Practical Sketching-Based Randomized Tensor Ring Decomposition

Yajie Yu† and Hanyu Li†

Abstract. Based on sketching techniques, we propose two randomized algorithms for tensor ring (TR) decomposition. Specifically, by defining new tensor products and investigating their properties, we apply the Kronecker sub-sampled randomized Fourier transform and TensorSketch to the alternating least squares problems derived from the minimization problem of TR decomposition to devise the randomized algorithms. From the former, we find an algorithmic framework based on random projection for randomized TR decomposition. Theoretical results on sketch size and complexity analyses for the two algorithms are provided. We compare our proposals with the state-of-the-art method using both synthetic and real data. Numerical results show that they have quite decent performance in accuracy and computing time.

Key words. tensor ring decomposition, randomized algorithm, alternating least squares problem, sketching, Kronecker sub-sampled randomized Fourier transform, TensorSketch

MSC codes. 15A69, 68W20

1. Introduction. Tensor decompositions represent a higher-order tensor by multilinear operations over the latent factors, which have found many applications in machine learning, signal processing, chemometrics and so on; see the detailed review in [18, 31, 9]. Two popular decompositions are the Canonical Polyadic (CP) decomposition and the Tucker decomposition. The former has the parameters scaling linearly with the tensor order and the latter is easier to work numerically. So these two decompositions have wide applications [18, 31, 9]. However, they also suffer from some limitations. For example, finding the CP decomposition of a tensor is an NP-hard problem; the number of parameters of Tucker decomposition still scales exponentially with the tensor order. Tensor train (TT) decomposition, which is also known as the matrix product state (MPS) with open boundary conditions in quantum physics [1, 30], came into being to address these challenges [28]. However, it also has some limitations such as the constraint on TT-ranks, the fixed pattern of TT-ranks, and the strict order of TT-cores [35]. For these reasons, Zhao et al. [35] introduced the tensor ring (TR) decomposition, i.e., the MPS with periodic boundary conditions, which has the circular dimensional permutation invariance and hence can overcome some shortcomings of TT decomposition.

Specifically, the TR decomposition in the element-wise form of the tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ can be represented as follows:

$$\mathbf{X}(i_1, \cdots, i_N) = \text{trace} \left( \mathbf{G}_1(i_1) \mathbf{G}_2(i_2) \cdots \mathbf{G}_N(i_N) \right) = \text{trace} \left( \prod_{n=1}^{N} \mathbf{G}_n(i_n) \right),$$

where $\mathbf{G}_n(i_n) = \mathbf{G}_n(:, i_n, :) \in \mathbb{R}^{R_n \times R_{n+1}}$ is the $i_n$-th lateral slice of the core tensor (TR-core) $\mathbf{G}_n \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}$. Note that a slice is an 2-order section, i.e., a matrix, of a tensor obtained

---

*This work was funded by the National Natural Science Foundation of China (No. 11671060) and the Natural Science Foundation of Chongqing, China (No. cstc2019jcxy-msxmX0267).

†College of Mathematics and Statistics, Chongqing University, Chongqing 401331, P.R. China (zqyu@cqu.edu.cn; lihy.hy@gmail.com or hyli@cqu.edu.cn).
by fixing all the tensor indices but two, and \( R_{N+1} = R_1 \). The sizes of TR-cores, i.e., \( R_k \) with \( k = 1, \ldots, N \), are called TR-ranks. Additionally, we use the notation \( \text{TR} (\{\mathcal{G}_n\}_{n=1}^N) \) to denote the TR decomposition of a tensor. The problem of fitting \( \text{TR} (\{\mathcal{G}_n\}_{n=1}^N) \) to a tensor \( \mathcal{X} \) can be written as the following minimization problem:

\[
\arg\min_{\mathcal{G}_1, \cdots, \mathcal{G}_N} \| \text{TR} (\{\mathcal{G}_n\}_{n=1}^N) - \mathcal{X} \|_F,
\]

where \( \| \cdot \|_F \) denotes the Frobenius norm of a matrix or tensor. One of the common approaches to this problem is SVD based, and another one uses the alternating least squares (ALS) [35]. We use the abbreviations TR-SVD and TR-ALS for these two methods in the following text.

Due to the expensive computational costs of the above two methods, it is crucial to design efficient randomized algorithms for computing the TR decompositions of large-scale tensors. As far as we know, the first randomized algorithm for TR decomposition was proposed by Yuan et al. [34] who first applied the randomized Tucker decomposition to the original tensor and then the TR-SVD or TR-ALS to the core tensor of the Tucker decomposition. Finally, the original TR-cores were recovered by combining the factor matrices of the randomized Tucker decomposition and the TR-cores of the core tensor. Later, Ahmadi-Asl et al. [2] summarized some algorithms of TR decomposition and presented several randomized algorithms with randomized SVD. Recently, building on TR-ALS, Malik and Becker [25] provided a sampling method based on leverage scores, which can avoid forming the large coefficient matrices in TR-ALS via the multi-index and can outperform the randomized method given in [34]. Moreover, Malik [21] also provided a new approach to approximate the leverage scores and devised the corresponding sampling algorithm for TR decomposition.

As we know, there are two main types of methods for designing randomized algorithms [13, 10, 26, 14, 33]. One is random sampling, and the other is random projection. The latter is often referred to as the sketching-based method. Actually, both of the two types of methods can be regarded as sketching-based methods because the processed matrix in these two methods can be written in a unified form. That is, the matrix is multiplied by a matrix with random entries. Besides TR decomposition, the above two types of methods have also been applied to other tensor decompositions such as CP, Tucker, and TT decompositions. For example, Battaglino et al. [4] first investigated the uniform sampling for CP decomposition comprehensively, and then Larsen and Kolda [19] considered its leverage-based sampling. The latter inspires the works in [25, 21] on TR decomposition mentioned above\(^1\). Again, it should be emphasized that the sampling methods used in these works can avoid forming the large coefficient matrices and they carry out sample selection in factor matrices or core tensors. For random projection, scholars mainly used Gaussian random matrix, sub-sampled randomized Hadamard transform (SRHT), sub-sampled randomized Fourier transform (SRFT), CountSketch (CS) and its variant TensorSketch to design the randomized algorithms for CP, Tucker, and TT decompositions; see, e.g., [36, 4, 5, 27, 6, 7, 20]. In particular, in [4], the SRFT is applied to each factor matrix of CP decomposition rather than the large coefficient matrix in the ALS (CP-ALS) derived from the minimization problem of CP decomposition; in [20], the method is mainly built on a formula that combines the TensorSketch and the Kronecker-like form of coefficient matrix.

---

\(^1\) In [21], the author also provided the sampling algorithm for CP decomposition.
In this paper, motivated by [25] and the warning in [26, Section 9] that leverage score sampling is rarely a competitive method in practice and may perform markedly worse, we consider the sketching-based methods for TR decomposition on the basis of TR-ALS. Specifically, in our algorithm, we will use Kronecker SRFT (KSRFT), which is similar to what was done in [4] for CP decomposition but with some quite remarkable improvements. This is mainly because unlike the coefficient matrices in CP-ALS [4], it is difficult to straighten the structure of the coefficient matrices in TR-ALS; see (3.1) below. To this end, we define a new tensor product and find an elegant property of the product. Moreover, these new findings make us avoid forming the full coefficient matrices, the large sketching matrices and the matrix multiplications between them, and enable us to detect a framework for designing the randomized algorithm for TR decomposition; see Remark 4.2 below for details. Furthermore, as done in [20], we also consider TensorSketch. However, for our problem, it is more complicated. To this end, we define another new product of tensors and obtain a graceful formula. Therefore, in the ideas on sketching methods, the work is mainly inspired by [4] and [20] in essence.

The remainder of this paper is organized as follows. Section 2 provides some preliminaries. In Section 3, we review some existing algorithms for TR decomposition and present our new findings including new tensor products and their properties, structure analysis of the coefficient matrices, and a formula for TensorSketch. The algorithms based on KSRFT and TensorSketch and their theoretical analyses and computational complexities are given in Sections 4 and 5, respectively. In Section 6, we compare our algorithms with the existing ones using both synthetic and real data. Finally, the concluding remarks of the whole paper are presented.

2. Preliminaries. We first introduce some necessary definitions.

Definition 2.1 (Multi-index [12]). For a positive integer $I$, let $[I] \overset{def}{=} \{1, \cdots, I\}$. For indices $i_1 \in [I_1], \cdots, i_N \in [I_N]$, a multi-index $i = i_1 i_2 \cdots i_N$ refers to an index which takes all possible combinations of values of the indices, $i_1, i_2, \cdots, i_N$, for $i_n = 1, 2, \cdots, I_n$ with $n = 1, 2, \cdots, N$ in a specific order. Two common orders are as follows.

1. Little-endian convention (reverse lexicographic ordering)
   \[
   i_1 i_2 \cdots i_N = i_1 + (i_2 - 1)I_1 + (i_3 - 1)I_1 I_2 + \cdots + (i_N - 1)I_1 \cdots I_{N-1}.
   \]

2. Big-endian convention (colexicographic ordering)
   \[
   i_1 i_2 \cdots i_N = i_N + (i_{N-1} - 1)I_N + (i_{N-2} - 1)I_N I_{N-1} + \cdots + (i_1 - 1)I_2 \cdots I_N.
   \]

In this paper, unless otherwise stated, we will use the little-endian convention.

Definition 2.2 (Kronecker product). The Kronecker product of two matrices $A = (a_{ij}) \in \mathbb{R}^{I_1 \times J_1}$ and $B \in \mathbb{R}^{I_2 \times J_2}$ is a matrix of size $I_1 I_2 \times J_1 J_2$ denoted by $A \otimes B$ and defined as

\[
A \otimes B = \begin{bmatrix}
  a_{11}B & \cdots & a_{1J_1}B \\
  \vdots & \ddots & \vdots \\
  a_{I_11}B & \cdots & a_{I_1J_1}B
\end{bmatrix}.
\]

If using the notation in Definition 2.1, the element-wise form of the Kronecker product is
as follows:
\[(A \otimes B)(i_2i_1, j_2j_1) = A(i_1, j_1)B(i_2, j_2).\]

**Definition 2.3 (Khatri-Rao product).** The Khatri-Rao product of two matrices \(A \in \mathbb{R}^I \times J\) and \(B \in \mathbb{R}^I \times J\) is a matrix of size \(I_1I_2 \times J\) denoted by \(A \hat{\otimes} B\) and defined as

\[A \hat{\otimes} B = [a_1 \otimes b_1, a_2 \otimes b_2, \ldots, a_J \otimes b_J],\]

where \(a_n\) and \(b_n\) with \(n \in [J]\) are the \(n\)-th columns of the matrices \(A\) and \(B\), respectively.

If using the notation in **Definition 2.1**, the element-wise form of the Khatri-Rao product is as follows:

\[(A \otimes B)(i_2i_1, j) = A(i_1, j)B(i_2, j).\]

**Definition 2.4 (Mode-\(n\) product [8, 18]).** The mode-\(n\) product of a tensor \(X \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}\) with a matrix \(U \in \mathbb{R}^{J \times I_n}\) is a tensor of size \(I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N\) denoted by \(X \times_n U\) and defined as

\[(X \times_n U)_{i_1 \cdots i_{n-1}j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1i_2\cdots i_n} u_{ji_n}.\]

**Definition 2.5 (Mode-\(n\) unfolding [35, 8]).** The mode-\(n\) unfolding of a tensor \(X \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}\) is the matrix \(X^{(n)}\) of size \(I_n \times \prod_{j \neq n} I_j\) defined element-wise via

\[X^{(n)}(i_n, i_1 \cdots i_{n-1}i_{n+1} \cdots i_N) = X(i_1, \cdots, i_N).\]

**Definition 2.6 (Classical mode-\(n\) unfolding [35, 8, 18]).** The classical mode-\(n\) unfolding of a tensor \(X \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}\) is the matrix \(X^{(n)}\) of size \(I_n \times \prod_{j \neq n} I_j\) defined element-wise via

\[X^{(n)}(i_n, i_1 \cdots i_{n-1}i_{n+1} \cdots i_N) = X(i_1, \cdots, i_N).\]

For the tensor in a series of mode-\(n\) product form, we can rewrite its mode-\(n\) and classical mode-\(n\) unfolding matrices using the following formulas.

**Proposition 2.7 ([8]).** Let the tensor \(\hat{X}\) have the form \(\hat{X} = X \times_1 U_1 \times_2 U_2 \cdots \times_N U_N\), where \(X \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}\), and \(U_n \in \mathbb{R}^{J \times I_n}\) for \(n = 1, \ldots, N\). Then

\[\hat{X}^{(n)} = U_n X^{(n)} (U_{n-1} \otimes \cdots \otimes U_1 \otimes U_N \otimes \cdots \otimes U_{n+1})^T,\]
\[\hat{X}^{(n)} = U_n X^{(n)} (U_N \otimes \cdots \otimes U_{n+1} \otimes U_{n-1} \otimes \cdots \otimes U_1)^T.\]

**Definition 2.8 (Subchain tensor [35]).** Let \(X = TR(\{G_n\}_{n=1}^N) \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}\). The subchain tensor \(G^{\pi_n} \in \mathbb{R}^{R_{n+1} \times \prod_{j \neq n} I_j \times R_n}\) is the merging of all TR-cores expect the \(n\)-th one and can be written slice-wise via

\[
G^{\pi_n}(i_{n+1} \cdots i_{N}i_1 \cdots i_{n-1}) = \prod_{j=n+1}^{N} G_j(i_j) \prod_{j=1}^{n-1} G_j(i_j).
\]
In the following, we introduce KSRFT and TensorSketch. As mentioned in Section 1, we will use them to devise the corresponding algorithms.

**Definition 2.9 (KSRFT [4, 17]).** The KSRFT is defined as

\[
\Phi = \sqrt{\prod_{j=1}^{N} \frac{I_j}{m}} S \left( \bigotimes_{j=1}^{N} (F_j D_j) \right),
\]

where

- \( S \in \mathbb{R}^{m \times \prod_{j=1}^{N} I_j} \): \( m \) rows of the \( \prod_{i=1}^{N} I_i \) \( I_j \times \prod_{j=1}^{N} I_j \) identity matrix drawn uniformly at random with replacement from the identity matrix;
- \( F_j \in \mathbb{C}^{I_j \times I_j} \): (unitary) discrete Fourier transform of dimension \( I_j \) (also called DFT/FFT matrix);
- \( D_j \in \mathbb{R}^{I_j \times I_j} \): a diagonal matrix with independent random diagonal entries drawn uniformly from \( \{+1, -1\} \) (also called random sign-flip operator).

**Remark 2.10.** Actually, the KSRFT is a special Kronecker fast Johnson-Lindenstrauss transform, i.e., KFJLT, proposed and studied in [4, 17, 24].

**Definition 2.11 (TensorSketch [29, 11]).** The TensorSketch is defined as \( T = \Omega D \), where

- \( \Omega \in \mathbb{R}^{m \times \prod_{j=1}^{N} I_j} \): a matrix with \( \Omega(j, i) = 1 \) if \( j = H(i) \) for all \( i \in \prod_{j=1}^{N} I_j \) and \( \Omega(j, i) = 0 \) otherwise;
- \( D \in \mathbb{R}^{\prod_{j=1}^{N} I_j \times \prod_{j=1}^{N} I_j} \): a diagonal matrix with \( D(i, i) = S(i) \).

In the definitions of \( \Omega \) and \( D \),

\[
H : [I_1] \times [I_2] \times \cdots \times [I_N] \rightarrow [m] : (i_1, \ldots, i_N) \mapsto \left( \sum_{n=1}^{N} (H_n(i_n) - 1) \mod m \right) + 1,
\]

\[
S : [I_1] \times [I_2] \times \cdots \times [I_N] \rightarrow \{-1, 1\} : (i_1, \ldots, i_N) \mapsto \prod_{n=1}^{N} S_n(i_n),
\]

where each \( H_n \) for \( n \in [N] \) is a \( 3 \)-wise independent hash map that maps \([I_n] \to [m]\), and each \( S_n \) is a \( 4 \)-wise independent hash map that maps \([I_n] \to \{-1, 1\}\). Recall that a hash map is \( k \)-wise independent if all the designated \( k \) keys are independent random variables.

**Remark 2.12.** In Definition 2.11, we use the little-endian convention to compute

\[
H(i) = H(i_1 i_2 \cdots i_N) \quad \text{and} \quad S(i) = S(i_1 i_2 \cdots i_N),
\]

while the definition of the TensorSketch in [11, 23, 22, 20] uses the big-endian convention.

### 3. Existing algorithms and new findings.

We first introduce TR-ALS mentioned in Section 1 and the main idea of the randomized algorithm proposed in [25].

According to Theorem 3.5 in [35], the objective in (1.1) can be rewritten as the following \( N \) subproblems

\[
\arg\min_{G_{(2)}^{n(2)}} \| G_{(2)}^{n(2)} G_{(2)}^{n(2)} - X_{(2)}^{n} \|_F, \ n = 1, \cdots, N.
\]
The so-called TR-ALS is a method that keeps all cores fixed except the \( n \)-th one and finds the solution to the least squares problem (3.1) with respect to it. We summarize TR-ALS in Algorithm 3.1.

**Algorithm 3.1 TR-ALS [35]**

1: function \( \{ G_n \}_{n=1}^N = \text{TR-ALS}(X, R_1, \cdots, R_N) \) \( \triangleright \) \( X \) is the input tensor
2: \( \triangleright R_1, \cdots, R_N \) are the TR-ranks
3: Initialize cores \( G_2, \cdots, G_N \)
4: repeat
5: for \( n = 1, \cdots, N \) do
6: Compute \( G_n^{[2]} \) from cores
7: Update \( G_n = \arg \min_Z \| G_n^{[2]} Z_{[n]} - X_{[n]} \|_F \)
8: end for
9: until termination criteria met
10: return \( G_1, \cdots, G_N \)

Since the size of the matrix \( G_n^{[2]} \) is \( \prod_{j \neq n} I_j \times R_n R_{n+1} \), it is very expensive to solve the least squares problem (3.1) directly. To tackle this problem, Malik and Becker [25] proposed a random sampling variant of TR-ALS based on importance sampling. The main advantage of the variant is that it can obtain the sketch with sampling without forming the large matrix \( G_n^{[2]} \). This is mainly due to the authors noting that the \( i_{n+1} \cdots i_{N+1} \cdots i_{n-1} \)-th lateral slice of the subchain tensor \( G_n^{[2]} \) can be given by a sequence of matrix multiplications, that is, (2.1) in Definition 2.8. So they devised an algorithm to find the sampled sketch of the tensor \( G_n^{[2]} \) and hence of the matrix \( G_n^{[2]} \) by extracting \( m \) lateral slices from each of the cores \( G_j \).

The algorithm of this sampling method is summarized in Algorithm 3.2. With this algorithm and the leverage scores or their approximations of the classical mode-2 unfolding matrices of TR-cores, the random sampling variant of TR-ALS, i.e., TR-ALS-Sampled, can be derived [25]. We summarize it in Algorithm 3.3.

**Algorithm 3.2 Sampled Subchain Tensor (SST), summarized from [25]**

1: function \( G_n^{[2]} = \text{SST}(\text{idxs}, G_{n+1}, \cdots, G_N, G_1, \cdots, G_{n-1}) \) \( \triangleright \) \( G_n \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}, n \in [N] \)
2: \( \triangleright \text{idxs} \in \mathbb{R}^{m \times (N-1)} \) is from the set of tuples \( \{ i_{n+1}^{(j)}, \cdots, i_{N+1}^{(j)}, i_1^{(j)}, \cdots, i_{n-1}^{(j)} \} \) for \( j \in [m] \)
3: \( \triangleright \text{idxs} \) can be retrieved from the sampling matrix \( S \in \mathbb{R}^{m \times \prod_{k \in n} I_k} \) or the specific sampling with given probabilities
4: Let \( G_n^{[2]} \) be a tensor of size \( R_{n+1} \times m \times R_n \), where every lateral slice is an \( R_{n+1} \times R_n \) identity matrix
5: for \( k = n + 1, \cdots, N, 1, \cdots, n - 1 \) do
6: \( G_{[k]}^{[2]} \leftarrow G_{[k]}(; \text{idxs}(; k), :) \)
7: \( G_n^{[2]} \leftarrow G_n^{[2]} \boxplus_2 G_{[k]}^{[2]} \) \( \triangleright \) we use \( \boxplus_2 \) defined in Definition 3.3 below to simplify the representation
8: end for
9: return \( G_n^{[2]} \)
10: end function

Motivated by TR-ALS-Sampled and the warning in [26, Section 9], i.e., sampling can be a bit precarious, and inspired by the idea in [4], we would like to combine TR-ALS with the sketching techniques to design the sketch-based algorithms for TR decomposition. To this
Algorithm 3.3 TR-ALS-Sampled [25]

1: function \{\mathcal{G}_n\}_{n=1}^N = \text{TR-ALS-Sampled}(\mathcal{X}, R_1, \ldots, R_N, m) \quad \triangleright \text{m is the sampling size}
2: Initialize cores \mathcal{G}_2, \ldots, \mathcal{G}_N
3: Compute probability distributions \mathbf{p}_2, \ldots, \mathbf{p}_N based on leverage scores
4: repeat
5: for \(n = 1, \ldots, N\) do
6: Draw sampling matrix \(S\) using the probability distributions of \(N - 1\) cores
7: Retrieve \(\text{idxs} \) from \(S\)
8: Update \(\mathcal{G}_n = \arg \min_{Z} \| \mathcal{G}_n Z - X_s \|_F\)
9: end for
10: Update probability distributions \(p_n\) based on leverage scores
11: until termination criteria met
12: end function

end, we have to straighten the structure of the matrices \(SG_{[2]}^{\neq n}\) and \(G_{[2]}^{\neq n}\), where \(S\) is some sketching matrix. This is because, like [4], we also want to find the sketch \(SG_{[2]}^{\neq n}\) without forming \(S\) and \(G_{[2]}^{\neq n}\). In the following, we present the specific analysis.

Firstly, note that each row of \(SG_{[2]}^{\neq n}\) is the linear combination of the rows of the matrix \(G_{[2]}^{\neq n}\), and the rows of the matrix \(G_{[2]}^{\neq n}\) are the vectorization of the lateral slices of the subchain tensor \(G^{\neq n}\). Therefore, from the perspective of the subchain tensor, the above process can be carried out from a linear combination of the lateral slices of \(G^{\neq n}\). Moreover, if we use the properties of mode-2 product, the above operation can be written as \(G^{\neq n} \times_2 S\). We illustrate this transformation and process in Figure 1.

Secondly, from (2.1), we observe that the subchain tensor \(G^{\neq n}\) may be written as a Kronecker-like or Khatri-Rao-like product of TR-cores if we define a tensor product felicitously. The product is given as follows, and its graphical illustration is provided in Figure 2a.

**Definition 3.1 (Subchain product).** Let \(A \in \mathbb{R}^{I_1 \times J_1 \times K}\) and \(B \in \mathbb{R}^{K \times J_2 \times I_2}\) be two 3-order tensors, and \(A(j_1)\) and \(B(j_2)\) be the \(j_1\)-th and \(j_2\)-th lateral slices of \(A\) and \(B\), respectively. The mode-2 subchain product of \(A\) and \(B\) is a tensor of size \(I_1 \times J_1 J_2 \times I_2\) denoted by \(A \boxtimes_2 B\) and defined as

\[
(A \boxtimes_2 B)(j_1 j_2) = A(j_1) B(j_2).
\]

That is, with respect to the correspondence on indices, the lateral slices of \(A \boxtimes_2 B\) are the classical matrix products of the lateral slices of \(A\) and \(B\). The mode-1 and mode-3 subchain products can be defined similarly.

Using this definition, we can rewrite the subchain tensor \(G^{\neq n}\) in Definition 2.8 as follows

\[(3.2) \quad G^{\neq n} = G_{n+1} \boxtimes_2 \cdots \boxtimes_2 G_N \boxtimes_2 G_1 \boxtimes_2 \cdots \boxtimes_2 G_{n-1}.\]

In the following, we present a property of the above subchain product.
Figure 1. Illustration of the process for obtaining $SG_{[2]}^{\neq n}$ via $G_{\neq n}^{\neq n} = n \times 2 S$.

(a) Illustration of the subchain product given in Definition 3.1.

(b) Illustration of the slices-Hadamard product given in Definition 3.3.

Figure 2. Illustration of two new products of 3-order tensors.
Proposition 3.2. Let $\mathcal{A} \in \mathbb{R}^{I_1 \times J_1 \times K}$ and $\mathcal{B} \in \mathbb{R}^{K \times J_2 \times I_2}$ be two 3-order tensors, and $\mathcal{A} \in \mathbb{R}^{R_1 \times J_1}$ and $\mathcal{B} \in \mathbb{R}^{R_2 \times J_2}$ be two matrices. Then

$$(\mathcal{A} \times_2 \mathcal{A}) \boxtimes_2 (\mathcal{B} \times_2 \mathcal{B}) = (\mathcal{A} \boxtimes_2 \mathcal{B}) \times_2 (\mathcal{B} \otimes \mathcal{A}).$$

Proof. By Definitions 2.2, 2.4, and 3.1, upon some computations, we have

$$
((\mathcal{A} \boxtimes_2 \mathcal{B}) \times_2 (\mathcal{B} \otimes \mathcal{A}))(i_1, j_1 j_2, i_2) = \sum_{j_1 j_2 = 1}^{J_1 J_2} (\mathcal{A} \boxtimes_2 \mathcal{B})(i_1, j_1 j_2, i_2)(\mathcal{B} \otimes \mathcal{A})(j_1 j_2, i_1) \quad \text{by Definition 2.4}
$$

$$= \sum_{j_1 j_2 = 1}^{J_1 J_2} \sum_{k=1}^{J_1} \mathcal{A}(i_1, j_1, k) \mathcal{B}(k, j_2, i_2) \mathcal{B}(r_2, j_2) \mathcal{A}(r_1, j_1) \quad \text{by Definitions 2.2 and 3.1}
$$

$$= \sum_{k=1}^{K} \sum_{j_1 = 1}^{J_1} \mathcal{A}(i_1, j_1, k) \mathcal{B}(r_1, j_2) \mathcal{B}(r_2, j_2) \mathcal{A}(r_1, j_1) \quad \text{by Definition 2.4}
$$

$$= (\mathcal{A} \times_2 \mathcal{A}) \boxtimes_2 (\mathcal{B} \times_2 \mathcal{B})(i_1, j_1 j_2, i_2). \quad \text{by Definition 3.1}
$$

Thus, the proof is completed. $\blacksquare$

Based on the above findings, we can compute $\mathcal{S}\mathcal{G}^{[2]}_n$ without forming $\mathcal{S}$ and $\mathcal{G}^{[2]}_n$, explicitly by choosing a special $\mathcal{S}$. The details are presented in Section 4.

Furthermore, to facilitate the design of the algorithm by TensorSketch as in [20], we define the following product, whose graphical illustration is presented in Figure 2b.

Definition 3.3 (Slices-Hadamard product). Let $\mathcal{A} \in \mathbb{R}^{I_1 \times J \times K}$ and $\mathcal{B} \in \mathbb{R}^{K \times J \times I_2}$ be two 3-order tensors, and $\mathcal{A}(j)$ and $\mathcal{B}(j)$ are the $j$-th lateral slices of $\mathcal{A}$ and $\mathcal{B}$, respectively. The mode-2 slices-Hadamard product of $\mathcal{A}$ and $\mathcal{B}$ is a tensor of size $I_1 \times J \times I_2$ denoted by $\mathcal{A} \boxtimes_2 \mathcal{B}$ and defined as

$$(\mathcal{A} \boxtimes_2 \mathcal{B})(j) = \mathcal{A}(j) \mathcal{B}(j).$$

That is, the $j$-th lateral slice of $\mathcal{A} \boxtimes_2 \mathcal{B}$ is the classical matrix product of the $j$-th lateral slices of $\mathcal{A}$ and $\mathcal{B}$. The mode-1 and mode-3 slices-Hadamard product can be defined similarly.

Using Definition 2.2, similar to the proof of Proposition 3.2, we can prove the following result.

Proposition 3.4. Let $\mathcal{A} \in \mathbb{R}^{I_1 \times J_1 \times K}$ and $\mathcal{B} \in \mathbb{R}^{K \times J_2 \times I_2}$ be two 3-order tensors, and $\mathcal{A} \in \mathbb{R}^{M \times J_1}$ and $\mathcal{B} \in \mathbb{R}^{M \times I_2}$ be two matrices. Then

$$(\mathcal{A} \times_2 \mathcal{A}) \boxtimes_2 (\mathcal{B} \times_2 \mathcal{B}) = (\mathcal{A} \boxtimes_2 \mathcal{B}) \times_2 (\mathcal{B}^\top \otimes \mathcal{A}^\top)^\top.$$
Proposition 3.5. Let $\mathbf{S}_n = \Omega_n \mathbf{D}_n \in \mathbb{R}^{m \times I_n}$, where $\Omega_n \in \mathbb{R}^{m \times I_n}$ and $\mathbf{D}_n \in \mathbb{R}^{I_n \times I_n}$ are defined based on $H_n$ and $S_n$ in Definition 2.11, respectively. Let $\mathbf{T} \in \mathbb{R}^{m \times \prod_{n=1}^{N} I_n}$ be defined in Definition 2.11 and $\mathcal{P} = \mathcal{A}^{(1)} \boxtimes_2 \mathcal{A}^{(2)} \boxtimes_2 \ldots \boxtimes_2 \mathcal{A}^{(N)}$ with $\mathcal{A}^{(n)} \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}$ for $n \in [N]$. Then

$$\mathcal{P} \times_2 \mathbf{T} = \text{FFT}^{-1} \left( \bigotimes_{n=1}^{N} \text{FFT} \left( \mathcal{A}^{(n)} \times_2 \mathbf{S}_n, [1], 2 \right), [1], 2 \right).$$

Proof. For simplicity, we only present the proof for $N = 2$ here. Extending the proof to the general case is immediate.

On the one hand, the mode-2 products of $\mathcal{A}^{(1)}$ with $\mathbf{S}_1$ and of $\mathcal{A}^{(2)}$ with $\mathbf{S}_2$ can be represented by the following two $m - 1$ degree matrix polynomials

$$\mathcal{P}^{(1)}(\omega) = \sum_{i_1=1}^{I_1} S_1(i_1) \mathcal{A}^{(1)}(:, i_1, :)\omega^{H_1(i_1) - 1} = \sum_{m'=1}^{m} c^{(1)}_{m'} \omega^{m'-1} \in \mathbb{R}^{R_1 \times R_2},$$

$$\mathcal{P}^{(2)}(\omega) = \sum_{i_2=1}^{I_2} S_2(i_2) \mathcal{A}^{(2)}(:, i_2, :)\omega^{H_2(i_2) - 1} = \sum_{m'=1}^{m} c^{(2)}_{m'} \omega^{m'-1} \in \mathbb{R}^{R_2 \times R_3},$$

where $c^{(1)}_{1}, \ldots, c^{(1)}_{m} \in \mathbb{R}^{R_1 \times R_2}$ and $c^{(2)}_{1}, \ldots, c^{(2)}_{m} \in \mathbb{R}^{R_2 \times R_3}$ are coefficient matrices, and $\omega$ satisfies $\omega^m = 1$ but $\omega^{m'} \neq 1$ for $m' = 1, \ldots, m - 1$.

On the other hand, $\mathcal{P} \times_2 \mathbf{T}$ can be represented as the following matrix polynomial

$$(3.4) \quad \mathcal{P}(\omega) = \sum_{i=1}^{I_1 I_2} S(i) \mathcal{P}(\cdot, i, :)\omega^{H(i) - 1} = \sum_{m'=1}^{m} c_{m'} \omega^{m'-1} \in \mathbb{R}^{R_1 \times R_3},$$

where $c_{1}, \ldots, c_{m} \in \mathbb{R}^{R_1 \times R_3}$ are coefficient matrices, $i = i_1 i_2$ and $\omega$ satisfies the constraints mentioned above.

Note that $\mathcal{P} = \mathcal{A}^{(1)} \boxtimes_2 \mathcal{A}^{(2)} \in \mathbb{R}^{R_1 \times I_1 I_2 \times R_3}$ with $\mathcal{A}^{(n)} \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}$ for $n = 1, 2$. Then, according to Definition 3.1, we have

$$(3.5) \quad \mathcal{P}(\cdot, i, :) = \mathcal{A}^{(1)}(:, i_1, :)\mathcal{A}^{(2)}(:, i_2, :),$$

where $i \in [I_1 I_2], i_1 \in [I_1]$, and $i_2 \in [I_2]$. Substituting (3.5) into (3.4) implies

$$\mathcal{P}(\omega) = \sum_{i_1 i_2=1}^{I_1 I_2} S(i_1 i_2) \mathcal{A}^{(1)}(:, i_1, :)\mathcal{A}^{(2)}(:, i_2, :)\omega^{H(i_1 i_2) - 1},$$

which together with Definition 2.11 and the constraints on $\omega$ mentioned above gives

$$\mathcal{P}(\omega) = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} S_1(i_1) S_2(i_2) \mathcal{A}^{(1)}(:, i_1, :)\mathcal{A}^{(2)}(:, i_2, :)\omega^{H_1(i_1) + H_2(i_2) - 2 \mod m}$$

$$\mathcal{P}(\omega) = \sum_{i_1=1}^{I_1} S_1(i_1) \mathcal{A}^{(1)}(:, i_1, :)\omega^{H_1(i_1) - 1} \sum_{i_2=1}^{I_2} S_2(i_2) \mathcal{A}^{(2)}(:, i_2, :)\omega^{H_2(i_2) - 1}$$

$$(3.6) \quad = \mathcal{P}^{(1)}(\omega) \mathcal{P}^{(2)}(\omega).$$
Thus, viewing
\[ c_1, \ldots, c_m; \quad c_1^{(1)}, \ldots, c_m^{(1)}; \quad c_1^{(2)}, \ldots, c_m^{(2)} \]
as the lateral slices of the tensors \( C, C^{(1)}, \) and \( C^{(2)}, \) respectively, according to the famous convolution theorem, we can see that (3.6) can be represented as
\[ C = \text{FFT}^{-1} \left( \text{FFT}(C^{(1)}, |, 2) \boxtimes_2 \text{FFT}(C^{(2)}, |, 2), |, 2 \right). \]

More precisely, we have
\[ \mathcal{P} \times_2 T = \text{FFT}^{-1} \left( \text{FFT} \left( A^{(1)} \times_2 S_1, |, 2 \right) \boxtimes_2 \text{FFT} \left( A^{(2)} \times_2 S_2, |, 2 \right), |, 2 \right). \]

This is the desired result.

4. KSRFT-based randomized TR decomposition. In this section, we present the randomized algorithm of TR decomposition based on KSRFT and discuss its theoretical analysis and computational complexity.

4.1. Algorithms. According to the analyses in Section 3 and inspired by [4], we choose \( S \in \mathbb{R}^{m \times \prod_{j \neq n} I_j} \) as\(^2\)
\begin{equation}
S = \text{SFDG},
\end{equation}
where \( F = \bigotimes_{N, \ldots, n+1} F_j \) and \( D = \bigotimes_{N, \ldots, n+1} D_j \) with \( F_j \in \mathbb{R}^{I_j \times I_j} \) and \( D_j \in \mathbb{R}^{I_j \times I_j} \) being defined in Definition 2.9, and the meaning of \( S \) will be clarified later. Thus, the reduced problem of (3.1) is
\begin{equation}
\arg \min_{G_{(2)}} \left\| \text{SFDG}^{[2]}_{\frac{n}{n}} G_{(2)}^\top - \text{SFDG}^\top_{[n]} \right\|_F.
\end{equation}

From the discussions in Section 3, we know that finding \( \text{FDG}^{[2]}_{\frac{n}{n}} \) can be transformed into finding \( \mathcal{G}^{\frac{n}{n}} \times_2 (\text{FD}) \). While, using (3.2) and Proposition 3.2, we have
\begin{equation}
\mathcal{G}^{\frac{n}{n}} = \mathcal{G}^{\frac{n}{n}} \times_2 (\text{FD}) = \mathcal{G}^{\frac{n}{n}} \times_2 \left( \left( \bigotimes_{n-1, \ldots, n+1} F_j \right) \left( \bigotimes_{n-1, \ldots, n+1} D_j \right) \right)
= (\mathcal{G}_{n+1} \times_2 (F_{n+1} D_{n+1})) \boxtimes_2 \cdots \boxtimes_2 \left( \mathcal{G}_N \times_2 (F_N D_N) \right)
\boxtimes_2 \left( \mathcal{G}_1 \times_2 (F_1 D_1) \right) \boxtimes_2 \cdots \boxtimes_2 \left( \mathcal{G}_{n-1} \times_2 (F_{n-1} D_{n-1}) \right).
\end{equation}

As a result,
\[ \text{FDG}^{\frac{n}{n}}_{[2]} = \mathcal{G}^{\frac{n}{n}}_{[2]}. \]

\(^2\)From Definition 2.9, there should be a scaling factor, i.e., \( \sqrt{\frac{1}{\prod_{j \neq n} I_j}} \), in the right side of (4.1). However, it has no effect on the solution. So we omit it here. In addition, there should also be a subscript, e.g., \( \neq n \), in the matrices \( S, S, F, \) and \( D \). For simplicity, we also omit it here.
Thus, by mixing the lateral slices of each core $G_j$ via the small matrices $F_j$ and $D_j$, we get $FDG_{ij}^{[n]}$, which can avoid forming the large matrices $F$, $D$, and $G_{ij}^{[n]}$ explicitly.

Note that the second term in (4.2) is equivalent to

$$S \hat{X}^\top_{[n]} (D_n F_n^*)^\top,$$

where $\hat{X} = \mathcal{X}_1 (F_1 D_1) \times (F_2 D_2) \cdots \times (F_N D_N)$, i.e., $\hat{X}_{[n]} = F_n D_n X_{[n]} (F_{n-1} D_{n-1}) \cdots \times (F_{n+1} D_{n+1})^\top$, and the superscript "*" denotes the conjugate transpose. As explained in [4], using the within the loop body and hence can significantly improve the computational efficiency.

Thus, our reduced problem has the following form

$$(4.4) \quad \arg \min_{G_{n(2)}} \| (S \hat{G}_{[2]}^{\neq n}) G_{n(2)}^\top - S \hat{X}_{[n]}^\top (D_n F_n^*)^\top \|_F.$$

To avoid sampling the rows of $\hat{G}_{[2]}^{\neq n}$, i.e., the lateral slices of the large mixed subchain tensor $G^{\neq n}$ directly, we use the sampling strategy in Algorithm 3.2. That is, we sample the lateral slices of the $N - 1$ cores individually and then form the sampled mixed subchain tensor $\hat{G}_{[2]}^{\neq n} \times 2 S$. This process can be expressed as follows:

$$\hat{G}_{[2]}^{\neq n} \times 2 S = ((G_{n+1} \times 2 (F_{n+1} D_{n+1})) \boxtimes_2 \cdots \boxtimes_2 (G_N \times 2 (F_N D_N))) \boxtimes_2 (G_{n-1} \times 2 (F_{n-1} D_{n-1})) \boxtimes_2 S \nonumber$$

$$= (G_{n+1} \times 2 (S_{n+1} F_{n+1} D_{n+1})) \boxtimes_2 \cdots \boxtimes_2 (G_N \times 2 (S_N F_N D_N)) \boxtimes_2 (G_{n-1} \times 2 (S_{n-1} F_{n-1} D_{n-1})), \nonumber$$

where $S_j \in \mathbb{R}^{m \times I_j}$ is the sampling matrix for the $j$-th core. According to Proposition 3.4, we can see that $S = \left( \bigoplus_{n=1}^{N} S_j^{[n]} \right)^\top$. Thus, using the property of Khatri-Rao product, we have

$$(4.6) \quad S = \left( \bigoplus_{n=1}^{N} S_j^{[n]} \right)^\top \left( \bigotimes_{n=1}^{N} (S_j F_j D_j) \right)^\top = \left( \bigoplus_{n=1}^{N} S_j F_j D_j \right)^\top,$$

and hence we can also call it the Khatri-Rao SRFT. Note that the $S$ used in [4] has the same structure as $S$ in (4.6) though the structure was not provided explicitly there.

The above discussions imply Algorithm 4.1.

Furthermore, note that, with the property of the Frobenius norm, the reduced problem (4.4) can be rewritten as

$$\arg \min_{G_{n(2)}} \| (S \hat{G}_{[2]}^{\neq n}) G_{n(2)}^\top - S \hat{X}_{[n]}^\top \|_F. \nonumber$$

Thus, setting $\hat{G}_{n(2)} = F_n D_n G_{n(2)}$, we have

$$(4.7) \quad \arg \min_{\hat{G}_{n(2)}} \| (S \hat{G}_{[2]}^{\neq n}) \hat{G}_{n(2)}^\top - (S \hat{X}_{[n]}^\top) \|_F.$$
Algorithm 4.1 TR-KSRFT-ALS (Proposal)

1: function \( \{ \mathcal{G}_n \}_{n=1}^{N} = \text{TR-KSRFT-ALS}(\mathcal{X}, R_1, \ldots, R_N, m) \)
\[ \triangleright \mathcal{G}_n \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}, n \in [N]; \mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \]
\[ \triangleright R_1, \ldots, R_N \text{ are the TR-ranks} \]
\[ \triangleright m \text{ is the uniform sampling size} \]

2: Initialize cores \( \mathcal{G}_2, \ldots, \mathcal{G}_N \)
3: Define random sign-flip operators \( \mathbf{D}_j \) and FFT matrices \( \mathbf{F}_j \), for \( j \in [N] \)
4: Mix cores: \( \hat{\mathcal{G}}_n \leftarrow \mathcal{G}_n \times_2 (\mathbf{F}_1 \mathbf{D}_n) \), for \( n = 2, \ldots, N \)
5: Mix tensor: \( \mathcal{X} \leftarrow \mathcal{X} \times_1 (\mathbf{F}_1 \mathbf{D}_1) \times_2 (\mathbf{F}_2 \mathbf{D}_2) \cdots \times_N (\mathbf{F}_N \mathbf{D}_N) \)
6: repeat
7: \hspace{1em} for \( n = 1, \ldots, N \) do
8: \hspace{2em} Define sampling operator \( \mathbf{S} \in \mathbb{R}^{m \times \prod_{j \notin \tau_j}} \)
9: \hspace{2em} Retrieve \( \text{idxs} \) from \( \mathbf{S} \)
10: \hspace{2em} \( \tilde{\mathcal{G}}_S^{\neq n} = \text{SST}(\text{idxs}, \hat{\mathcal{G}}_{n+1}, \ldots, \hat{\mathcal{G}}_1, \ldots, \hat{\mathcal{G}}_{n-1}) \)
11: \hspace{2em} \( \hat{\mathbf{X}}_{S[n]}^\dagger \leftarrow \text{S} \hat{\mathbf{X}}_{S[n]}^\dagger (\mathbf{D}_n \mathbf{F}_n^*) \)
12: \hspace{2em} Update \( \hat{\mathcal{G}}_n = \arg \min_{\mathcal{Z}} \| \hat{\mathcal{G}}_{S[2]}^{\neq n} \mathbf{Z}^7 - \hat{\mathbf{X}}_{S[n]}^\dagger \| F \) subject to \( \hat{\mathcal{G}}_n \) being real-valued
13: \hspace{2em} \( \hat{\mathcal{G}}_n \leftarrow \hat{\mathcal{G}}_n \times_2 (\mathbf{F}_n \mathbf{D}_n) \)
14: \hspace{1em} end for
15: until termination criteria met
16: return \( \hat{\mathcal{G}}_1, \ldots, \hat{\mathcal{G}}_N \)
17: end function

Consequently, we can solve the above problem to get \( \hat{\mathcal{G}}_n \) first and then recover the original core \( \mathcal{G}_n \) from it. The specific algorithm is summarized in Algorithm 4.2.

Algorithm 4.2 TR-KSRFT-ALS-Premix (Proposal)

1: function \( \{ \mathcal{G}_n \}_{n=1}^{N} = \text{TR-KSRFT-ALS-PREMIX}(\mathcal{X}, R_1, \ldots, R_N, m) \)
\[ \triangleright \mathcal{G}_n \in \mathbb{C}^{R_n \times I_n \times R_{n+1}}, n \in [N]; \mathcal{X} \in \mathbb{C}^{I_1 \times \cdots \times I_N} \]
\[ \triangleright R_1, \ldots, R_N \text{ are the TR-ranks} \]
\[ \triangleright m \text{ is the uniform sampling size} \]

2: Define random sign-flip operators \( \mathbf{D}_j \) and FFT matrices \( \mathbf{F}_j \), for \( j \in [N] \)
3: Mix tensor: \( \mathcal{X} \leftarrow \mathcal{X} \times_1 (\mathbf{F}_1 \mathbf{D}_1) \times_2 (\mathbf{F}_2 \mathbf{D}_2) \cdots \times_N (\mathbf{F}_N \mathbf{D}_N) \)
4: Initialize cores \( \hat{\mathcal{G}}_2, \ldots, \hat{\mathcal{G}}_N \)
5: repeat
6: \hspace{1em} for \( n = 1, \ldots, N \) do
7: \hspace{2em} Define sampling operator \( \mathbf{S} \in \mathbb{R}^{m \times \prod_{j \notin \tau_j}} \)
8: \hspace{2em} Retrieve \( \text{idxs} \) from \( \mathbf{S} \)
9: \hspace{2em} \( \tilde{\mathcal{G}}_S^{\neq n} = \text{SST}(\text{idxs}, \hat{\mathcal{G}}_{n+1}, \ldots, \hat{\mathcal{G}}_1, \ldots, \hat{\mathcal{G}}_{n-1}) \)
10: \hspace{2em} \( \hat{\mathbf{X}}_{S[n]}^\dagger \leftarrow \text{S} \hat{\mathbf{X}}_{S[n]}^\dagger (\mathbf{D}_n \mathbf{F}_n^*) \)
11: \hspace{2em} Update \( \hat{\mathcal{G}}_n = \arg \min_{\mathcal{Z}} \| \hat{\mathcal{G}}_{S[2]}^{\neq n} \mathbf{Z}^7 - \hat{\mathbf{X}}_{S[n]}^\dagger \| F \)
12: \hspace{2em} end for
13: \hspace{1em} until termination criteria met
14: \hspace{1em} for \( n = 1, \ldots, N \) do
15: \hspace{2em} Unmix cores: \( \hat{\mathcal{G}}_n \leftarrow \hat{\mathcal{G}}_n \times_2 (\mathbf{D}_n \mathbf{F}_n^*) \)
16: \hspace{1em} end for
17: \hspace{1em} return \( \hat{\mathcal{G}}_1, \ldots, \hat{\mathcal{G}}_N \)
18: end function

Remark 4.1. Algorithms 4.1 and 4.2 are inspired by [4], which designed some practical randomized algorithms for CP decomposition. One of our main contributions is that we
present an elegant way to transform the problem (3.1) to the problem (4.4) with the new tensor product and its property. This allows us to design algorithms as elegantly as in [4]. Compared with the method in [25], which is mainly inspired by the work in [19], our method may work better for some special data, such as for the data whose TR-cores include outliers\(^3\) [16, 15]. This is mainly because, with the sampling method in Algorithm 3.2, TR-ALS-Sampled may sample the aforementioned outliers with high probability many times and hence may cause an unstable result. Although we also use Algorithm 3.2, the sampling in our method is uniform sampling and the tensor is transformed by the randomized Fourier transform before sampling. The above analysis is supported by the numerical results of the third experiment on synthetic data in Section 6.

**Remark 4.2.** Upon close examination of the derivation of Algorithm 4.1, we may let \(F_jD_j\) be any suitable randomized matrices. For example, we can set them to be the CountSketch matrices for sparse input data. The corresponding method can be denoted by TR-CS-ALS. Moreover, we are surprised to find that if we set them to be the transpose of the factor matrices appearing in the randomized Tucker decomposition, we can recover the algorithm rTR-ALS proposed in [34]. These analyses present an affirmative result that reducing the dimension of each mode of the original tensor with a suitable sketching matrix, finding the TR decomposition of the reduced tensor, and then recovering the TR decomposition of the original tensor is feasible. Note that, in these cases, the step on sampling is not necessary and hence the final product is the subchain product as in (4.3) but not the slices-Hadamard product as in (4.5). The differences are illustrated in Figure 3, from which we can also find that the sketching-based methods for TR decomposition can be mainly classified into two classes: one is based on the subchain product, and the other is based on the slices-Hadamard product. We also summarize all the algorithms mentioned above in Table 1.

**Table 1**

| Algorithms               | \(S_j\) in Figure 3 | \(S\) in Figure 3 | Product                     |
|-------------------------|----------------------|-------------------|-----------------------------|
| rTR-ALS ([34])          | \(Q_j^t\) from the QR decomposition of \(X_{[j]}M\) with \(M\) being a Gaussian matrix | \(\bigotimes_{N=n+1}^{n-1} Q_j^t\) | Subchain product             |
| TR-ALS-Sampled ([25])   | sampling matrix via leverage score | \(\bigotimes_{N=n+1}^{n-1} (S_j)^\dagger\) | Slices-Hadamard product      |
| TR-KSRFT-ALS (Proposal) | \(S_jF_jD_j\)        | \(\bigotimes_{N=n+1}^{n-1} (S_jF_jD_j)^\dagger\) | Slices-Hadamard product      |
| TR-TS-ALS (Proposal)    | CountSketch          | TensorSketch      | Slices-Hadamard product      |
| TR-KSRFT-ALS without sampling | \(F_jD_j\)          | \(\bigotimes_{N=n+1}^{n-1} F_jD_j\) | Subchain product              |
| TR-CS-ALS               | CountSketch          | \(\bigotimes_{N=n+1}^{n-1} S_j\) | Subchain product              |

**Remark 4.3.** As done in [4], to get the real-valued TR-cores from the real-valued input...
Figure 3. Illustration of how to efficiently construct $G^{\neq n} \otimes 2$ by sketching the core tensors.

tensor, we should solve the following equivalent form of Line 12 in Algorithm 4.1:

$$G_n = \arg \min_{Z} \left\| \begin{bmatrix} R(G^{\neq n}[2]) \\ J(G^{\neq n}[2]) \end{bmatrix} Z^{(2)} - \begin{bmatrix} R(\tilde{X}^T[S_{[n]}]) \\ J(\tilde{X}^T[S_{[n]}]) \end{bmatrix} \right\|_F.$$  

Of course, if the matrices $F_j$ used in the algorithm are real-valued orthogonal, this process can be ignored. For Algorithm 4.2, it is very applicable to finding the complex-valued core tensors from the complex-valued input tensor.

4.2. Theoretical analysis. Since the problem we focus on is essentially a sketched least squares problem like the one in [4, 17], we can apply Proposition 2.1 in [17] to our method. The results are given as follows.

Theorem 4.4 (A slight restatement of Proposition 2.1 in [17]). For the matrices $G^{\neq n}_{[2]}$ and $X^\top_{[n]}$ in (3.1), denote $\text{rank}(G^{\neq n}_{[2]}) = r$ and fix $\varepsilon, \eta \in (0, 1)$ such that $\prod_{j \neq n} I_j \lesssim 1/\varepsilon^r$ with $r \geq 2$. 
Then a sketching matrix $S \in \mathbb{C}^{m \times \prod_{j \neq n} I_j}$ used in Algorithms 4.1 or 4.2, i.e., (4.6) with

$$m = O \left( \frac{1}{\varepsilon} r^2(N-1) \log^{2N-3} \left( \frac{r}{\varepsilon} \right) \log^4 \left( \frac{r}{\varepsilon} \log \left( \frac{r}{\varepsilon} \right) \right) \log \prod_{j \neq n} I_j \right)$$

is sufficient to output

$$\tilde{G}^T_{n(2)} = \arg \min_{G^T_{n(2)} \in \mathbb{R}^{R_n I_{n+1} \times I_n}} \| S G^T_{[2]} G^T_{n(2)} - SX^T_{[n]} \|_F,$$

such that

$$\Pr \left( \| G^T_{[2]} \tilde{G}^T_{n(2)} - X^T_{[n]} \|_F = (1 \pm O(\varepsilon)) \min \| G^T_{[2]} G^T_{n(2)} - X^T_{[n]} \|_F \right) \geq 1 - \eta - 2^{-\Omega \left( \log \prod_{j \neq n} I_j \right)}.$$

Remark 4.5. It is worth clarifying that the algorithms rTR-ALS, TR-KSRFT-ALS without sampling and TR-CS-ALS listed in Table 1 aren’t covered by the guarantees in the above theorem. This is because there is no step on sampling in these methods.

4.3. Computational complexity. We first analyze the computational complexity of Algorithm 3.1 because our algorithms are based on TR-ALS. For simplicity, we assume that $I_n = I$ and $R_n = R$ for all $n \in [N]$ and that $N < I$, $R^2 < I$ and $NR < I$. And we ignore any cost associated with e.g. normalization and checking termination conditions.

Upfront costs of TR-ALS:

- **Line 2**: Initializing cores. This depends on how to initiate the cores. We assume that they are randomly drawn, e.g. from a Gaussian distribution, resulting in a cost $O(NIR^2)$.

The costs of per outer loop iteration in TR-ALS:

- **Line 5**: Computing the unfolding subchain tensor. If the $N - 1$ cores are dense and contracted in sequence, the cost is

$$R^3 (I^2 + I^3 + \cdots + I^{N-1}) \leq R^3 (NI^{N-2} + I^{N-1}) \leq 2R^3 I^{N-1} = O \left( I^{N-1} R^3 \right).$$

Doing this for each of the $N$ cores in the inner loop brings the cost to $O \left( N I^{N-1} R^3 \right)$.

- **Line 6**: Solving the least squares problem. We use the standard QR-based approach to analyze the computational complexity. Doing a QR decomposition of $G^T_{[2]} \in \mathbb{R}^{I_{N-1} \times R^2}$ costs $O \left( I^{N-1} R^4 \right)$, and updating the right hand side and doing back substitution costs $O \left( I (I^{N-1} R^2 + R^4) \right) = O \left( I N R^2 \right)$. The leading order complexity for solving the least squares problem is therefore $O \left( I N R^2 \right)$. Doing this for each of the $N$ cores in the inner loop brings the cost to $O \left( N I N R^2 \right)$.

Putting them all together, we have that the leading order complexity of TR-ALS is

$$O \left( N I R^2 + it \cdot N I N R^2 \right),$$

where “$it$” denotes the number of outer loop iterations.

Now, we present the computational complexity of Algorithm 4.1.

Upfront costs of TR-KSRFT-ALS:
• **Line 2:** Initializing cores. It is the same as TR-ALS, which costs $O(NIR^2)$.
• **Line 4:** Mixing cores. It costs $O((N - 1)I\log I)$ because there are $N - 1$ cores.
• **Line 5:** Mixing tensor. It requires a significant upfront cost, which is $O(I^N\log(I^N))$.

The costs of per outer loop iteration in TR-KSRFT-ALS:
• **Line 8:** Generating $N$ sampling matrices. It costs $O(mN)$. Actually, these matrices are not generated explicitly in specific implementation.
• **Line 9-Line 10:** Computing the sampled subchain tensor. The main cost is the matrix multiplication of $N - 1$ matrices of size $R \times R$ for each of the $m$ sampled mode-2 slices, which costs $O(mR^3N)$ per inner loop iteration, i.e., $O(mR^3N^2)$ per outer loop iteration.
• **Line 11:** Sampling the input tensor. It needs to copy $O(mI)$ elements from the input tensor and then do matrix multiplication. So the cost is $O(mI^2)$ per inner loop iteration, i.e., $O(NmI^2)$ per outer loop iteration.
• **Line 12:** Solving the least squares problem. Similar to TR-ALS, it costs $O(ImR^2)$ per inner loop iteration, i.e., $O(NImR^2)$ per outer loop iteration.
• **Line 13:** Update the mixed cores. It costs $O(I\log I)$ per inner loop iteration, i.e., $O(NI\log I)$ per outer loop iteration.

It follows that the overall leading order complexity of TR-KSRFT-ALS is

$$O(I^N\log(I^N) + it \cdot NmI^2),$$

where “$it$” denotes the number of outer loop iterations.

For the computational complexity of TR-KSRFT-ALS-Premix, the analysis is similar. Next we only show the difference:
• **Line 15:** Unmixing the cores. It costs $O(NI\log I)$ because there are $N$ cores.

So the overall leading order complexity of TR-KSRFT-ALS-Premix is

$$O(I^N\log(I^N) + it \cdot NImR^2),$$

where “$it$” denotes the number of outer loop iterations.

**Remark 4.6.** Like Algorithms 4 and 5 in [4], the dominant cost of TR-KSRFT-ALS and TR-KSRFT-ALS-Premix also appears in the upfront costs, i.e., the mixing tensor step. Without considering this cost, i.e., we assume that the data has been preprocessed, the complexities of TR-KSRFT-ALS and TR-KSRFT-ALS-Premix can be reduced remarkably when $I$ and $N$ are huge.

5. **TS-based randomized TR decomposition.** In this section, we set $S$ to be the TensorSketch $T_{\neq n} \in \mathbb{R}^{m \times \prod_{j\neq n} I_j}$ defined as in Definition 2.11 with $j = 1, \ldots, n - 1, n + 1, \ldots, N$. Thus, from the discussions in Sections 3 and 4, especially (3.2) and Proposition 3.5, we can propose the following Algorithm 5.1 for TR decomposition based on the problem (3.1), and the Line 9 is illustrated in Figure 3 and Table 1.

**Remark 5.1.** Algorithm 5.1 is inspired by the work in [20], which proposes a fast and accurate sketched ALS algorithm for Tucker decomposition by using TensorSketch. Our main contribution is that we find the graceful formula in Proposition 3.5 using the new tensor product and its property. With it, we can design the algorithm as efficiently as in [20].
Algorithm 5.1 TR-TS-ALS (Proposal)

1: function $\{G_n\}_{n=1}^N = \text{TR-TS-ALS}(X, R_1, \ldots, R_N, m)$
   
   $\triangleright G_n \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}, n \in [N]; X \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
   
   $\triangleright R_1, \ldots, R_N$ are the TR-ranks
   
   $\triangleright m$ is the embedding size

2: Define $S_j$, i.e., the CountSketch, based on $H_n$ and $S_n$ in Definition 2.11, for $j \in [N]$
3: for $n = 1, \ldots, N$ do
4:   Compute the sketch of $X^\top_{[n]}$: $\hat{X}^\top_{[n]} \leftarrow T_{\neq n}X^\top_{[n]}$
5: end for
6: Initialize cores $G_2, \ldots, G_N$
7: repeat
8:   for $n = 1, \ldots, N$ do
9:     Compute $\hat{G}^{\neq n} = \text{FFT}^{-1} \left( \oplus_{j=n+1}^{n-1} \text{FFT}(G_j \times S_j, [1, 2], [1, 2]) \right)$
10:    Update $G_n = \arg\min_{\mathcal{F}} \|G^{\neq n} Z^{(2)}_{(2)} - \hat{X}^\top_{[n]}\|_F$
11: end for
12: until termination criteria met
13: return $G_1, \ldots, G_N$
14: end function

Similar to Theorem A.7 in [20], we have the following theoretical sketch size for achieving $O(\epsilon)$-relative error.

Theorem 5.2 (A slight restatement of Theorem A.7 in [20]). For the matrices $G_{[2]}^{\neq n}$ and $X^\top_{[n]}$ in (3.1), fix $\epsilon, \eta \in (0, 1)$. Then a TensorSketch $T_{\neq n}$ used in Algorithm 5.1 with

$$m = O \left( (R_n R_{n+1} \cdot 3^{N-1})(R_n R_{n+1} + 1/\epsilon^2)/\eta \right)$$

is sufficient to output

$$\hat{G}^{\top}_{n(2)} = \arg\min_{G^{\top}_{n(2)} \in \mathbb{R}^{R_n \times I_n}} \|T_{\neq n} G^{\neq n} G^{\top}_{n(2)} - T_{\neq n} X^\top_{[n]}\|_F,$$

such that

$$\Pr \left( \|G^{\neq n} G^{\top}_{n(2)} - X^\top_{[n]}\|_F = (1 \pm O(\epsilon)) \min \|G^{\neq n} G^{\top}_{n(2)} - X^\top_{[n]}\|_F \right) \geq 1 - \eta.$$
• **Line 10:** **Solving the least squares problem.** Similar to TR-ALS, it costs $O(1mR^2)$ per inner loop iteration, i.e., $O(NImR^2)$ per outer loop iteration. As a result, the overall leading order complexity of TR-TS-ALS is 

$$O\left(Nnnz(X) + it \cdot NImR^2\right),$$

where “$it$” denotes the number of outer loop iterations. So, TR-TS-ALS is very applicable to sparse tensor. For dense tensor, similar to TR-KSRFT-ALS and TR-KSRFT-ALS-Premix, the dominant cost of Algorithm 5.1 also appears in the upfront costs, i.e., the Line 4.

**Remark 5.3.** The leading order computational complexities of the algorithms involved in this paper are summarised in Table 2.

| Method                        | Complexity                                  | Ignoring mixing tensor |
|-------------------------------|---------------------------------------------|------------------------|
| TR-ALS ([35])                | $O(NIR^2 + it \cdot NImR^2)$               | ——                     |
| TR-ALS-Sampled ([25])        | $O(NIR^2 + it \cdot NImR^2)$               | ——                     |
| TR-SRFT-ALS (Proposal)       | $O(IN \log(IN) + it \cdot NmI^2)$         | $O(NIR^2 + it \cdot NImR^2)$ |
| TR-SRFT-ALS-Premix (Proposal)| $O(IN \log(IN) + it \cdot NImR^2)$         | $O(NIR^2 + it \cdot NImR^2)$ |
| TR-TS-ALS (Proposal)         | $O(Nnnz(X) + it \cdot NImR^2)$             | ——                     |

As before, “$it$” in this table denotes the number of outer loop iterations. From Table 2, we can find that the complexities of TR-SRFT-ALS and TR-SRFT-ALS-Premix are lower than that of the regular TR-ALS, but are higher compared with TR-ALS-Sampled. However, if we don’t consider the complexity of mixing the tensor, our methods will have some advantages over TR-ALS-Sampled. For TR-TS-ALS, its complexity can be lower than that of TR-ALS-Sampled for sparse tensor.

**6. Numerical results.** To test our proposed methods, we choose two methods as the baselines. The first one is TR-ALS, with which we want to show the advantages of the randomized algorithms for TR decomposition for big data. The second one is TR-ALS-Sampled from [25] because it is the state-of-the-art randomized method for TR decomposition; see the detailed comparison of this method with others given in [25]. Meanwhile, we have also used the same sampling strategy from TR-ALS-Sampled in TR-KSRFT-ALS and TR-KSRFT-ALS-Premix. For TR-ALS and TR-ALS-Sampled, the functions `tr_als.m` and `tr_als_sampled.m` from [25] are used in the specific experiments. These two functions are available at [https://github.com/OsmanMalik/tr-als-sampled](https://github.com/OsmanMalik/tr-als-sampled). Additionally, we also use the MATLAB Tensor Toolbox [3].

For the initialization of the involved algorithms, as discussed in [4] on ALS and randomized ALS for CP decomposition, TR-ALS also requires a good starting point to ensure good performance, but the randomized variants gain no obvious advantage from the same initial guess. So, in our experiments, we use random initialization as mentioned in the analysis of computational complexity for randomized algorithms. For TR-ALS, we can choose the results of TR-SVD as the initialization. However, for fairness, we also use random initialization.

In addition, all experiments are run on Matlab R2020b on a computer with an Intel Xeon W-2255 3.7 GHz CPU and 256 GB memory, and all numerical results and plotted quantities are the average over 10 runs.
6.1. Synthetic data. Our experimental method for synthetic tensors can be summarized in the following two stages.

**Preparation Stage.** In this stage, we use TR-ALS to determine the maximum number of iterations for termination of various algorithms, which will be used as the only termination criterion in **Experimental Stage.** Specifically, we run TR-ALS until the iterations are larger than \( M = 500 \) or the relative error is smaller than \( \varepsilon = 1 \times 10^{-6} \) and record the iterations \( T \). Here, the relative error is defined by

\[
\frac{\| \text{TR} \left( \{ G'_n \}_{n=1}^N \right) - X \|_F}{\| X \|_F},
\]

where the TR-cores \( G'_n \) are computed by TR-ALS. This relative error is also used to measure the quality of various randomized algorithms for TR decomposition in the subsequent experiments. This is because we have no way to know the true cores in most applications.

**Experimental Stage.** In this stage, we use \( 2T \) as the termination criterion to run TR-ALS/TR-ALS-Sampled/TR-KSRFT-ALS/TR-TS-ALS. For TR-ALS-Sampled/TR-KSRFT-ALS/TR-TS-ALS, we set the embedding size \( m \) to be started at \( J_{\text{init}} \) and incremented by \( J_{\text{inc}} \) until \( J_{\text{fin}} \). In the specific experiments, we set \( J_{\text{init}} = 500 \), \( J_{\text{inc}} = 250 \), and \( J_{\text{fin}} = 5000 \). Thus, we can obtain the corresponding relative errors and running time at each fixed embedding size, and the vectors of relative errors and running time for all the embedding sizes. As for TR-ALS, it doesn’t need the embedding size and its relative error and running time are almost fixed in each experiment. Therefore, with respect to the increase of embedding size, we can plot figures to reflect the variation of relative errors and running time of different algorithms.

Four numerical experiments are carried out to test our methods. All of our synthetic tensors of size \( I \times I \times I \) are generated by creating 3 cores of size \( R_{\text{true}} \times I \times R_{\text{true}} \), i.e., \( G_1, G_2 \) and \( G_3 \), and hence the tensor we are trying to recover is \( X_{\text{true}} = \text{TR} \left( \{ G_n \}_{n=1}^3 \right) \). In the specific experiments, we set \( R_{\text{true}} = 10 \) and use \( R \) to denote the target rank run in the algorithms. To show the robustness and scalability of algorithms, the noise tensor \( N \in \mathbb{R}^{I \times I \times I} \) with entries drawn from a standard normal distribution is added into the true tensor. Then the observed tensor is of the following form

\[
X = X_{\text{true}} + \text{noise} \left( \frac{\| X_{\text{true}} \|}{\| N \|} \right) N,
\]

where the parameter \( \text{noise} \) is the amount of noise. For the sparse tensor, we will only add the noise into the non-zero entries.

In the following, we present the specific way to generate the data and the corresponding numerical results for each of the four experiments.

In the first experiment, we generate a \( 500 \times 500 \times 500 \) tensor as done in [25]. Specifically, 3 cores of size \( 10 \times 500 \times 10 \) with entries drawn independently from a standard normal distribution are first created and then one entry for each core is chosen uniformly and set to 20 to increase the coherence. Finally, the true tensor can be formed using TR decomposition.

Figure 4 shows the numerical results on relative errors and running time for this data with or without noise. From the relative errors shown in the top half of the figure, we can see
that TR-KSRFT-ALS and TR-TS-ALS match, and in some cases surpass, the performance of TR-ALS-Sampled, and when the embedding size increases to a certain value, these three methods can achieve similar errors as TR-ALS. For the running time shown in the bottom half of the figure, our two methods and TR-ALS-Sampled have almost the same performance for the case with noise, and are much better than TR-ALS. For the case without noise, our methods perform a little worse than TR-ALS-Sampled, however, still significantly outperform TR-ALS. In addition, when the noise increases, our algorithms are still efficient, which implies that they may be robust to noise.

In the second experiment, we consider the $500 \times 500 \times 500$ sparse tensor, which is created by 3 cores whose non-zero entries are drawn from a standard normal distribution. Specifically, the 3 cores of size $10 \times 500 \times 10$ are generated by using the Matlab function `sptenrand([10,500,10], density)`, where $0 \leq \text{density} \leq 1$ denotes the sparsity. So the core tensor contains approximately $\text{density} \times 50000$ normal distributed non-zero entries. In the specific experiments, we set $\text{density} = 0.05$.

Figure 5 shows the numerical results of this experiment. The conclusions from these results are similar to the ones from the first experiment. That is, TR-KSRFT-ALS, TR-TS-ALS and TR-ALS-Sampled have similar performance in relative errors and running time for the data with noise, and if the data has no noise, our methods are a little worse in running time compared with TR-ALS-Sampled. For all the cases, the above three methods always perform much better than TR-ALS in running time. Furthermore, our methods also show superior robustness.

In the third experiment, we use the $500 \times 500 \times 500$ tensor generated by a method similar to the one in Appendix C.1 of [19] which is used to create data for testing the algorithms for
Figure 5. Embedding sizes v.s. relative errors and running time (seconds) of the second synthetic experiment with true and target ranks $R_{true} = R = 10$ and different noises.

CP decomposition. Specifically, we first create 3 cores of size $10 \times 500 \times 10$ with entries drawn independently from a standard normal distribution and then set the first three columns of the mode-2 unfolding matrix of each core to be zero. Finally, a few nonzero elements are added to those zero columns. Hence, there are two user-specified parameters in this kind of data: spread and magnitude. **spread** means how many nonzero elements are added to each column. For example, $spread = 15$ means that the first 15 rows in the first column, the second 15 rows in the second column, and the third 15 rows in the third column are nonzero. **magnitude** means the size of the added nonzero elements. In the specific experiments, we set $spread = 15$, $magnitude = I/4 - 10$, and do a change on the third core further. That is, we set all the rows except the first 15 ones of the mode-2 unfolding matrix to be zero.

The numerical results for this data are shown in Figure 6. They are a little different from the ones from the previous two experiments. Our algorithms perform better, especially for the case with noise, than TR-ALS-Sampled in terms of the relative errors under almost the same running time. Moreover, to achieve the similar errors as TR-ALS, the embedding sizes of our algorithms are much smaller than that of TR-ALS-sampled. We plot the leverage scores of the classical mode-2 unfolding matrices of the original and transformed TR-cores in Figure 7, which shows that the original cores indeed have some special fibers. This is consistent with the analysis in Remark 4.1.

The fourth experiment is mainly used to test Algorithm 4.2 (TR-KSRFT-ALS-Premix). The complex tensor of size $500 \times 500 \times 500$ is generated in the same way as the first experiment except that the cores are complex. Figure 8 shows the relative errors and running time for this data with different noises, which is in good agreement with the results shown in the first experiment, and the difference between the running time for data without noise is also not
remarkable. So, we can conclude that Algorithm 4.2 can perform well and is stable under noise for complex data.

6.2. Real data. We consider five real data tensors whose brief information is listed in Table 3 including specific links provided in footnotes.

| Dataset        | Size               | Type          |
|----------------|--------------------|---------------|
| Indian Pines   | $145 \times 145 \times 220$ | Hyperspectral |
| SalinasA.      | $83 \times 86 \times 224$  | Hyperspectral |
| C1-vertebrae   | $512 \times 512 \times 47$  | CT Images     |
| Uber.Hour      | $183 \times 1140 \times 1717$ | Sparse        |
| Uber.Date      | $24 \times 1140 \times 1717$  | Sparse        |

The first two datasets are composed of 3-order tensors containing hyperspectral images, where the first two orders are the image height and width, and the third one is the number of spectral bands. The third dataset consists of $512 \times 512 \times 47$ tensors where the $512 \times 512$ frontal slice of each tensor is a 2-dimensional slice of the C1-vertebrae. These images were

4http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral\_Remote\_Sensing\_Scenes
5https://isbweb.org/data/vsj/
6http://frostt.io/
obtained from the Laboratory of Human Anatomy and Embryology, University of Brussels (ULB), Belgium. The last two datasets are extracted from the Uber tensor which is a 4-order tensor of size $183 \times 24 \times 1140 \times 1717$ with 3,309,490 nonzeros (0.038%). The entry $(i, j, k, l)$ denotes the number of pickups on date $i$, hour $j$, at latitude $k$ and longitude $l$, in New York city during the period April–August 2014. Due to the limitation of experimental equipment, we divide it into two parts with respect to date and hour respectively, i.e., the Uber.Date and Uber.Hour. The former contains the number of pickups on hour $j$, at latitude $k$ and longitude $l$ for a day, i.e., the entry is $(j, k, l)$, and the latter contains the number of pickups on date $i$, at latitude $k$ and longitude $l$ for an hour, i.e., the entry is $(i, k, l)$.

Considering that the real data always has irregular dimensions, we will not use the same experimental methods as in the second stage in synthetic experiments. Here, we mainly compare the running time and the embedding size $m$ for the same number of iterations and almost the same relative errors. Specifically, we first find the maximum iterations according to the method in the first stage of synthetic experiments with $M = 100$ and $\varepsilon = 1 \times 10^{-3}$. Then, for TR-ALS, we can get the running time and relative error; for other algorithms, we try the embedding size $m$ incrementally using the fixed maximum of iterations until the error is close to the 1.1 times of the TR-ALS error. The size $m$ is started at $J_{\text{init}} = 2R^2$ and incremented by $J_{\text{inc}} = 100$, and hence $J_{\text{fin}} = J_{\text{init}} + J_{\text{inc}} \times \text{num}$, where the target rank $R$ is a priori estimate and is different for different data. Thus, if we record the number of increases for the embedding size, i.e., $\text{num}$, we can report the final embedding size $m$. Meanwhile, we can record the running time and the final error for each algorithm. The specific numerical results are presented in Table 4.

Compared with the results of part of the experiments on synthetic data, the performance of our methods on these real data tensors is a little better. Except for TR-TS-ALS on the
Figure 8. Embedding sizes v.s. relative errors and running time (seconds) of the fourth synthetic experiment with true and target ranks $R_{true} = R = 10$ and different noises.

7. Concluding remarks. In this paper, we propose two randomized algorithms, i.e., TR-KSRFT-ALS and TR-TS-ALS, for TR decomposition using KSRFT and TensorSketch. They have pretty decent performance in terms of accuracy and computing time compared with the state-of-the-art method, i.e., TR-ALS-Sampled. To achieve these, two new tensor products are defined and their properties are investigated. These new products and properties are of interest in their own right. For example, they can be used to devise the hybrid algorithms for TR decomposition based on stochastic gradient descent and randomized preconditioning. We will present this work in a subsequent paper. Another future work is to try to reduce the computational cost of the mixing tensor since it usually dominates the total cost of the algorithm.

REFERENCES

[1] I. Affleck, T. Kennedy, E. H. Lieb, and H. Tasaki, Valence bond ground states in isotropic quantum antiferromagnets, Comm. Math. Phys., 115 (1988), pp. 477–528, https://doi.org/10.1007/BF01218021.

[2] S. Ahmadi-Asl, A. Cichocki, A. H. Phan, M. G. Asante-Mensah, M. M. Ghazani, T. Tanaka, and I. V. Oseledets, Randomized algorithms for fast computation of low rank tensor ring model, Mach. Learn.: Sci. Technol., 2 (2020), p. 011001, https://doi.org/10.1088/2632-2153/abad87.
Table 4
Decomposition results for real datasets with different target ranks.

| Method      | TR-ALS | TR-ALS-Sampled | TR-KSRFT-ALS | TR-TS-ALS |
|-------------|--------|----------------|--------------|-----------|
| Indian Pines (R = 20) | Error 0.0263 | 0.0289 | 0.0289 | 0.0289 |
|             | Time 32.9536 | 13.7424 | 12.3571 | 12.0229 |
|             | num - | 120 | 53 | 73 |
|             | Embedding size (m) - | 12800 | 6100 | 8100 |
| SalinasA. (R = 15) | Error 0.0066 | 0.0069 | 0.0073 | 0.0073 |
|             | Time 4.0225 | 2.4166 | 1.8510 | 2.2868 |
|             | num - | 54 | 23 | 30 |
|             | Embedding size (m) - | 5850 | 2750 | 3450 |
| C1-vertebrae (R = 25) | Error 0.0804 | 0.0882 | 0.0883 | 0.0883 |
|             | Time 409.7951 | 128.3391 | 101.7646 | 156.5089 |
|             | num - | 228 | 88 | 217 |
|             | Embedding size (m) - | 24050 | 10050 | 22950 |
| Uber.Hour (R = 15) | Error 0.7530 | 0.8246 | 0.8272 | 0.8274 |
|             | Time 869.1631 | 64.7240 | 39.0307 | 45.3829 |
|             | num - | 230 | 40 | 47 |
|             | Embedding size (m) - | 23450 | 4450 | 5150 |
| Uber.Date (R = 18) | Error 0.3864 | 0.4226 | 0.4246 | 0.4239 |
|             | Time 1452.1900 | 159.1936 | 51.3584 | 113.8542 |
|             | num - | 320 | 46 | 147 |
|             | Embedding size (m) - | 32648 | 5248 | 15348 |

[3] B. W. Bader, T. G. Kolda, et al., Tensor toolbox for matlab, 2021, https://www.tensortoolbox.org (accessed 2021/04/05). Version 3.2.1.
[4] C. Battaglino, G. Ballard, and T. G. Kolda, A practical randomized CP tensor decomposition, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 876–901, https://doi.org/10.1137/17M1112303.
[5] M. Che and Y. Wei, Randomized algorithms for the approximations of Tucker and the tensor train decompositions, Adv. Comput. Math., 45 (2019), pp. 395–428, https://doi.org/10.1007/s10444-018-9622-8.
[6] M. Che, Y. Wei, and H. Yan, The computation of low multilinear rank approximations of tensors via power scheme and random projection, SIAM J. Matrix Anal. Appl., 41 (2020), pp. 605–636, https://doi.org/10.1137/19M1237016.
[7] M. Che, Y. Wei, and H. Yan, An efficient randomized algorithm for computing the approximate Tucker decomposition, J. Sci. Comput., 88 (2021), pp. 1–29, https://doi.org/10.1007/s10915-021-01545-5.
[8] A. Cichocki, N. Lee, I. V. Oseledets, A. H. Phan, Q. Zhao, and D. P. Mandic, Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions, Found. Trends Mach. Learn., 9 (2016), pp. 249–429, https://doi.org/10.1561/2200000059.
[9] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. PHAN, Ten-
Tensor decompositions for signal processing applications: From two-way to multiway component analysis, IEEE Signal Process. Mag., 32 (2015), pp. 145–163, https://doi.org/10.1109/MSP.2013.2297439.

[10] K. L. Clarkson and D. P. Woodruff, Low-rank approximation and regression in input sparsity time, J. ACM, 63 (2017), pp. 54:1–54:45, https://doi.org/10.1145/3019134.

[11] H. Diao, Z. Song, W. Sun, and D. P. Woodruff, Sketching for Kronecker product regression and p-splines, in International Conference on Artificial Intelligence and Statistics, vol. 84, Playa Blanca, Lanzarote, Canary Islands, 2018, PMLR, pp. 1299–1308.

[12] S. V. Dolgov and D. V. Savostyanov, Alternating minimal energy methods for linear systems in higher dimensions, SIAM J. Sci. Comput., 36 (2014), pp. A2248 –A2271, https://doi.org/10.1137/140953289.

[13] P. Drineas and M. W. Mahoney, Lectures on randomized numerical linear algebra, in The Mathematics of Data, M. W. Mahoney, J. C. Duchi, and A. C. Gilbert, eds., vol. 25 of IAS/Park City Mathematics Series, AMS/IAS/SIAM, 2018, pp. 1–48, https://doi.org/10.1090/p cms/025/00829.

[14] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, Faster least squares approximation, Numer. Math., 117 (2011), pp. 219–249, https://doi.org/10.1007/s00211-010-0331-6.

[15] T. Gong, Y. Dong, H. Chen, B. Dong, and C. Li, Markov subsampling based huber criterion, 2021, https://arxiv.org/abs/2112.06134.

[16] T. Gong, Q. Xi, and C. Xu, Robust gradient-based markov subsampling, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34(04), New York Hilton Midtown, USA, 2020, AAAI Press, pp. 4004–4011.

[17] R. Jin, T. G. Kolda, and R. Ward, Faster Johnson-Lindenstrauss transforms via Kronecker products, Inf. Inference, 10 (2021), pp. 1533–1562, https://doi.org/10.1093/imaiai/iaaa028.

[18] T. G. Kolda and B. W. Bader, Tensor decompositions and applications, SIAM Rev., 51 (2009), pp. 455–500, https://doi.org/10.1137/0707111X.

[19] B. W. Larsen and T. G. Kolda, Practical leverage-based sampling for low-rank tensor decomposition, 2020, https://arxiv.org/abs/2006.16438.

[20] L. Ma and E. Solomonik, Fast and accurate randomized algorithms for low-rank tensor decompositions, in Advances in Neural Information Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds., vol. 34, Curran Associates, Inc., 2021, pp. 24299–24312.

[21] O. A. Malik, More efficient sampling for tensor decomposition, 2021, https://arxiv.org/abs/2110.07631.

[22] O. A. Malik and S. Becker, Low-rank Tucker decomposition of large tensors using TensorSketch, in Advances in Neural Information Processing Systems, vol. 31, Montréal, Canada, 2018, Curran Associates, Inc., pp. 10117–10127.

[23] O. A. Malik and S. Becker, Fast randomized matrix and tensor interpolative decomposition using CountSketch, Adv. Comput. Math., 46 (2020), https://doi.org/10.1007/s10444-020-09816-9.

[24] O. A. Malik and S. Becker, Guarantees for the Kronecker fast Johnson-Lindenstrauss transform using a coherence and sampling argument, Linear Algebra Appl., 602 (2020), pp. 120–137, https://doi.org/10.1016/j.laa.2020.05.004.

[25] O. A. Malik and S. Becker, A sampling-based method for tensor ring decomposition, in Proceedings of the 38th International Conference on Machine Learning, vol. 139, Virtual Event, 2021, PMLR, pp. 7400–7411.

[26] P.-G. Martinsson and J. A. Tropp, Randomized numerical linear algebra: Foundations and algorithms, Acta Numer., 29 (2020), pp. 403–572, https://doi.org/10.1017/S0962492920000021.

[27] R. Minster, A. K. Saibaba, and M. E. Kilmer, Randomized algorithms for low-rank tensor decompositions in the Tucker format, SIAM J. Math. Data Sci., 2 (2020), pp. 189–215, https://doi.org/10.1137/19M1261043.

[28] I. V. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317, https://doi.org/10.1137/1007952286.

[29] R. Pagh, Compressed matrix multiplication, ACM Trans. Comput. Theory, 5 (2013), pp. 1–17, https://doi.org/10.1145/2493252.2493254.

[30] D. Perez-Garcia, F. Verstraete, M. M. Wolf, and J. I. Cirac, Matrix product state representations, Quantum Inform. Comput., 7 (2007), pp. 401–430.

[31] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, Tensor decomposition for signal processing and machine learning, IEEE Trans. Signal Process., 65
(2017), pp. 3551–3582, https://doi.org/10.1109/TSP.2017.2690524.

[32] M. Vandecappelle, Numerical Algorithms for Tensor Decompositions, PhD thesis, Arenberg Doctoral School, Faculty of Engineering Science, 2021.

[33] D. P. Woodruff, Sketching as a tool for numerical linear algebra, Found. Trends Theor. Comput. Sci., 10 (2014), pp. 1–157, https://doi.org/10.1561/0400000060.

[34] L. Yuan, C. Li, J. Cao, and Q. Zhao, Randomized tensor ring decomposition and its application to large-scale data reconstruction, in ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton Conference Centre Brighton, U.K., 2019, IEEE, pp. 2127–2131.

[35] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, Tensor ring decomposition, 2016, https://arxiv.org/abs/1606.05535.

[36] G. Zhou, A. Cichocki, and S. Xie, Decomposition of big tensors with low multilinear rank, 2014, https://arxiv.org/abs/1412.1885.