Sampling

1 Introducing

Consider a N-order tensor, $\mathcal{X}$ with size $L_1 \times L_2 \times \ldots \times L_N$, the CP decomposition of this tensor is

$$\mathcal{X} = \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] = \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \cdots \circ \boldsymbol{a}_r^{(N)} \tag{1}$$

where

$$\begin{aligned}
\mathbf{A}^{(1)} &= \left[\boldsymbol{a}_1^{(1)}, \boldsymbol{a}_2^{(1)}, \cdots, \boldsymbol{a}_r^{(1)} \right] \in R^{L_1 \times R} \\
\mathbf{A}^{(2)} &= \left[\boldsymbol{a}_1^{(2)}, \boldsymbol{a}_2^{(2)}, \cdots, \boldsymbol{a}_r^{(2)} \right] \in R^{L_2 \times R} \\
&\vdots \\
\mathbf{A}^{(N)} &= \left[\boldsymbol{a}_1^{(N)}, \boldsymbol{a}_2^{(N)}, \cdots, \boldsymbol{a}_r^{(1)} \right] \in R^{L_N \times R}
\end{aligned}$$

In equation 1, $\mathbf{A}^{(n)}$ is the factor matrix, and the column size R represents the rank of this tenors, which means that $\mathcal{X}$ can be represented by a sum of these R rank one tensors. The vector $\boldsymbol{a}_r^{(n)}$ is the r-th column of matrix $\mathbf{A}^{(n)}$. In general, let $\boldsymbol{a}_{*r}^{(n)}$ be the k-th column of $\mathbf{A}^{(n)}$, $\boldsymbol{a}_{i_n *}^{(n)}$ be the i_n-th row vector of $\mathbf{A}^{(n)}$, and $a_{i_n k}^{(n)}$ be the element of $\mathbf{A}^{(n)}$.

The element in $\mathcal{X}$ satisfies the following equation:

$$x_{\boldsymbol{i}} = \sum_{r=1}^{R} a_{i_1 r}^{(1)} \cdot a_{i_2 r}^{(2)} \cdots a_{i_N r}^{(N)} \tag{2}$$

The indexes vector $\boldsymbol{i}$ is a shorthand for multi-index $(i_1, i_2, \ldots, i_N)$. We propose two methods, which are wedge sampling and diamond sampling, for estimating the maximum elements $x_{\boldsymbol{i}}$ in tensor when factor matrices $\mathbf{A}^{(n)}, n = 1, 2, \ldots, N$ are given. The wedge sampling is so much like the diamond sampling, and the diamond sampling ia an improvement of wedge sampling.

2 Diamond Sampling

The diamond sampling is a way to sample the multi-index $\boldsymbol{i} : (i_1, i_2, \ldots, {}_1 N)$ that is proportional to $x_{\boldsymbol{i}}^2$. We consider a matrix A with size $m \times n$ as a weighted bipartite graph G_A. Meanwhile, in the adjacent matrix of the graph we use 0(instead of ∞) to represent the two vertices are not adjacent. And the adjacent matrix of G_A is:

$$\begin{pmatrix} \mathbf{0}_{m \times m} & A \\ A^T & \mathbf{0}_{n \times n} \end{pmatrix}$$

Notation	Explanation
$\mathcal{A}$	tensor
$\mathbf{A}$	matrix
$\mathbf{A}^{(n)}$	n-th factor matrix of tensor
$\boldsymbol{a}_{*r}, \boldsymbol{a}_r$	k-th column of matrix
$\boldsymbol{a}_{i*}$	i-th row of matrix
$\boldsymbol{a}$	vector
a_{ir}	element of matrix

Table 1: Notation

2.1 Graph representation

In equation 1, we have N factor matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$. Those N factor matrices will be represented by a weighted $(N+1)$-partite graph G_T. And we call those $N+1$ partitions as partition $\overline{V}, V_1, V_2, \ldots, V_N$, in which partition V_n has L_n vertexes and partition $\overline{V}$ has R vertexes. We call $v_{i_n}^n$ the i_n-th vertex in partition V_n, and $\overline{v}_r$ the r-th vertex in partition $\overline{V}$. Every two vertexes from different partition classes $V_i, V_j, i, j \in 1, 2, \ldots, N$ are not adjacent. A vertex $\overline{v}_r$ in $\overline{V}$ is only adjacent to vertexes v_i^n in V_n when $a_{i_n r}^{(n)}$ is non-zero with the edge weight $a_{i_n r}^{(n)}$. And the adjacent matrix of G_T is :

$$\begin{pmatrix} \mathbf{0}_{R\times R} & \mathbf{A}^{(1)^T} & \ldots & \mathbf{A}^{(N)^T} \\ \mathbf{A}^{(1)} & \mathbf{0}_{L_1\times L_1} & \ldots & \mathbf{0}_{L_1\times L_N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{(N)} & \mathbf{0}_{L_N\times L_1} & \ldots & \mathbf{0}_{L_1\times L_N} \end{pmatrix}$$

Under the graph presentation, we define an event $\varepsilon_{\boldsymbol{i},r',r}$ consisting of three phases:

- phase 1. Pick an edge $e = (v_{i_1}^1, \overline{v}_r)$;
- phase 2. Walk $N-1$ times from the $\overline{v}_r$ to other partitions $V_2, \ldots, V_N$, arrive at $v_{i_2}^2, v_{i_3}^3, \ldots, v_{i_N}^N$;
- phase 3. Walk from $v_{i_1}^1$ to $\overline{V}$ and end in $\overline{v}_r'$.

When the event $\varepsilon_{\boldsymbol{i},r',r}$ happened, we give a score according to indexes $\boldsymbol{i} : (i_1, i_2, \ldots, i_N)$. We will assign each phase probabilities so that the final score of $\boldsymbol{i} : (i_1, i_2, \ldots, i_N)$ will be a good estimation of $x_{\boldsymbol{i}}$.

The sampling algorithm is shown in Algorithm 1.

2.2 Probability of picking edge and walking to other partitions

In this part, we introduce the probabilities in each phase.

- Walking with probability (lines 8–10 of Algorithm 1)

  In phase 2 we start from a vertex in $\overline{V}$ to V_i, and phase 3 from a vertex in V_1 to $\overline{V}$. We choose the path(edge) that we walk through according to its weight. That is picking $r \in \{1, 2, \ldots, R\}$ with probability $|a_{i_1 r'}^{(1)}| / \parallel \boldsymbol{a}_{i_1 *}^{(1)} \parallel_1$ or given r, picking $i_n \in \{1, 2, \ldots, L_n\}$ with probability $|a_{i_n r}^{(n)}| / \parallel \boldsymbol{a}_{*r}^{(n)} \parallel_1$.

- Picking an edge (line 6 of Algorithm 1)

  When we pick an edge $e = (v_{i_1}^1, \overline{v}_r)$ in phase 1, we are picking the vertexes pair $(v_{i_1}^1, \overline{v}_r)$. Beforehand, we assign each pair a probability while $a_{i_1 k}^{(1)} \neq 0$:

$$p(i_1, r) = \mid a_{i_1 k}^{(1)} \mid \parallel \boldsymbol{a}_{i_1 *}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{*r}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{*r}^{(2)} \parallel_1 \ldots \parallel \boldsymbol{a}_{*r}^{(N)} \parallel_1 / \parallel \mathbf{W} \parallel_1$$

Algorithm 1 Diamond Sampling with factor matrixes

Given factor matrix $\mathbf{A}^{(n)} \in R^{L_n \times R}, n = 1, 2, \ldots, N$.
Let s be the number of samples.

1: **for** all $a^{(1)}_{i_1 r} \neq 0$ **do**
2: $\quad w_{i_1 r} \leftarrow\mid a^{(1)}_{i_1 r} \mid\mid\mid \boldsymbol{a}^{(1)}_{i_1 *} \parallel_1 \parallel \boldsymbol{a}^{(1)}_{*r} \parallel_1 \parallel \boldsymbol{a}^{(2)}_{*r} \parallel_1 \ldots \parallel \boldsymbol{a}^{(N)}_{*r} \parallel_1$
3: **end for**
4: $\mathcal{X} \leftarrow$ all-zeros tensor of size $L_1 \times L_2 \ldots \times L_N$
5: **for** $\ell = 1, \ldots, s$ **do**
6: $\quad$ Sample (i_1, r) with probability $w_{i_1 r} / \parallel \mathbf{W} \parallel_1$
7: $\quad$ **for** $n = 2, ..., N$ **do**
8: $\quad\quad$ Sample i_n with probability $|a^{(n)}_{i_n r}| / \parallel \boldsymbol{a}^{(n)}_{*r} \parallel_1$
9: $\quad$ **end for**
10: $\quad$ Sample r' with probability $|a^{(1)}_{i_1 r'}| / \parallel \boldsymbol{a}^{(1)}_{i_1 *} \parallel_1$
11: $\quad x_{i_1, i_2, \cdots, i_N} \leftarrow x_{i_1, i_2, \cdots, i_N} + sgn(a^{(1)}_{i_1 r} \cdot a^{(2)}_{i_2 r} \cdots a^{(N)}_{i_N r} \cdot a^{(1)}_{i_1 r'}) a^{(2)}_{i_2 r'} \cdots a^{(N)}_{i_N r'}$
12: **end for**

Where

$$\parallel \mathbf{W} \parallel_1 = \sum_{i_1, r} \mid a^{(1)}_{i_1 k} \mid\mid\mid \boldsymbol{a}^{(1)}_{i_1 *} \parallel_1 \parallel \boldsymbol{a}^{(1)}_{*r} \parallel_1 \parallel \boldsymbol{a}^{(2)}_{*r} \parallel_1 \ldots \parallel \boldsymbol{a}^{(N)}_{*r} \parallel_1$$

Then we pick the pair $(v^1_{i_1}, \overline{v}_r)$ according the probability $p(i_1, r)$.

2.3 Scoring samples

Totally, we do s times sampling, and each sample we will get an coordinate $\boldsymbol{i} = (i_1, i_2, \ldots, \imath_N)$. If this coordinate has not been sampled previously, let the score of this coordiante in the ℓ-th turn be

$$\mathbf{X}_{\boldsymbol{i},\ell} = sgn(a^{(1)}_{i_1 r} \cdot a^{(2)}_{i_2 r} \cdots a^{(N)}_{i_N r} \cdot a^{(1)}_{i_1 r'}) a^{(2)}_{i_2 r'} \cdots a^{(N)}_{i_N r'},$$

and add $\widehat{x}_{\boldsymbol{i}} = \mathbf{X}_{\boldsymbol{i},\ell}$ into a set where we save the scores. Otherwise, increase $\widehat{x}_{\boldsymbol{i}}$ in the set by $\mathbf{X}_{\boldsymbol{i},\ell}$. It is shown in line 11 of Algorithm 1. For $\boldsymbol{i}$ that not be sampled in ℓ-th turn, we can assume that $\mathbf{X}_{\boldsymbol{i},\ell} = 0$. So

$$\widehat{x}_{\boldsymbol{i}} = \sum_{\ell} \mathbf{X}_{\boldsymbol{i},\ell}$$

In the next part, we will show that $\widehat{x}_{\boldsymbol{i}}$ is a good estimation of $x_{\boldsymbol{i}}$ in scale.

2.4 Correctness and error bounds

As we defined previously, the event $\varepsilon_{\boldsymbol{i},r',r}$ is picking a pair $(v^1_{i_1}, \overline{v}_r)$ then pick paths from V_1 to $\overline{V}$ and some addition pathes from $\overline{V}$ to $V_i, i \in 2, \ldots, N$. And we assume that $\mathbf{X}_{\boldsymbol{i},\ell}, \ell \in \{1, 2 \ldots, s\}$ are independent. Under these assumptions, we give two lemmas.

Lemma 2.1. *The expectation of $\widehat{x}_{\boldsymbol{i}}$ equals to $s \cdot x^2_{\boldsymbol{i}} / \parallel \mathbf{W} \parallel_1$.*

Proof: The final score $\widehat{x}_{\boldsymbol{i}} = \sum_{\ell=1} \mathbf{X}_{\boldsymbol{i},\ell}$. And

$$\mathbb{E}[\widehat{x}_{\boldsymbol{i}}] = \mathbb{E}[\sum_{\ell=1} \mathbf{X}_{\boldsymbol{i},\ell}] = s\mathbb{E}[\mathbf{X}_{\boldsymbol{i},1}] \tag{3}$$

The probability of $\varepsilon_{\boldsymbol{i},r',r}$ is

$$\begin{aligned}
Pr(\varepsilon_{\boldsymbol{i},r',r}) &= Pr(\text{pick } (v^1_{i_1}, \overline{v}_r)) \cdot Pr(\text{walk to } \overline{v}'_r | \text{given } v^1_{i_1}) \cdot Pr(\text{walk to } v^2_{i_2} | \text{given } \overline{v}_r) \cdots Pr(\text{walk to } v^N_{i_N} | \text{given } \overline{v}_r) \\
&= \frac{w_{i_1 r}}{\parallel \mathbf{W} \parallel_1} \cdot \frac{|a^{(1)}_{i_1 r'}|}{\parallel \boldsymbol{a}^{(1)}_{i_1 *} \parallel_1} \cdot \frac{|a^{(2)}_{i_2 r}|}{\parallel \boldsymbol{a}^{(2)}_{*r} \parallel_1} \cdots \frac{|a^{(N)}_{i_N r}|}{\parallel \boldsymbol{a}^{(N)}_{*r} \parallel_1} \\
&= \frac{|a^{(1)}_{i_1 r}| \parallel \boldsymbol{a}^{(1)}_{i_1 *} \parallel_1 \parallel \boldsymbol{a}^{(1)}_{*r} \parallel_1 \cdots \parallel \boldsymbol{a}^{(N)}_{*r} \parallel_1}{\parallel \mathbf{W} \parallel_1} \cdot \frac{|a^{(2)}_{i_2 r'}|}{\parallel \boldsymbol{a}^{(2)}_{i_2 *} \parallel_1} \cdot \frac{|a^{(1)}_{i_1 r}|}{\parallel \boldsymbol{a}^{(1)}_{*r} \parallel_1} \cdots \frac{|a^{(N)}_{i_N r}|}{\parallel \boldsymbol{a}^{(N)}_{*r} \parallel_1} \\
&= \frac{|a^{(1)}_{i_1 r'} a^{(1)}_{i_1 r} \cdots a^{(N)}_{i_N r}|}{\parallel \mathbf{W} \parallel_1}
\end{aligned}$$

We get the probability of one walk:

$$Pr(\varepsilon_{\boldsymbol{i},r',r}) = \frac{|a^{(1)}_{i_1 r'} a^{(1)}_{i_1 r} \cdots a^{(N)}_{i_N r}|}{\parallel \mathbf{W} \parallel_1} \tag{4}$$

Using Equation 3 and Equation 4. The expectation

$$\begin{aligned}
\mathbb{E}[x_{\boldsymbol{i}}/s] &= \mathbb{E}[\mathbf{X}_{\boldsymbol{i},1}] \\
&= \sum_r \sum_{r'} Pr(\varepsilon_{\boldsymbol{i},r',r}) \cdot sgn(a^{(1)}_{i_1 r'} a^{(1)}_{i_1 r} \cdots a^{(N)}_{i_N r}) a^{(2)}_{i_2 r'} \cdots a^{(N)}_{i_N r'} \\
&= \frac{\sum_r \sum_{r'} |a^{(1)}_{i_1 r'} a^{(1)}_{i_1 r} \cdots a^{(N)}_{i_N r}| sgn(a^{(1)}_{i_1 r'} a^{(1)}_{i_1 r} \cdots a^{(N)}_{i_N r}) a^{(2)}_{i_2 r'} \cdots a^{(N)}_{i_N r'}}{\parallel \mathbf{W} \parallel_1} \\
&= \frac{\sum_r \sum_{r'} a^{(1)}_{i_1 r} \cdot a^{(2)}_{i_2 r} \cdots a^{(N)}_{i_N r} \cdot a^{(1)}_{i_1 r'} a^{(2)}_{i_2 r'} \cdots a^{(N)}_{i_N r'}}{\parallel \mathbf{W} \parallel_1} \\
&= \frac{\{\sum_r a^{(1)}_{i_1 r} \cdot a^{(2)}_{i_2 r} \cdots a^{(N)}_{i_N r}\}^2}{\parallel \mathbf{W} \parallel_1} \\
&= \frac{x^2_{\boldsymbol{i}}}{\parallel \mathbf{W} \parallel_1}
\end{aligned}$$

□

Lemma 2.2. *Fix $\varepsilon > 0$ and error probability $\sigma \in (0,1)$. Assuming all entries in factor matrices are nonnegative and at most K. If the number of samples*

$$s \leq 3K^{N-1} \parallel \mathbf{W} \parallel_1 \log(2/\sigma)/(\varepsilon^2 {x_{\boldsymbol{i}}}^2),$$

then

$$Pr[|\widehat{x}_{\boldsymbol{i}} \parallel \mathbf{W} \parallel_1 /s - {x_{\boldsymbol{i}}}^2| \geq \varepsilon {x_{\boldsymbol{i}}}^2] \leq \sigma$$

Proof: Let

$$y_{\boldsymbol{i}} = \sum_\ell \mathbf{Y}_{\boldsymbol{i},\ell} = \sum_\ell \mathbf{X}_{\boldsymbol{i},\ell}/K^{N-1}$$

Where $\mathbf{Y}_{\boldsymbol{i},\ell}$ is in $[0,1]$ for $\mathbf{X}_{\boldsymbol{i},\ell}$ is in $[0, K^{N-1}]$ and $y_{\boldsymbol{i}}$ is a sum of random variables in $[0,1]$. Applying the Chernoff bound,

$$Pr[y_{\boldsymbol{i}} \geq \mathbb{E}[y_{\boldsymbol{i}}]] < \exp(-\epsilon^2 \mathbb{E}[y_{\boldsymbol{i}}]/3)$$

By Lemma **??**

$$\mathbb{E}[y_{\boldsymbol{i}}] = \frac{s x^2_{\boldsymbol{i}}}{K^{N-1} \parallel \mathbf{W} \parallel_1}$$

Algorithm 2 Finding top-t largest value

Given factor matrix $\mathbf{A}^{(n)} \in R^{L_n \times R}, n = 1, 2, \ldots, N$.
Let s be the number of samples, t' be the budget.

1: Sample the score $\widehat{x}_{\boldsymbol{i}}$ using Algorithm 1 and record the coordinates set Ω_s have been sampled.
2: Sort the score to extract

$$\Omega_{t'} = \{\boldsymbol{i} | \widehat{x}_{\boldsymbol{i}} \geq \widehat{x}_{\boldsymbol{i'}}, \forall i' \in \Omega_s \backslash \Omega_{t'}\}$$

3: Compute the actual value $x_{\boldsymbol{i}}$ of each coordinate in $\Omega_{t'}$
4: Sort the actual value to extract

$$\Omega_t = \{\boldsymbol{i} | x_{\boldsymbol{i}} \geq x_{\boldsymbol{i'}}, \forall i' \in \Omega_{t'} \backslash \Omega_t\}$$

By the choice of s we have $\mathbb{E}[y_{\boldsymbol{i}}] = (sx_{\boldsymbol{i}}^2)/(K^{N-1} \parallel \mathbf{W} \parallel_1) \leq 3 \log(2/\sigma)$. Then

$$Pr[y_{\boldsymbol{i}} \geq \mathbb{E}[y_{\boldsymbol{i}}]] < \sigma/2$$

By the substitution of $y_{\boldsymbol{i}} = \sum_\ell \mathbf{X}_{\boldsymbol{i},\ell}/K^{N-1} = \widehat{x}_{\boldsymbol{i}}/K^{N-1}$

$$Pr[\widehat{x}_{\boldsymbol{i}\cdot} \parallel \mathbf{W} \parallel_1 \geq s \cdot x_{\boldsymbol{i}}] < \sigma/2$$

Using the Chernoff lower tail bound and identical reasoning. We get

$$Pr[\widehat{x}_{\boldsymbol{i}\cdot} \parallel \mathbf{W} \parallel_1 /s \leq (1-\epsilon) x_{\boldsymbol{i}}] \leq \sigma/2$$

□

Theorem 2.1. *Fix some threshold τ and error probability $\sigma \in (0,1)$. Assume all entries in factor matrices are nonnegative and at most K. Suppose $s \geq 12K^{N-1} \parallel \mathbf{W} \parallel_1 \log(2L_1 L_2 \cdots L_N/\sigma)/\tau^2$. Then with probability at least $1-\sigma$, the following holds for all indexes $\boldsymbol{i} = (i_1, i_2, \ldots, i_N)$ and $\boldsymbol{i'} = (i'_1, i'_2, \ldots, i'_N)$: if $x_{\boldsymbol{i}} > \tau$ and $x_{\boldsymbol{i'}} < \tau/4$, then $\widehat{x}_{\boldsymbol{i}} > \widehat{x}_{\boldsymbol{i'}}$.*

proof: □

2.5 Finding top-t largest value

We use the sampling score $\widehat{x}_{\boldsymbol{i}}$ and coordinate set $\boldsymbol{i}$ to find the top-t largest value in $\mathcal{X}$ when given N factor matrices. Let $\Omega_s = \{\boldsymbol{i}_j | j = 1, 2, \ldots, s\}$ be the coordinates have been sampled.

To reduce the computing, a pre-sort is carried out. It sort the scores $\widehat{x}_{\boldsymbol{i}}$ and reserve the top-t' elements. Obviously, t',which called budget, is always much larger than t.

Then we compute the actual value $x_{\boldsymbol{i}}$ of the coordinates in the less small set $\Omega_{t'} = \{\boldsymbol{i} | \widehat{x}_{\boldsymbol{i}} \geq \widehat{x}_{\boldsymbol{i'}}, \forall i' \in \Omega_s \backslash \Omega_{t'}\}$. And the top-$t$ largest value's coordinates will be $\Omega_t = \{\boldsymbol{i} | x_{\boldsymbol{i}} \geq x_{\boldsymbol{i'}}, \forall i' \in \Omega_{t'} \backslash \Omega_t\}$.

The reason to do so is that although the score $\widehat{x}$ is a good estimation, the variance is much higher in practical. And the budget t' is a tradeoff between accuracy and computation. The algorithm for finding the top-t largest value is shown in Algorithm 2.

2.6 Finding k-NN for query

In recommendation system, it is demanded to find the most k relevant items for a user. On this occasion, there will be a matrix for all users, for consistency, say $\mathbf{A}^{(1)}$ and $N-1$ matrices $\mathbf{A}^{(n)}, n = 2, \ldots, N$ for items. Each row in matrix $\mathbf{A}^{(1)}$ represents a particular user $\boldsymbol{u}$. And the tenor with $N-1$ order consisting a user $\boldsymbol{u}$ and items matrices is called a ranking tensor(matrix).

$$\mathcal{X}_{\boldsymbol{u}} = \left[\!\left[\boldsymbol{u}, \mathbf{A}^{(2)} \ldots, \mathbf{A}^{(N)}\right]\!\right] = \sum_{r=1}^{R} u_r \cdot \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(N)} \tag{5}$$

It is a exception to reduce one factor matrix into a vector in previous algorithm. However, when we have so many queries, it is time consuming to do sampling one query by one. Obviously, the difference between different queries in sampling processing is the probability for picking vertexes in partition $\overline{V}$, in other word, or the frequency number $(c_1, c_2, \ldots, c_R)$ of each $\overline{v}_r$ that is expected to be sampled.

In the phase 2(walk from a vertex $\overline{v}_r$ in $\overline{V}$ to $V_i, i \in \{2, \ldots, N\}$) of the event $\varepsilon_{\boldsymbol{i},r',r}$, we call the coordinate $\{i_2, \ldots, i_N\}$ a sub-path. For effectively implementation, we use these lists $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_R$ to save the sub-paths of sampled query.

2.6.1 Weight assigning and frequency generating

Instead of pick a random pair $(\boldsymbol{u}, \overline{v}_{\boldsymbol{i}})$, we use the frequency number sequence $(c_1, c_2, \ldots, c_R)$ for phase 1. For each $1 \le r \le R$, let $u'_r \leftarrow\mid u_r \mid\!\mid\!\mid \boldsymbol{a}_{*r}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{*r}^{(2)} \parallel_1 \ldots \parallel \boldsymbol{a}_{*r}^{(N)} \parallel_1$ the same in phase 1. And then choose c_r to have the expected value of $su'_r / \parallel u' \parallel_1$.

$$c_r = \begin{cases} \lfloor su'_r / \parallel u' \parallel_1 \rfloor, & \text{with probability } \lceil su'_r / \parallel u' \parallel_1 \rceil - su'_r / \parallel u' \parallel_1 \\ \lceil su'_r / \parallel u' \parallel_1 \rceil, & \text{with probability } \lfloor su'_r / \parallel u' \parallel_1 \rfloor - su'_r / \parallel u' \parallel_1 \end{cases}$$

2.6.2 Sub-paths

After the choice of c_r, the sampling method repeat c_r times to sample the the rest indexes $(i_1, i_2, \ldots, i_N)$ by randomly walking from the $\overline{v}_r$ to other partitions $V_2, \ldots, V_N$ that is of the same probability between different queries. So the sub-path $(i_1, i_2, \ldots, i_N)$ is used for the next query to save computation. We give the algorithm in Algorithm 3.

3 Implementing details

In section, we introduce some details get the improvement.

3.1 Data Structure

We arrange the factor matrices in specific storage format. we arrange $\mathbf{A}^{(1)}$ to be saved in row major order, and the rest factor matrices in column major order. Also, the sum of each row in $\mathbf{A}^{(1)}$ and the sum of each column in $\mathbf{A}^{(n)}, n = 2, \ldots, N$ will be computed previously.

3.2 Picking the edge

We pick the s edge with probability $w_{i_1 r} / \parallel \mathbf{W} \parallel_1$. The weight matrix $\mathbf{W}$ is saved in row major order also and suppose $\boldsymbol{\rho}$ is the vectorization of $\boldsymbol{w}$. Since s is much larger than the length of $\boldsymbol{\rho}$, we use the binary search on the cumulative of $\boldsymbol{p}$ via Algorithm 4. We use $\boldsymbol{e} = (e_1, \ldots, e_s)$ to record the sampled indexes, and the vertexes pair is $(e_\ell \% L_1, e_\ell \setminus L_1)$. So the sample result is sorted according ro r.

3.3 Random walking

In phase 2, when we walk from $\overline{v}_r$ to other partitions V_n. We use the previous result γ_r, which means we need walk totally γ_r times from $\overline{v}_r$ to each partition. Let $\boldsymbol{\rho} = (|a_1^{(n)} r| / \parallel \boldsymbol{a}_{*r}^{(n)} \parallel_1 , \ldots, |a_{L_n}^{(n)} r| / \parallel \boldsymbol{a}_{*r}^{(n)} \parallel_1)$, then sample γ_r times using vose alias's method.

Algorithm 3 Finding k-NN for a query

Given factor matrix $\mathbf{A}^{(n)} \in R^{L_n \times R}, n = 1, 2, \ldots, N.\mathbf{A}^{(1)}$ is the query matrix.
Let s be the number of samples, t' be the budget.

Initialize R empty sub-paths lists $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_R$ for each $\overline{v}_r$.
for $i_1 = 1, 2, \ldots, L_1$ **do**
  Let query $\boldsymbol{u} = \boldsymbol{a}^{(1)}_{i_1 *}$
  for $r = 1, \ldots, R$ **do**
    Let $u'_r \leftarrow \mid u_r \mid\mid\mid \boldsymbol{a}^{(1)}_{*r} \parallel_1 \parallel \boldsymbol{a}^{(2)}_{*r} \parallel_1 \ldots \parallel \boldsymbol{a}^{(N)}_{*r} \parallel_1$
  end for
  for $r = 1, \ldots, R$ determine c_r **do**

$$c_r = \begin{cases} \lfloor s u'_r / \parallel u' \parallel_1 \rfloor, & \text{with probability } \lceil s u'_r / \parallel u' \parallel_1 \rceil - s u'_r / \parallel u' \parallel_1 \\ \lceil s u'_r / \parallel u' \parallel_1 \rceil, & \text{with probability } \lfloor s u'_r / \parallel u' \parallel_1 \rfloor - s u'_r / \parallel u' \parallel_1 \end{cases}$$

  end for
  for $r = 1, \ldots, R$ **do**
    if $c_r \leq |\boldsymbol{g}_r|$ **then**
      Use c_r sub-paths in $\boldsymbol{g}_r$ as the sampled coordinates.
    else
      Sample $n = c_r - |\boldsymbol{g}_r|$ sub-paths and append it into $\boldsymbol{g}_r$.
    end if
    Use the method in Algorithm 2 to find the top-k largest value for query $\boldsymbol{u}$.
  end for
end for

Algorithm 4 Picking Edge

Given probability $\mathbf{W} \in R^{L_1 \times R}$,$\boldsymbol{\rho} \in [0, 1]^p$ is the vectorization, and number of samples s. The result saved in $\boldsymbol{e}$

Sample $\mu_\ell \sim U(0, 1)$ for $\ell \in \{1, \ldots, s\}$
Sort $\boldsymbol{\mu}$ so that $\mu_1 \leq \cdots \leq \mu_s$
$k \leftarrow 1, \overline{\mu} \leftarrow \mu_k$,$\boldsymbol{\gamma} \in R^R \leftarrow$ all-zero vector.
for $\ell = 1, \ldots, s$ **do**
  while $\mu_\ell > \overline{\mu}$ **do**
    $k \leftarrow k + 1, \overline{\mu} \leftarrow \mu_k$
  end while
  $r = k \setminus L_1$
  $e_\ell \leftarrow k$,$\gamma_r \leftarrow \gamma_r + 1$
end for