# Supplementary Information

# 1 Supplementary Methods

## 1.1 Tensor Preliminaries

In this section, we describe preliminaries of a tensor and its factorization methods. Table 1 summarizes symbols used in this paper.

Table 1. Table of symbols.

| Symbol | Definition |
|---|---|
| $\mathcal{X}$ | tensor (Euler script, bold letter) |
| $\mathbf{X}$ | matrix (uppercase, bold letter) |
| $x$ | scalar (lower case, italic letter) |
| $N$ | order (number of modes) of a tensor |
| $I_n, J_n$ | dimensionality of the $n$th mode of input and core tensor |
| $\mathbf{A}^{(n)}$ | $n$th factor matrix ($\in \mathbb{R}^{I_n \times J_n}$) |
| $a^{(n)}_{i_n j_n}$ | $(i_n, j_n)$th entry of $\mathbf{A}^{(n)}$ |
| $\Omega$ | set of observable entries of $\mathcal{X}$ |
| $\lvert\Omega\rvert, \lvert\mathcal{G}\rvert$ | number of observable entries of input and core tensor |
| $\lambda$ | regularization parameter for factor matrices |
| $\lvert\lvert \bullet \rvert\rvert$ | Frobenius norm |
| $*$ | element-wise multiplication |

### 1.1.1 Tensor

A tensor is a multi-dimensional array which is a generalization of a matrix and a vector. A mode or a way indicates each axis of a tensor, and an order is the number of modes or ways. We denote a tensor using boldface Euler script letters (e.g., $\mathcal{X}$). A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is an $N$-order tensor which has $N$ modes whose lengths are from $I_1$ to $I_N$. A vector and a matrix are regarded as a 1- and a 2-order tensor, respectively. We denote a matrix and a vector using boldface uppercase (e.g., $\mathbf{X}$) and lowercase letters (e.g., $\mathbf{x}$), respectively. The $i_1$th row of $\mathbf{A}$ is denoted by $\mathbf{a}_{i_1:}$, and the $i_2$th column of $\mathbf{A}$ is denoted by $\mathbf{a}_{:i_2}$.

### 1.1.2 Tucker Decomposition for Partially Observable Tensors

Among many tensor decomposition methods, we use Tucker factorization (Tucker, 1966; De Lathauwer *et al.*, 2000), which allows us to discover not only latent concepts but also relations between the concepts hidden in tensors. Tucker factorization decomposes a given tensor $\mathcal{X}$ into a core tensor $\mathcal{G}$ and factor matrices $\mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}$, as defined in Definition 1.

Definition 1. ***(Tucker factorization)*** *Given a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with observable entries $\Omega$, Tucker factorization of rank $(J_1, \cdots, J_N)$ finds a core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ and factor matrices $\mathbf{A}^{(1)} \in \mathbb{R}^{I_1 \times J_1}, \cdots, \mathbf{A}^{(N)} \in \mathbb{R}^{I_N \times J_N}$ which minimize the following objective function.*

$$\mathcal{L}(\mathcal{G}, \mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}) = \sum_{\forall (i_1,\ldots,i_N) \in \Omega} \left( \mathcal{X}_{(i_1,\ldots,i_N)} - \sum_{\forall (j_1,\ldots,j_N) \in \mathcal{G}} \mathcal{G}_{(j_1,\ldots,j_N)} \prod_{n=1}^{N} a^{(n)}_{i_n j_n} \right)^2 + \lambda \left( \sum_{n=1}^{N} ||\mathbf{A}^{(n)}||_F^2 \right) \quad (1)$$

Note that $\lambda$ denotes a regularization parameter for factor matrices, and we used $L_2$-regularization to prevent overfitting, which has been widely used in machine learning (Koren *et al.*, 2009; Shin *et al.*, 2017*a*).

Equation (1) only utilizes observed entries of a tensor during factorizations as missing entries of $\mathcal{X}$ have unknown values. In addition, there are no constraints (e.g., non-negativity or orthogonality) on factor matrices in Equation (1). Each column vector of a factor matrix generally represents a concept or a pattern. A higher value in a vector indicates that the corresponding element is highly related to the concept. The core tensor encodes how these concepts are related to each other. Assuming a given tensor is movie rating data with (movie - user - time) triples, then a column vector in a movie-factor matrix can have a concept such as a horror or comic genre.

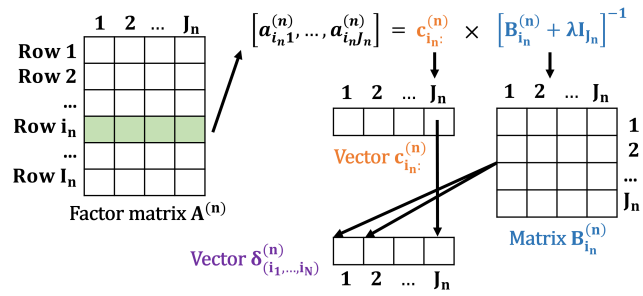


**Fig. 1.** An illustration of an update rule for a row of a factor matrix. P-Tucker requires three intermediate data $\mathbf{B}^{(n)}_{i_n}$, $\mathbf{c}^{(n)}_{i_n:}$, and $\delta^{(n)}_{(i_1,\ldots,i_N)}$ for updating the $i_n$th row of $\mathbf{A}^{(n)}$. Note that $\lambda$ is a regularization parameter, and $\mathbf{I}_{J_n}$ is a $J_n \times J_n$ identity matrix.

## 1.2 Baseline Approach: P-Tucker

P-Tucker is a scalable and accurate Tucker factorization method which updates factor matrices in a row-wise manner based on ALS. Algorithm 2 and Figures 1 and 2 illustrate how P-Tucker updates factor matrices. In Algorithm 2, P-Tucker first initializes all $\mathbf{A}^{(n)}$ and $\mathcal{G}$ with random real values between 0 and 1 (line 1). After which P-Tucker updates a single factor matrix in a row-wise manner and iterates it for all factor matrices (lines 3-8). When all factor matrices are updated, P-Tucker measures reconstruction error using (5) (line 9). P-Tucker stops iterations if the error converges or the maximum iteration is reached (line 10). Finally, P-Tucker performs a **optional** QR decomposition on all $\mathbf{A}^{(n)}$ to make them orthogonal and updates $\mathcal{G}$ (lines 12-14). Specifically, QR decomposition on each $\mathbf{A}^{(n)}$ is defined as follows:

$$\mathbf{A}^{(n)} = \mathbf{Q}^{(n)} \mathbf{R}^{(n)}, \; n = 1 \ldots N \quad (2)$$

where $\mathbf{Q}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ is *column-wise orthonormal* and $\mathbf{R}^{(n)} \in \mathbb{R}^{J_n \times J_n}$ is *upper-triangular*. Therefore, by substituting $\mathbf{Q}^{(n)}$ for $\mathbf{A}^{(n)}$, P-Tucker succeeds in making factor matrices orthogonal. When QR decomposition is performed, core tensor $\mathcal{G}$ must be updated accordingly in order to maintain the same reconstruction error. The update rule of core tensor $\mathcal{G}$ is given as follows:

$$\mathcal{G} \leftarrow \mathcal{G} \times_1 \mathbf{R}^{(1)} \cdots \times_N \mathbf{R}^{(N)}. \quad (3)$$

Note that performing QR decomposition is optional, and GIFT skips this process as QR decomposition may sweep out latent patterns in factor matrices.

Given a row of a factor matrix, an update rule is derived by computing a gradient with respect to the given row and setting it as zero, which minimizes the loss function (1). The update rule for the $i_n$th row of the $n$th factor matrix $\mathbf{A}^{(n)}$ (see Figure 1) is given as follows.

Theorem 1 (Row-wise update rule of factor matrices with no masking constraints). *The proposed row-wise update rule* (4) *minimizes the loss function* (1) *regarding the updated parameters.*

$$\underset{[a^{(n)}_{i_n 1}, \ldots, a^{(n)}_{i_n J_n}]}{\arg\min} \; L(\mathcal{G}, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) = \mathbf{c}^{(n)}_{i_n:} \times [\mathbf{B}^{(n)}_{i_n} + \lambda \mathbf{I}_{J_n}]^{-1} \quad (4)$$

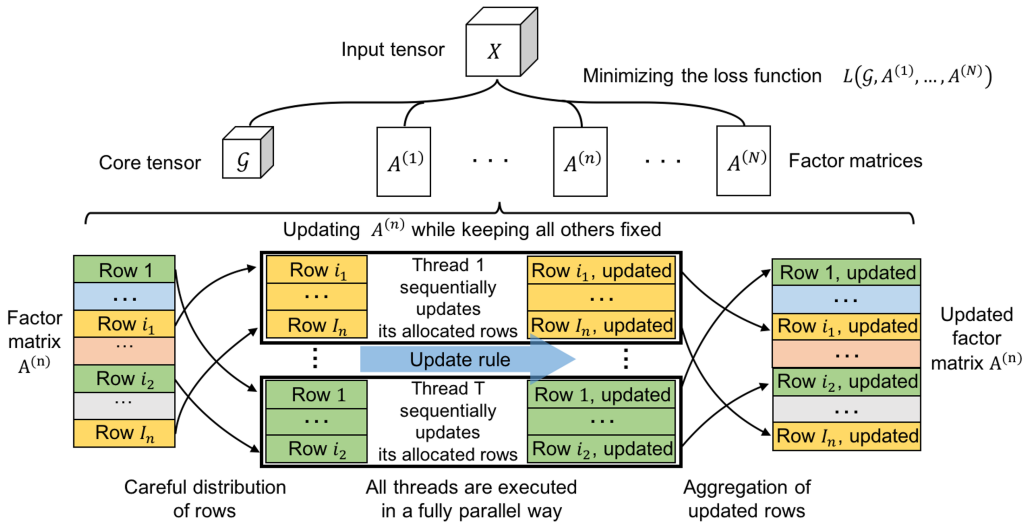


**Fig. 2.** An overview of P-Tucker. P-Tucker performs a row-wise ALS which updates each row of a factor matrix $\mathbf{A}^{(n)}$ while keeping all the others fixed. Since all rows of a factor matrix are independent of each other in terms of minimizing the loss function (1), P-Tucker fully exploits multi-core parallelism to update all rows of $\mathbf{A}^{(n)}$. First, all rows are carefully distributed to all threads to achieve a uniform workload among them. After that, all threads update their allocated rows in a fully parallel way. In a single thread, the allocated rows are updated in a sequential way. Finally, P-Tucker aggregates all updated rows from all threads to update $\mathbf{A}^{(n)}$. P-Tucker iterates this update procedure for all factor matrices one by one.

*where* $\mathbf{B}^{(n)}_{i_n}$ *is a* $J_n \times J_n$ *matrix whose* $(j_1, j_2)$*th entry is*

$$\sum_{\forall (i_1,\ldots,i_N) \in \Omega^{(n)}_{i_n}} \delta^{(n)}_{(i_1,\ldots,i_N)}(j_1)\delta^{(n)}_{(i_1,\ldots,i_N)}(j_2), \tag{5}$$

$\mathbf{c}^{(n)}_{i_n:}$ *is a length* $J_n$ *vector whose* $j$*th entry is*

$$\sum_{\forall (i_1,\ldots,i_N) \in \Omega^{(n)}_{i_n}} \mathcal{X}_{(i_1,\ldots,i_N)}\delta^{(n)}_{(i_1,\ldots,i_N)}(j), \tag{6}$$

$\delta^{(n)}_{(i_1,\ldots,i_N)}$ *is a length* $J_n$ *vector whose* $j$*th entry is*

$$\sum_{\forall (j_1\ldots j_n = j\ldots j_N) \in \mathcal{G}} \mathcal{G}_{(j_1\ldots j_n = j\ldots j_N)} \prod_{k \neq n} a^{(k)}_{i_k j_k}, \tag{7}$$

Proof.

$$\frac{\partial L}{\partial a^{(n)}_{i_n j_n}} = 0, \forall j_n, 1 \leq j_n \leq J_n$$

$$\Leftrightarrow \textstyle\sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \left( \left( \mathcal{X}_\alpha - \sum_{\forall \beta \in \mathcal{G}} \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n} \right) \times \left( -\delta^{(n)}_\alpha(j_n) \right) \right) + \lambda a^{(n)}_{i_n j_n} = 0$$

$$\Leftrightarrow [a^{(n)}_{i_n 1}, \ldots, a^{(n)}_{i_n J_n}] \left( \textstyle\sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \left( \delta^{(n)\mathbf{T}}_\alpha \delta^{(n)}_\alpha \right) + \lambda \mathbf{I}_{J_n} \right) = \sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \left( \mathcal{X}_\alpha \delta^{(n)}_\alpha \right)$$

$$\Leftrightarrow [a^{(n)}_{i_n 1}, \ldots, a^{(n)}_{i_n J_n}] = \mathbf{c}^{(n)}_{i_n:} \times [\mathbf{B}^{(n)}_{i_n} + \lambda \mathbf{I}_{J_n}]^{-1}$$

$\Omega^{(n)}_{i_n}$ indicates the subset of $\Omega$ whose $n$th mode's index is $i_n$, $\lambda$ is a regularization parameter, and $\mathbf{I}_{J_n}$ is a $J_n \times J_n$ identity matrix. As shown in Figure 1, the update rule for the $i_n$th row of $\mathbf{A}^{(n)}$ requires three intermediate data $\mathbf{B}^{(n)}_{i_n}$, $\mathbf{c}^{(n)}_{i_n:}$, and $\delta^{(n)}_{(i_1,\ldots,i_N)}$. Those data are computed by the subset of observable entries $\Omega^{(n)}_{i_n}$. Thus, computational costs of updating factor matrices are proportional to the number of observable entries, which lets P-Tucker fully exploit the sparsity of given tensors. Note that a matrix $[\mathbf{B}^{(n)}_{i_n} + \lambda \mathbf{I}_{J_n}]$ is positive-definite and invertible, and the full version of Proof 1.3 is available at https://datalab.snu.ac.kr/ptucker/supple.pdf.

**Algorithm 2** P-Tucker

**Input:** A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with observable entries $\Omega$, mask matrices $\mathbf{M}^{(1)}, \cdots, \mathbf{M}^{(N)}$, rank $(J_1, \cdots, J_N)$, and a regularization parameter $\lambda$.
**Output:** A core tensor $\mathcal{G}$ and factor matrices $\mathbf{A}^{(1)}, \cdots \mathbf{A}^{(N)}$.

```
initialize G and A^(1), ..., A^(N) randomly
repeat
    for n = 1, ..., N do
        for i_n = 1, ..., I_n do
            calculate intermediate data δ, B^(n)_{i_n}, and c^(n)_{i_n:} by Eq. (2) – (4)
            update a row a^(n)_{i_n:} by c^(n)_{i_n:} × [B^(n)_{i_n} + λI_{J_n}]^{-1}
        end for
    end for
    compute reconstruction error by Eq. (5)
until a convergence criterion is met
for n = 1...N do
    A^(n) → Q^(n) R^(n)
    A^(n) ← Q^(n)
    G ← G ×_n R^(n)
end for
```

## 1.3 GIFT Algorithm for General N-order Tensors

GIFT adapts the row-wise update rule proposed for P-Tucker with modified loss function. The derivation of the update rules are similar to that of P-Tucker as provided in the following Theorem 2 and proof.

Theorem 2 (Row-wise update rule of factor matrices with masking constraints). *The proposed row-wise update rule* (8) *minimizes the loss function* (6) *regarding the updated parameters.*

$$\underset{[a^{(n)}_{i_n:}]}{\arg\min}\, L(\mathcal{G}, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}, \mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(N)}) = \mathbf{c}^{(n)}_{i_n:} \times [\mathbf{B}^{(n)}_{i_n} + \lambda \mathbf{D}^{(n)}_{i_n}]^{-1} \tag{8}$$

*where* $\mathbf{D}^{(n)}_{i_n}$ *is a* $J_n \times J_n$ *diagonal matrix whose* $(j_n, j_n)$*th entry is* $m^{(n)}_{i_n, j_n}$, $\mathbf{B}^{(n)}_{i_n}$ *is a* $J_n \times J_n$ *matrix whose* $(j_1, j_2)$*th entry is Equation* (5), $\mathbf{c}^{(n)}_{i_n:}$ *is a length* $J_n$ *vector whose* $j$*th entry is Equation* (6), *and* $\delta^{(n)}_\alpha$ *is a length* $J_n$ *vector whose* $j$*th entry is Equation* (7).

Proof.

$$\frac{\partial L}{\partial a^{(n)}_{i_n j_n}} = 0, \forall j_n, 1 \leq j_n \leq J_n$$

$$\Leftrightarrow \textstyle\sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \left( \left( \mathcal{X}_\alpha - \sum_{\forall \beta \in \mathcal{G}} \mathcal{G}_\beta \prod_{n=1}^{N} a^{(n)}_{i_n j_n} \right) \times \left( -\delta^{(n)}_\alpha(j_n) \right) \right) + \lambda m^{(n)}_{i_n j_n} a^{(n)}_{i_n j_n} = 0$$

$$\Leftrightarrow a^{(n)}_{i_n:}\left(\sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}\left(\delta^{(n)\mathbf{T}}_{\alpha}\delta^{(n)}_{\alpha}\right)+\lambda\mathbf{D}^{(n)}_{i_n}\right)=\sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}\left(\mathcal{X}_{\alpha}\delta^{(n)}_{\alpha}\right)$$

$$\Leftrightarrow [a^{(n)}_{i_n 1},...,a^{(n)}_{i_n J_n}]=\mathbf{c}^{(n)}_{i_n:}\times[\mathbf{B}^{(n)}_{i_n}+\lambda\mathbf{D}^{(n)}_{i_n}]^{-1}$$

With the derived row-wise update rules, the GIFT algorithm for $N$-order tensor is provided in the Algorithm 3.

**Algorithm 3** N-order GIFT

**Input:** A tensor $\mathcal{X}\in\mathbb{R}^{I_1\times\cdots\times I_N}$ with observable entries $\Omega$, mask matrices $\mathbf{M}^{(1)},\cdots,\mathbf{M}^{(N)}$, rank $(J_1,\cdots,J_N)$, and a regularization parameter $\lambda$.
**Output:** A core tensor $\mathcal{G}$ and factor matrices $\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}$.
1: initialize $\mathcal{G}$ and $\mathbf{A}^{(1)},\cdots,\mathbf{A}^{(N)}$ randomly
2: **repeat**
3: **for** $n=1,\cdots,N$ **do**
4: **for** $i_n=1,\cdots,I_n$ **do**
5: calculate intermediate data $\delta$, $\mathbf{B}^{(n)}_{i_n}$, and $\mathbf{c}^{(n)}_{i_n:}$ by Eq. (2) – (4)
6: calculate diagonal matrix $\mathbf{D}^{(n)}_{i_n}$, where its $(j_n,j_n)$th entry is $\mathbf{M}^{(n)}_{i_n j_n}$
7: update a row $a^{(n)}_{i_n:}$ by $\mathbf{c}^{(n)}_{i_n:}\times[\mathbf{B}^{(n)}_{i_n}+\lambda\mathbf{D}^{(n)}_{i_n}]^{-1}$
8: **end for**
9: **end for**
10: compute reconstruction error by Eq. (5)
11: **until** error converges or exceeds maximum iteration

### 1.4 Theoretical analyses of GIFT

In this section, we offer theoretical analyses of GIFT in terms of time, memory, and convergence. Specifically, we analyze time and memory complexity of GIFT (P-Tucker and Silenced-TF have the same complexities to GIFT). For simplicity, we assume $I_1=...=I_N=I$ and $J_1=...=J_N=J$. Table 2 summarizes the time and memory complexities of GIFT. Note that we calculate time complexity per iteration, and we focus on memory complexity of intermediate data, not of all variables.

Table 2. Complexity analysis of GIFT with respect to time and memory. Note that memory complexity indicates the space requirement for intermediate data. $|\Omega|$ is the number of observable entries in $\mathcal{X}$ and $T$ is the number of threads.

| Algorithm | Time Complexity (per iteration) | Memory Complexity |
|---|---|---|
| GIFT | $O(NIJ^3+N^2\|\Omega\|J^N)$ | $O(TJ^2)$ |

Theorem 3 (Time complexity of GIFT). *The time complexity of* GIFT *is* $O(NIJ^3+N^2|\Omega|J^N)$.

Proof. Given the $i_n$th row of $\mathbf{A}^{(n)}$ (lines 3-4) in Algorithm 1 , computing $\delta$ (line 5) takes $O(N|\Omega^{(n)}_{i_n}|J^N)$. Updating $\mathbf{B}^{(n)}_{i_n}$, $\mathbf{c}^{(n)}_{i_n:}$, and $\mathbf{D}^{(n)}_{i_n}$ (lines 5-6) takes $O(|\Omega^{(n)}_{i_n}|J^2)$ since $\delta$ is already calculated. Inverting $[\mathbf{B}^{(n)}_{i_n}+\lambda\mathbf{D}^{(n)}_{i_n}]$ (line 7) takes $O(J^3)$, and updating a row (line 7) takes $O(J^2)$. Thus, the time complexity of updating the $i_n$th row of $\mathbf{A}^{(n)}$ (lines 5-7) is $O(J^3+N|\Omega^{(n)}_{i_n}|J^N)$. Iterating it for all rows of $\mathbf{A}^{(n)}$ takes $O(IJ^3+N|\Omega|J^N)$. Finally, updating all $\mathbf{A}^{(n)}$ takes $O(NIJ^3+N^2|\Omega|J^N)$. According to (5), reconstruction (line 10) takes $O(N|\Omega|J^N)$. Thus, the time complexity of GIFT is $O(NIJ^3+N^2|\Omega|J^N)$.

Theorem 4 (Memory complexity of GIFT). *The memory complexity of* GIFT *is* $O(TJ^2)$.

Proof. The intermediate data of GIFT consist of two vectors $\delta$ and $\mathbf{c}^{(n)}_{i_n:}$ ($\in\mathbb{R}^J$) , and two matrices $\mathbf{B}^{(n)}_{i_n}$ and $[\mathbf{B}^{(n)}_{i_n}+\lambda\mathbf{D}^{(n)}_{i_n}]^{-1}$ ($\in\mathbb{R}^{J\times J}$). Memory spaces for those variables are released after updating the $i_n$th row of $\mathbf{A}^{(n)}$. Thus, they are not accumulated during the iterations. Since each thread has their own intermediate data, the total memory complexity of GIFT is $O(TJ^2)$.

Theorem 5 (Convergence of GIFT). GIFT *converges since* (1) *is bounded and decreases monotonically.*

Proof. According to Theorem 1, the loss function (1) never increases since every update in GIFT minimizes it, and (1) is bounded by 0. Thus, GIFT converges.

## 2 Supplementary Results

In this section, we offer additional experimental results of GIFT and other methods. In detail, we first introduce value distributions of masked and unmasked entries derived by P-Tucker and Silenced-TF. After that, we describe pattern stability of GIFT, justifications for a hyperparameter selection, and scalability of the algorithms with respect to the number of observed entries in a tensor.

### 2.1 Interpretability

As shown in Figure 3 middle, P-Tucker fails to make a distinction between masked and unmasked entries. The results are easily expected since P-Tucker does not differentiate the masked and unmasked ones when it updates factor matrices.

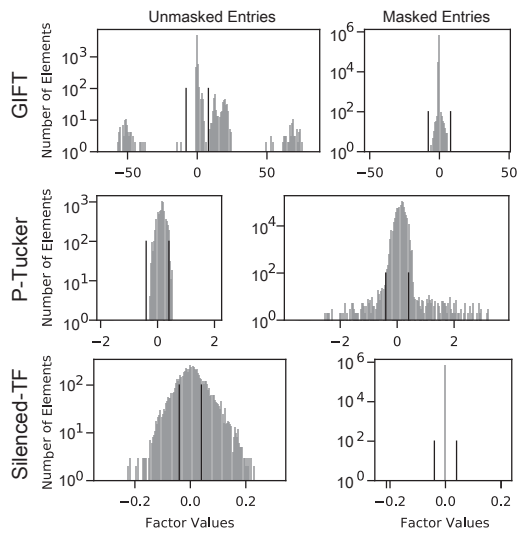


**Fig. 3.** Distributions of values in a gene factor matrix derived by GIFT, P-Tucker and Silenced-TF with $\lambda=10$.

On the other hand, Silenced-TF produces interpretable results, as presented in Figure 3 bottom. The values of masked entries are fixed to zeros, while the values of unmasked entries are varying from 0 to 0.3. Although Silenced-TF provides interpretable results, it cannot retrieve important masked entries as the values of them are set to zeros.

### 2.2 Pattern Stability

We repeat GIFT 10 times with random initializations in the default setting ($\lambda$ = 10 and $rank$ = 30 × 50 × 2). As shown in Tables 3 and 4, coefficient of variation (average / standard deviation) of two criteria are 0.02 and 0.061, respectively. The small values tell us that GIFT shows stable convergence regardless of random initialization. Note that 'Fit' is defined as $1-\frac{||\mathcal{X}-\mathcal{X}'||}{||\mathcal{X}||}$, where higher 'Fit' value indicates high accuracy.

Table 3. Stability experiments of GIFT. We repeat GIFT 10 times with random initializations. All parameters are set as default ($\lambda$=10 and rank=$30 \times 50 \times 2$).

| **Experiments** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Fit** | 0.85762 | 0.86785 | 0.82672 | 0.86191 | 0.82902 | 0.85963 | 0.82409 | 0.85601 | 0.82875 | 0.83383 |
| **Test RMSE** | 0.06842 | 0.06703 | 0.07939 | 0.0698 | 0.07531 | 0.07057 | 0.07419 | 0.0719 | 0.0805 | 0.07259 |

Table 4. Stability summary of GIFT. GIFT has small standard deviations compared to its averages, which implies that GIFT shows stable convergence with respect to reconstruction error and test RMSE.

| | **Fit** | **Test RMSE** |
|---|---|---|
| **Average** | 0.844543 | 0.07297 |
| **Standard Deviation** | 0.01648 | 0.004447 |

## 2.3 Hyperparameter Selection

One of the strengths of GIFT is that there are only few hyperparameters to be manually adjusted. The major parameter is a regularization coefficient $\lambda$, and we selected $\lambda = 10$ through various experiments, as presented in Table 5 (**bold** indicates the best one). Our criteria for choosing $\lambda$ include 1) high interpretability 2) low test RMSE. Regarding interpretability, $\lambda = 100$ is the best choice since it clearly distinguishes masked and unmasked entries (refer to Figure 4). Although it provides more interpretable results, it is hard to reveal important masked entries when $\lambda = 100$ as masked entries have too small values due to high penalties. Meanwhile, in the case of test RMSE, $\lambda = 1$ records the lowest value. Thus, we choose the middle of the parameters as $\lambda = 10$ since it has high interpretability and almost the same test RMSE to $\lambda = 1$. We note that it is not straightforward to make the optimization process automatic as interpretability is hard to be derived in a numerical format. In the case of rank, we choose the best one $30 \times 50 \times 2$ considering running time and accuracy.

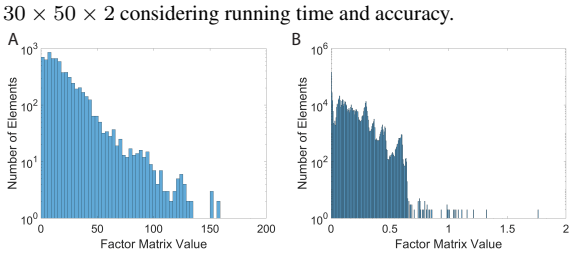


**Fig. 4.** Distributions of values in a gene factor matrix derived by GIFT ($\lambda$ = 100) for unmasked (A) and masked entries (B). The values of unmasked entries are much larger than that of masked ones.

## 2.4 Significant Factor Value

Significance of a gene factor value is chosen based on distribution of the factor values. Fig. 3 show gene factor value distribution of unmasked entries (gene set members) and masked entries solved via GIFT. Many unmasked entries had significant factor values ($\geq 8$ or $\leq -8$) and majority of the masked entries had insignificant factor values. This shows that GIFT has learned the latent relationships of cancer patients to gene sets and significant genes in the gene set by encoding prior knowledge during its training. P-Tucker, on the other hand, produces a gene factor matrix with value distribution that has small or no correlation to a gene set. (refer to Supplementary Figure 3 for gene factor distribution of Silenced-TF and P-Tucker).

Table 5. Hyperparameter experiment of GIFT. $\lambda = 10$ was selected as the best parameter with respect to interpretability and accuracy.

| $\lambda$ | 0.001 | 0.01 | 0.1 | 1 | **10** | 100 |
|---|---|---|---|---|---|---|
| Training RMSE | **0.0362** | 0.038425 | 0.038 | 0.0448 | 0.0731 | 0.08 |
| Test RMSE | 0.08659 | 0.097444 | 0.01 | **0.0747** | 0.075 | 0.082 |

## 2.5 Scalability

We vary the number of observable entries by randomly sampling 20%, 40%, 60%, 80%, and 100% from the PANCAN12 tensor. As shown in Figs 5, P-Tucker (**A**) and Silenced-TF (**B**) scale near linearly in terms of the number of observable entries.

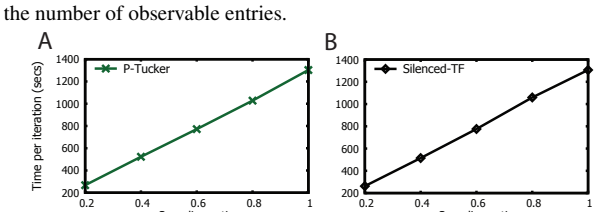


**Fig. 5.** Scalability of P-Tucker (A) and Silenced-TF (B) with respect to the number of observable entries in the tensor. As the number of observed entries increases, a running time of P-Tucker and Silenced-TF increases proportionally.

Figure 6 shows total running time of GIFT in terms of density of a tensor. GIFT shows linear scalability regarding the number of non-zeros.

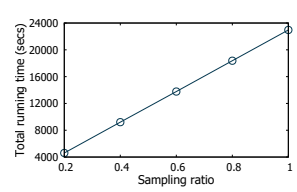


**Fig. 6.** Total running time of GIFT with respect to the number of non-zeros of a tensor. GIFT presents linear scalability similar to the case of per iteration time.

Moreover, we underline that GIFT is still scalable with the large-scale dataset since GIFT only stores non-zeros of a tensor and does not require huge intermediate data during its computation (refer to Table 6). The current size of the PanCan12 dataset is up to 2 GB and easily handled by modern hardware. However, provided that the dataset is scaled up enormously due to increased samples or platforms (e.g., $100,000 \times 14,000 \times 100$), the total size would be 2 TB (1,000 times larger), which cause out-of-memory error for standard Tucker methods.

Table 6. Scalability experiment of GIFT for a large-scale tensor. GIFT shows small time and memory overhead for the large-scale dataset (2 TB).

| **Scale** | **Time per iteration** | **Memory** |
|---|---|---|
| 4000 x 14000 x 5 (2 GB) | 1131.290453 | 5.61 GB |
| 100000 x 14000 x 100 (2 TB) | 1148.971755 | 5.66 GB |

## References

De Lathauwer,L., De Moor,B. and Vandewalle,J. (2000) A multilinear singular value decomposition. *SIAM J. Matrix Anal.& Appl,* **21** (4), 1253–1278.

Koren,Y., Bell,R. and Volinsky,C. (2009) Matrix factorization techniques for recommender systems. *Computer,* **42** (8), 30–37.

Tucker,L.R. (1966) Some mathematical notes on three-mode factor analysis. *Psychometrika,* **31** (3), 279–311.