r_{i-1}n_i, \tilde{r}_i) + \sum_{i=2}^{p} f_{\text{SVD}}(\tilde{r}_{i-1}, n_i r_i), \tag{13}$$

where $\{\tilde{r}_k\}$ and $\{r_k\}$ are the TT-ranks before and after executing Algorithm 6. According to (10), where the upper bound of $\tilde{r}_k$, i.e., $\overline{r}_k$, is given, we can estimate the upper bound of the FLOP count before any actual computation. With this estimation, p can be automatically selected as described in Algorithm 9. In line 3, $\{\tilde{r}_k\}$ can be obtained alternatively by actually executing *Depar* for a more precise estimation since it will not take much time after all. In line 4, if the actual final TT-ranks are already known or easy to predict, we can explicitly specify r_k for a better estimation as well.

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 $\mathcal{A} \in \mathbb{R}^{10\times20\times20\times10\times15\times20\times3}$ with density $\sigma = 0.001$. According to (1), the approximate FLOP count of TT-SVD is

$$\begin{aligned} f_{\text{TTSVD}} \approx & f_{\text{SVD}}(10, 20\times20\times10\times15\times20\times3) + f_{\text{SVD}}(20r, 20\times10\times15\times20\times3) \\ & + f_{\text{SVD}}(20r, 10\times15\times20\times3) + \cdots + f_{\text{SVD}}(20r, 3) \end{aligned}$$

Algorithm 9 Automatically select p.

```
Input: A sparse tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$.
Output: Selected $\bar{p}$ for best estimated performance in Algorithm 7.
1: for $p = 1, \dots, d$ do
2:     $R :=$ the number of nonzero $p$-fibers of $\mathcal{A}$.
3:     $\{\tilde{r}_k\} := \{\bar{r}_k\}$ given in (10), or executing Depar with $p$ for better estimation.
4:     $\{r_k\} := \{\tilde{r}_k\}$, or specified by users.
5:     Calculate the estimated FLOP count $f_p$ with $\{\tilde{r}_k\}$ and $\{r_k\}$ as (13).
6: end for
7: Return $\bar{p} = \operatorname{argmin} pf_p$.
```

$$
\begin{aligned}
&\approx (3.6 \times 10^8 + 7.6r^2 \times 10^7) C_{\text{SVD}} \\
&\approx (8 \times 10^9) C_{\text{SVD}}.
\end{aligned}
$$

As for f_{fasttt}, we let $p = 7$. Since the elements of $\mathcal{A}$ are grouped in triples stored in the last dimension, the number of nonzero 7-fibers R satisfies $R \le \text{nnz}(\mathcal{A})/3 = 12000$, which means $\{\bar{r}_k\}$ given in (10) is no more than $\{10, 200, 4000, 12000, 12000, 12000\}$. According to (13), we have

$$
\begin{aligned}
f_{\text{fasttt}} \approx\, & f_{\text{SVD}}(3, 12000) + f_{\text{SVD}}(20r, 12000) + f_{\text{SVD}}(15r, 12000) \\
& + f_{\text{SVD}}(10r, 4000) + f_{\text{SVD}}(20r, 200) \\
\approx\, & (1.08 \times 10^5 + 8r^2 \times 10^6) C_{\text{SVD}} \\
\approx\, & (8 \times 10^8) C_{\text{SVD}}.
\end{aligned}
$$

In this case, Algorithm 7 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 4. If we increase the density σ to 0.01, f_{TTSVD} will remain the same and f_{fasttt} will change into

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

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 $\mathcal{M}' \in \mathbb{R}^{m_1 n_1 \times \cdots \times m_d n_d}$, hence (13) 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. Therefore, 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. Since Algorithm 4 is not friendly to MATLAB, FastTT is implemented in C++ based on the *xerus* C++ toolbox [26] and with Intel Math Kernel Library.[3] The *xerus* library also contains implementations of TT-SVD and randomized TT-SVD (rTTSVD) [20]. As no C++ implementation of the TT-cross method [18] is available, we use TT-cross from *TT-toolbox*[4] in MATLAB. Since the TT-cross algorithm involves mostly matrix computations, there should be no significant performance difference between its MATLAB and C++ implementations.[5] All experiments are carried out on a x86-64 Linux server with 32 CPU cores and 512G RAM. The desired accuracy tolerance ε of TT-SVD, TT-cross and our FastTT algorithms is 10^{-14}, unless otherwise stated. The oversampling parameter of rTTSVD algorithm is set to 10. In all experiments, the CPU time is reported.

3 https://software.intel.com/en-us/mkl.
4 https://github.com/oseledets/TT-Toolbox.
5 In fact, the MATLAB implementation of TT-SVD from *TT-toolbox* is just as efficient as the C++ implementation from *xerus* on our machine.

Table 1
Experimental results on an image and a video with different observation ratios and preset TT-ranks.

Data	TT-rank	σ	time (s)				Speedup
			TT-SVD	TT-cross	rTTSVD	FastTT	
image	10	0.001	32.9	20.2	24.1	3.43	9.6X
	10	0.005	32.3	22.7	23.9	10.9	3.0X
	10	0.01	32.8	23.7	26.0	14.2	2.3X
	30	0.001	42.7	68.1	38.1	12.2	3.5X
	30	0.005	42.9	70.9	33.8	20.5	2.1X
	100	0.001	67.3	366	91.5	23.7	2.8X
video	10	0.001	66.2	24.9	56.0	10.4	6.4X
	10	0.005	66.6	30.3	60.5	26.2	2.5X
	10	0.01	66.9	31.2	62.7	33.3	2.0X
	30	0.001	103	122	108	26.5	3.9X
	30	0.005	110	140	94.2	47.6	2.3X
	100	0.001	232	1080	221	107	2.2X

4.1. Image/video inpainting

Applications like tensor completion [13] require a fixed-rank TT-approximate of the given tensors. The tensors used in this section are a large color image *Dolphin*[6] which has 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*[7] which has 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 σ 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.

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 TT-cross algorithm and the rTTSVD algorithm which also speed up the calculation in some cases. However, the speedup of them is not as great as ours, and in cases where the preset TT-rank is high we observe that they are even slower than the TT-SVD algorithm. In addition, the quality of the TT-approximation calculated by the TT-cross/rTTSVD algorithm is not as good as ours. For example, in the image inpainting task where the TT-rank is 100 and the observation ratio σ is 0.001, the mean square error (MSE) of both TT-SVD algorithm and our algorithm is 22.3, while the MSE of TT-cross and rTTSVD is 66.0 and 23.5, correspondingly.

It is also worth noting that the output from FastTT is already in site-p-mixed-canonical form [13], while the output from TT-cross or rTTSVD is not. Tensor train in site-p-mixed-canonical form is very important for collaborating with the regularization techniques [13], which has been successful to tackle the tensor completion problems.

For each of the experiments the integer p is selected automatically by the FLOP estimation in Algorithm 9. Now, we validate this FLOP estimation. For the parameters TT-rank $= 100$, $\sigma = 0.001$ in the image experiment we run Algorithm 7 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 Fig. 6, where we can see that the trend of the two curves is basically consistent. The integer p selected by Algorithm 9 is $p = 7$, with which the exact CPU time is only slightly more than the best selection at $p = 6$. Although Algorithm 9 does not always produce the best p, it certainly avoids bad values like $p = 3$ in this case.

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 [4] or AMEn [27].

For a domain partitioned into $n \times m \times k$ grids, FDM produces a coefficient matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, where $N = n \times m \times k$. For example, the sparsity pattern of the coefficient matrix $\boldsymbol{A}$ for FDM with $20 \times 20 \times 20$ grids is shown in Fig. 7. Naturally, the $\boldsymbol{A}$ matrix can be regarded as a 6-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n\times n\times m\times m\times k\times k}$, which is then converted into an MPO. Since the tensor $\boldsymbol{\mathcal{A}}$ 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.

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 *Depar* procedure can greatly reduce the TT-rank and thus simplify the computation of the TT-rounding procedure. Like in the first experiment, the TT-cross algorithm is faster when the TT-ranks are low

6 http://absfreepic.com/absolutely_free_photos/original_photos/dolphin-4000x3000_21859.jpg.
7 https://www.youtube.com/watch?v=UPtDJuJMyhc.

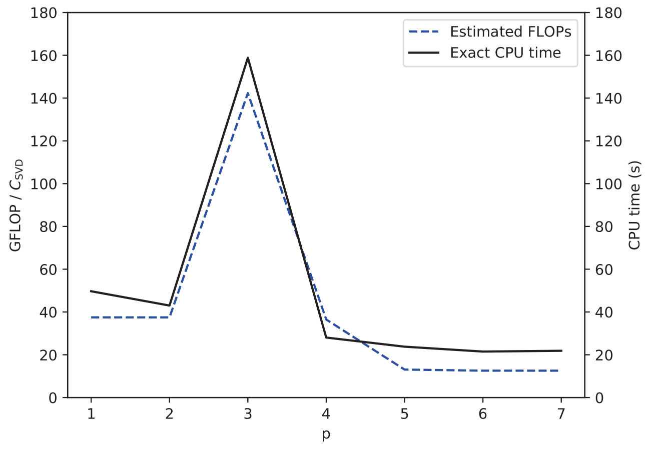

Fig. 6. The CPU time and estimated FLOPs with (13) of the FastTT algorithm for different p values.

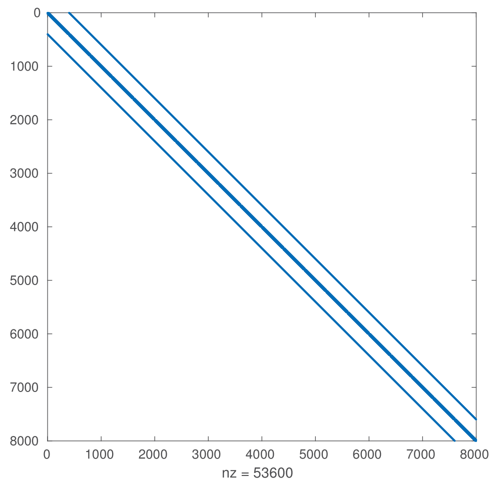

Fig. 7. The sparsity pattern of the coefficient matrix for FDM with $20 \times 20 \times 20$ grids.

but gets slower when the ranks grow. 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/TT-cross algorithm cannot produce any result in a reasonable time, while FastTT finishes in 57.5 s 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 $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, which can also be 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*[8] from SNAP [28], which contains 1088092 nodes and 1541898 undirected edges. Since the graph is fairly large, we take the subgraph of the first N nodes as our data and preprocess its adjacency matrix by performing reverse Cuthill–McKee ordering [29]. Additionally, different desired accuracy tolerances ε and the actual relative error are tested in this experiment. The truncation parameters in Algorithm 6 are either set as $\delta_k = \frac{\varepsilon}{\sqrt{p-1}+\sqrt{d-p}}\|\boldsymbol{\mathcal{A}}\|_F$ or determined by Algorithm 8. The results are shown in Table 3. The TT-cross/rTTSVD algorithm is not tested because both of them require the TT-ranks to be set the same, which is not possible in this experiment.

[8] Road network of Pennsylvania. http://snap.stanford.edu/data/roadNet-PA.html.

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

n	coefficients	method	time(s)	speedup	$\varepsilon_{\text{actual}}$[a]	TT-ranks[b]
20	pattern	TT-SVD	43.6	–	4.0×10^{-16}	r : 2, 2
		TT-cross	7.96	5.5X	2.0×10^{-16}	r : 2, 2
		rTTSVD	1.29	34X	1.2×10^{-15}	r : 2, 2
		FastTT	0.788	55X	9.6×10^{-16}	R: 1920; $\tilde{r}$: 58,58; r: 2,2
30	pattern	TT-SVD	690	–	2.0×10^{-15}	r : 2, 2
		TT-cross	26.5	26X	8.8×10^{-16}	r : 2, 2
		rTTSVD	19.3	36X	1.6×10^{-15}	r : 2, 2
		FastTT	2.88	240X	1.1×10^{-15}	R: 4380; $\tilde{r}$: 88,88; r: 2,2
20	random	TT-SVD	53.4	–	2.5×10^{-15}	r : 58, 58
		TT-cross	226	0.24X	1.1×10^{-15}	r : 58, 58
		rTTSVD	23.4	2.3X	4.8×10^{-15}	r : 58, 58
		FastTT	1.67	32X	2.4×10^{-15}	R: 1920; $\tilde{r}$: 58,58; r: 58,58
30	random	TT-SVD	762	–	3.4×10^{-15}	r : 88, 88
		TT-cross	2725	0.28X	6.2×10^{-15}	r : 88, 88
		rTTSVD	67.0	11X	4.2×10^{-15}	r : 88, 88
		FastTT	12.4	61X	3.3×10^{-15}	R: 4380; $\tilde{r}$: 88,88; r: 88,88
40	random	TT-SVD	NA	–	NA	NA
		TT-cross	NA	–	NA	NA
		rTTSVD	597	–	5.0×10^{-15}	r : 118, 118
		FastTT	57.5	–	2.6×10^{-15}	R: 7840; $\tilde{r}$: 118,118; r: 118,118

[a] $\varepsilon_{\text{actual}} = \frac{\|\mathcal{A}-\mathcal{B}\|_F}{\|\mathcal{A}\|_F}$. The same below.
[b] R is the number of nonzero p-fibers. $\tilde{r}$ is the TT-ranks after parallel-vector rounding. r is the final TT-ranks.

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

N	ε	method[a]	time (s)	speedup	TT-ranks	$\varepsilon_{\text{actual}}$
20^3	1×10^{-14}	TT-SVD	75.4	–	58, 400	3.7×10^{-15}
		FastTT	14.1	5.3X	58, 400	3.3×10^{-15}
		FastTT^{sp}	0.065	1160X	58, 400	$<10^{-15}$
20^3	5×10^{-1}	TT-SVD	62.2	–	31, 281	4.8×10^{-1}
		FastTT	11.8	5.3X	31, 281	4.8×10^{-1}
		FastTT^{+}	10.4	6.0X	55, 209	5.0×10^{-1}
10^4	1×10^{-14}	TT-SVD	833	–	28, 1407, 70	3.9×10^{-15}
		FastTT	23.3	34X	28, 1407, 70	4.3×10^{-15}
		FastTT^{sp}	0.089	9360X	28, 1446, 70	$<10^{-15}$
10^4	1×10^{-2}	TT-SVD	839	–	28, 1390, 70	5.5×10^{-3}
		FastTT	24.4	34X	28, 1395, 70	3.8×10^{-3}
		FastTT^{+}	24.2	35X	28, 1377, 70	9.8×10^{-3}
100^3	1×10^{-14}	TT-SVD	NA	–	NA	NA
		FastTT^{sp}	39.9	–	298, 5442	$<10^{-15}$

[a] FastTT: use Algorithm 6 for TT-rounding; FastTT^{+}: use Algorithm 8 for TT-rounding; FastTT^{sp}: use FastTT without 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 ε. If ε is small enough, the TT-rank obtained by FastTT is the same as those obtained by TT-SVD. Otherwise, Algorithm 8 (used in FastTT^{+}) usually produces lower TT-ranks and a little bit higher relative error than Algorithm 6 (used in FastTT) which sets unified truncation parameters.

If we do not demand an output whose TT-ranks are exactly as small as those from TT-SVD, we can use FastTT^{sp} which skips TT-rounding and produces sparse output without precision loss. From the results on the 1st data and the 3rd data in Table 3, we can observe that FastTT^{sp} is much faster and the resulted TT-ranks are almost as good as those from TT-SVD. For the last data in Table 3, we see that FastTT^{sp} can process nearly the whole *roadNet-PA* with 2836814 nonzeros and 79957 nonzero fibers in only 39.9s, while the other counterparts cannot produce any result in a reasonable time.

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 have verified the actual performance of each algorithm and confirmed our theoretical analysis. The experimental results also show that the proposed FastTT algorithm for sparse tensors is of good efficiency and versatility. Previous state-of-the-art algorithms are 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. The proposed algorithm is therefore very promising to tackle tensor applications that were previously infeasible.

Appendix. Proof of Theorem 10

Proof. From the proof of Theorem 9, we know that

$$\|\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{B}}\|_F \leq \sqrt{\sum_{k=p}^{d-1} \|\boldsymbol{E}_k\|_F^2} + \sqrt{\sum_{k=2}^{p} \|\boldsymbol{E}_{k-1}\|_F^2}.$$

Now we are going to use a loop invariant to prove the correctness of Algorithm 8. The loop invariant for Loop 2–6 is

$$\delta_{\text{right}}^2 + \sum_{i=p}^{k-1} \|\boldsymbol{E}_i\|_F^2 = \frac{d-p}{\sqrt{d-p} + \sqrt{p-1}} \varepsilon^2 \|\boldsymbol{\mathcal{A}}\|_F^2.$$

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

Maintenance: After each iteration, δ_{right}^2 is decreased by $\|\boldsymbol{E}_k\|_F^2$ and $\sum_{i=p}^{k-1} \|\boldsymbol{E}_i\|_F^2$ is increased by $\|\boldsymbol{E}_k\|_F^2$. Thus the invariant remains satisfied.

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

$$\delta_{\text{right}}^2 + \sum_{i=p}^{d-1} \|\boldsymbol{E}_i\|_F^2 = \frac{d-p}{\sqrt{d-p} + \sqrt{p-1}} \varepsilon^2 \|\boldsymbol{\mathcal{A}}\|_F^2$$

$$\Rightarrow \sqrt{\sum_{k=p}^{d-1} \|\boldsymbol{E}_k\|_F^2} \leq \frac{\sqrt{d-p}}{\sqrt{d-p} + \sqrt{p-1}} \varepsilon \|\boldsymbol{\mathcal{A}}\|_F.$$

Similarly we can prove that

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

is satisfied after Loop 8–12.

Thus

$$\begin{aligned} \|\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{B}}\|_F &\leq \sqrt{\sum_{k=p}^{d-1} \|\boldsymbol{E}_k\|_F^2} + \sqrt{\sum_{k=2}^{p} \|\boldsymbol{E}_{k-1}\|_F^2}. \\ &\leq \frac{\sqrt{d-p}}{\sqrt{d-p} + \sqrt{p-1}} \varepsilon \|\boldsymbol{\mathcal{A}}\|_F + \frac{\sqrt{p-1}}{\sqrt{d-p} + \sqrt{p-1}} \varepsilon \|\boldsymbol{\mathcal{A}}\|_F \\ &= \varepsilon \|\boldsymbol{\mathcal{A}}\|_F. \quad \square \end{aligned}$$

References

[1] Frank L. Hitchcock, The expression of a tensor or a polyadic as a sum of products, Stud. Appl. Math. 6 (1–4) (1927) 164–189.
[2] Ledyard R. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika 31 (3) (1966) 279–311.
[3] Ivan V. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput. 33 (5) (2011) 2295–2317.
[4] Ivan V. Oseledets, S.V. Dolgov, Solution of linear systems and matrix inversion in the TT-format, SIAM J. Sci. Comput. 34 (5) (2012) A2718–A2739.
[5] Zheng Zhang, Xiu Yang, Ivan V Oseledets, George E Karniadakis, Luca Daniel, Enabling high-dimensional hierarchical uncertainty quantification by ANOVA and tensor-train decomposition, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 34 (1) (2014) 63–76.
[6] Haotian Liu, Luca Daniel, Ngai Wong, Model reduction and simulation of nonlinear circuits via tensor decomposition, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 34 (7) (2015) 1059–1069.
[7] Zheng Zhang, Kim Batselier, Haotian Liu, Luca Daniel, Ngai Wong, Tensor computation: A new framework for high-dimensional problems in EDA, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 36 (4) (2017) 521–536.
[8] Kim Batselier, Zhongming Chen, Ngai Wong, Tensor network alternating linear scheme for MIMO Volterra system identification, Automatica 84 (2017) 26–35.
[9] Ivan V. Oseledets, Approximation of $2^d \times 2^d$ matrices using tensor decomposition, SIAM J. Matrix Anal. Appl. 31 (4) (2010) 2130–2145.

[10] Daniel Kressner, André Uschmajew, On low-rank approximability of solutions to high-dimensional operator equations and eigenvalue problems, Linear Algebra Appl. 493 (2016) 556–572.
[11] Kim Batselier, Wenjian Yu, Luca Daniel, Ngai Wong, Computing low-rank approximations of large-scale matrices with the tensor network randomized SVD, SIAM J. Matrix Anal. Appl. 39 (3) (2018) 1221–1244.
[12] Wenqi Wang, Vaneet Aggarwal, Shuchin Aeron, Efficient low rank tensor ring completion, in: IEEE International Conference on Computer Vision, ICCV, 2017, pp. 5698–5706.
[13] Ching-Yun Ko, Kim Batselier, Wenjian Yu, Ngai Wong, Fast and accurate tensor completion with total variation regularized tensor trains, 2018, arXiv preprint arXiv:1804.06128.
[14] Wenqi Wang, Vaneet Aggarwal, Shuchin Aeron, Tensor train neighborhood preserving embedding, IEEE Trans. Signal Process. 66 (10) (2018) 2724–2732.
[15] Yongkang Wang, Weicheng Zhang, Zhuliang Yu, Zhenghui Gu, Hao Liu, Zhaoquan Cai, Congjun Wang, Shihan Gao, Support vector machine based on low-rank tensor train decomposition for big data applications, in: 2017 12th IEEE Conference on Industrial Electronics and Applications, ICIEA, IEEE, 2017, pp. 850–853.
[16] Zhongming Chen, Kim Batselier, Johan AK Suykens, Ngai Wong, Parallelized tensor train learning of polynomial classifiers, IEEE Trans. Neural Netw. Learn. Syst. 29 (10) (2017) 4621–4632.
[17] Xiaowen Xu, Qiang Wu, Shuo Wang, Ju Liu, Jiande Sun, Andrzej Cichocki, Whole brain fMRI pattern analysis based on tensor neural network, IEEE Access 6 (2018) 29297–29305.
[18] Ivan Oseledets, Eugene Tyrtyshnikov, TT-Cross approximation for multidimensional arrays, Linear Algebra Appl. 432 (1) (2010) 70–88.
[19] Dmitry Savostyanov, Ivan 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.
[20] Benjamin Huber, Reinhold Schneider, Sebastian Wolf, A randomized tensor train singular value decomposition, in: Compressed Sensing and Its Applications, Springer, Cham, 2017, pp. 261–290.
[21] Nathan Halko, Per-Gunnar Martinsson, Joel A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2) (2011) 217–288.
[22] Sergei A Goreinov, Eugene E Tyrtyshnikov, Nickolai L Zamarashkin, A theory of pseudoskeleton approximations, Linear Algebra Appl. 261 (1–3) (1997) 1–21.
[23] Vicente Hernandez, Jose E. Roman, Vicente Vidal, SLEPC: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Trans. Math. Softw. 31 (3) (2005) 351–362.
[24] C. Hubig, I.P. McCulloch, U. Schollwöck, Generic construction of efficient matrix product operators, Phys. Rev. B 95 (3) (2017) 035129.
[25] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, 2009.
[26] Benjamin Huber, Sebastian Wolf, Xerus - a general purpose tensor library, 2014–2017.
[27] Sergey V. Dolgov, Dmitry V. Savostyanov, Corrected one-site density matrix renormalization group and alternating minimal energy algorithm, in: Numerical Mathematics and Advanced Applications-ENUMATH 2013, Springer, 2015, pp. 335–343.
[28] Jure Leskovec, Andrej Krevl, SNAP Datasets: Stanford large network dataset collection, 2015.
[29] Alan George, Joseph W. Liu, Computer Solution of Large Sparse Positive Definite, Prentice Hall Professional Technical Reference, 1981.