Sampling for Approximate Maximum Search in Factorized Tensor

Zhi Lu and **Yang Hu**[*] and **Bing Zeng**
School of Electronic Engineering
University of Electronic Science and Technology of China
zhilu@std.uestc.edu.cn, {yanghu,eezeng}@uestc.edu.cn

Abstract

Factorization models have been extensively used for recovering the missing entries of a matrix or tensor. However, directly computing all of the entries using the learned factorization models is prohibitive when the size of the matrix/tensor is large. On the other hand, in many applications, such as collaborative filtering, we are only interested in a few entries that are the largest among them. In this work, we propose a sampling-based approach for finding the top entries of a tensor which is decomposed by the CANDECOMP/PARAFAC model. We develop an algorithm to sample the entries with probabilities proportional to their values. We further extend it to make the sampling proportional to the k-th power of the values, amplifying the focus on the top ones. We provide theoretical analysis of the sampling algorithm and evaluate its performance on several real-world data sets. Experimental results indicate that the proposed approach is orders of magnitude faster than exhaustive computing. When applied to the special case of searching in a matrix, it also requires fewer samples than the other state-of-the-art method.

1 Introduction

Matrix or tensor completion has received considerable attention in recent years. Many problems in application can be formulated as recovering the missing entries of a matrix or tensor. In some scenarios, such as image in-painting [Bertalmio *et al.*, 2000], all lost entries are needed to be filled in. In some others, however, it is difficult and unnecessary to recover them all. Take recommender systems for example, the size of the matrix/tensor, determined by the numbers of users and items, is usually rather large. It is expensive computationally to compute all of the unknown values. On the other hand, for the recommendation purpose, we are only interested in a few entries that are the largest within a sub-array of the matrix/tensor. These entries are not only the central concerns for a personalized recommendation, but also meaningful in many other cases. In a similarity matrix, the top entries correspond to pairs of items that are most similar, which are of interest for applications like link prediction in graph [Dunlavy *et al.*, 2011], duplicate detection [Ke *et al.*, 2004] as well as information retrieval [Salton and Harman, 2003].

In this work, we study the problem of efficiently identifying the top entries of a factorized tensor. Tensors, as multiway generalizations of matrices, hav been exploited more and more recently. Take recommender systems for example, while the traditional focus is on the user-item matrix [Koren *et al.*, 2009], tensor is required for data representation in many emerging settings such as context-aware recommendation [Xiong *et al.*, 2010; Adomavicius and Tuzhilin, 2015], and set-based recommendation [Hu *et al.*, 2015] where the object to be recommended is a set of items such as a set of clothes that make an outfit.

Factorization models are essential tools for analyzing tensors. They assume that the tensor can be decomposed into some factors, which can be estimated from the observed entries by learning algorithms. Specifically, we focus on the CANDECOMP/PARAFAC [Kolda and Bader, 2009] decomposition model, a widely used technique for exploring and extracting the underlying structure of multi-way data. Given an N-order tensor $\mathcal{X} \in \mathbb{R}^{L_1 \times \cdots \times L_N}$, the CP decomposition approximates it by N factor matrices:

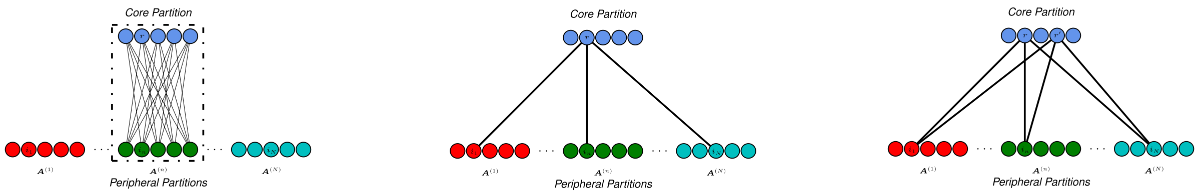
$$\mathcal{X} \approx [\![\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \cdots, \boldsymbol{A}^{(N)}]\!] = \sum_{r=1}^{R} \boldsymbol{a}_{*r}^{(1)} \circ \boldsymbol{a}_{*r}^{(2)} \circ \cdots \circ \boldsymbol{a}_{*r}^{(N)} \tag{1}$$

where each factor matrix $\boldsymbol{A}^{(n)} = [\boldsymbol{a}_{*1}^{(n)} \boldsymbol{a}_{*2}^{(n)} \cdots \boldsymbol{a}_{*R}^{(n)}], n = 1, \ldots, N$, is of size $L_n \times R$ with $\boldsymbol{a}_{*r}^{(n)} \in \mathbb{R}^{L_n}$ being the r-th column. The symbol "$\circ$" represents the vector outer product. R is the tensor rank, indicating the number of latent factors. Element-wise, Eq.(1) is written as:

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

where $\boldsymbol{i}$ is short for the index tuple $(i_1, i_2, \ldots, i_N)$. Given the factor matrices $\boldsymbol{A}^{(n)}$ and a parameter t, we would like to find t index tuples $\{\boldsymbol{i}_1, \ldots, \boldsymbol{i}_t\}$ that correspond to the top-t largest $x_{\boldsymbol{i}}$.

[*]The corresponding author is Yang Hu. This work was supported by the National Natural Science Foundation of China (61602090) and the 111 Project (B17008).



(a) The graph representation of factor matrices. (b) A sampled star. (c) Two intersecting stars.

Figure 1: Graph representation of factor matrices and sampled stars.

2 Related Work

Our problem subsumes many existing problems in the literature. When $N = 2$, it is exactly the MAD (Maximum All-pairs Dot-product Search) problem [Ballard *et al.*, 2015] that finds the largest entries in the product of two matrices. Notice that MAD contains the MIPS (Maximum Inner Product Search) [Coh, 1997] problem as the special case with one matrix being a single column.

The most obvious solution to the problem is to compute the entries exhaustively. However, this becomes prohibitive as the sizes of factor matrices grow. There is some literature in approximate matrix multiplication. However, these methods are not suited even for MAD, since only a few entries among millions are of interest. The more efficient solution is to directly search the top ones, which has been extensively studied for the MIPS problem. Popular approaches include LSH (Locality Sensing Hashing) [Andoni and Indyk, 2008; Shrivastava and Li, 2014], space partition techniques like k-d tree, and sampling-based approaches [Drineas *et al.*, 2006; Holodnak and Ipsen, 2015]. Recently, Ballard *et al.* proposed a randomized approach called diamond sampling [Ballard *et al.*, 2015] to the MAD problem. [Higham and Relton, 2016] derived an algorithms for estimating the largest elements of a matrix using a few matrix-vector products with it and its conjugate transpose. For tensor, however, there has not been any study conducted yet.

Inspired by the work of diamond sampling, we present an index sampling method for a factorized tensor, whose entries are computed by the CP decomposition model. We design a strategy to sample the index tuple $\boldsymbol{i}$ proportional to the k-th power of the entries, which amplifies the focus on the largest ones. We provide theoretical analysis for our sampling algorithm and derive concentration bound on its behavior. For applications in recommender systems, an algorithm that reuses the samples for multiple users is presented. We evaluate the proposed method on several real-world data sets. The results show that it is orders of magnitude faster than exact computing. When compared to previous approach for matrix sampling, our method requires much fewer samples.

3 Star Sampling

A graph is used to represent the factor matrices of CP decomposition. Each matrix $\boldsymbol{A}^{(n)} \in \mathbb{R}^{L_n \times R}, n \in [N]$, is represented by a bipartite graph as shown in the dashed box in Figure 1 (a). The L_n rows of $\boldsymbol{A}^{(n)}$, indexed by i_n, correspond to nodes on the bottom, and the R columns of $\boldsymbol{A}^{(n)}$, indexed by r, correspond to nodes on the top. Edge (i_n, r) exists if $a^{(n)}_{i_n r}$ is nonzero. Since all of the factor matrices have the same number of columns, they can share nodes for the columns. Therefore, the N factor matrices are represented by an (N+1)-partite graph with one partition, called the core partition, consisting of nodes for the common columns, and N peripheral partitions, each of which consists of nodes for the rows of a matrix.

To motivate our sampling procedure, we start with the case where all $\boldsymbol{A}^{(n)}$ are binary matrices. If a core node r in core partition has a neighbor i_n, i.e. $a^{(n)}_{i_n r} = 1$, in each peripheral partition, we call subgraph $(r, i_1, \dots, i_N)$ (shorted as $(r, \boldsymbol{i})$), a "star" (Figure 1 (b)). According to Eq.(2), $x_{\boldsymbol{i}}$ is simply the number of distinct stars connecting the same group of neighbors in the peripheral partitions indexed by $\boldsymbol{i}$. It can be shown that the probability of selecting a random star with endpoints $\boldsymbol{i}$ is proportional to $x_{\boldsymbol{i}}$ [Ballard *et al.*, 2015].

We further consider a "compound star", which is formed by two intersecting stars $(r, \boldsymbol{i})$ and $(r', \boldsymbol{i})$ (Figure 1 (c)). There are $x^2_{\boldsymbol{i}}$ compound stars with endpoints $\boldsymbol{i}$. The probability of selecting a random compound star with endpoints $\boldsymbol{i}$ is proportional to $x^2_{\boldsymbol{i}}$. More generally, if a k-compound star is formed by k intersecting stars, its selected probability will be proportional to $x^k_{\boldsymbol{i}}$.

Sampling random k-compound stars will accelerate identifying the top entries as compared to sampling basic stars but with higher complexity. Besides, the factor matrices $\boldsymbol{A}^{(n)}$ are usually real-valued, which requires the sampling to have a nonuniform distribution. In the following, we present a sampling algorithm that handles these issues carefully.

3.1 Sampling A Star

To sample k-compound stars, we take node set that contains k nodes from the core partition as a k-compound core node.

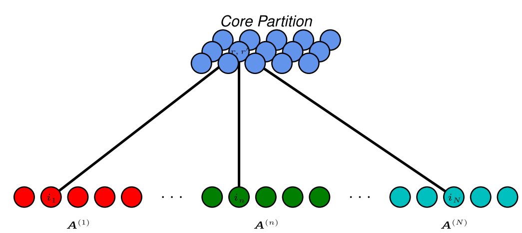


Figure 2: A sampled k-compound star with $k = 2$.

Algorithm 1 The Core^k Sampling Method

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

for $\boldsymbol{r} \in \underbrace{[R] \times \cdots \times [R]}_{k}$ **do**
  for $n = 1, ..., N$ **do**
    for $i_n = 1, \ldots, L_n$ **do**
      $e^{(n)}_{i_n \boldsymbol{r}} \leftarrow a^{(n)}_{i_n r_1} \cdots a^{(n)}_{i_n r_k}$
    end for
    $\|\boldsymbol{e}^{(n)}_{*\boldsymbol{r}}\|_1 \leftarrow \sum_{i_n} |e^{(n)}_{i_n \boldsymbol{r}}|$
  end for
  $w_{\boldsymbol{r}} \leftarrow \|\boldsymbol{e}^{(1)}_{*\boldsymbol{r}}\|_1 \cdots \|\boldsymbol{e}^{(N)}_{*\boldsymbol{r}}\|_1$
end for
for $\ell = 1, \ldots, s$ **do**
  Sample $\boldsymbol{r}$ with probability $w_{\boldsymbol{r}} / \|\boldsymbol{w}\|_1$
  for $n = 1, ..., N$ **do**
    Sample i_n with probability $|e^{(n)}_{i_n \boldsymbol{r}}| / \|\boldsymbol{e}^{(n)}_{*\boldsymbol{r}}\|_1$
  end for
  $\xi_{\boldsymbol{i},\ell} \leftarrow sign(e^{(1)}_{i_1 \boldsymbol{r}} \cdots e^{(N)}_{i_N \boldsymbol{r}})$
  if $\boldsymbol{i} = (i_1, i_2, \ldots, i_N)$ has not been sampled **then**
    Create $\hat{x}_{\boldsymbol{i}} \leftarrow \xi_{\boldsymbol{i},\ell}$
  else
    $\hat{x}_{\boldsymbol{i}} \leftarrow \hat{x}_{\boldsymbol{i}} + \xi_{\boldsymbol{i},\ell}$
  end if
end for

There are R^k k-compound nodes in total. Figure 2 shows a sampled star with two core nodes (r and r') coalesced to a single compound core node (r, r'). We weigh the edge between a k-compound core node $\boldsymbol{r} = (r_1, \ldots, r_k)$ and a node i_n in the n-th peripheral partition by

$$e^{(n)}_{i_n \boldsymbol{r}} = a^{(n)}_{i_n r_1} \cdots a^{(n)}_{i_n r_k} \tag{3}$$

Let $\boldsymbol{e}^{(n)}_{*\boldsymbol{r}} = [e^{(n)}_{1\boldsymbol{r}}, \cdots, e^{(n)}_{L_n \boldsymbol{r}}]^T$ be a vector containing weights from $\boldsymbol{r}$ to all nodes in the n-th peripheral partition, we assign weight for the k-compound node $\boldsymbol{r}$ as

$$w_{\boldsymbol{r}} = \|\boldsymbol{e}^{(1)}_{*\boldsymbol{r}}\|_1 \cdots \|\boldsymbol{e}^{(N)}_{*\boldsymbol{r}}\|_1 \tag{4}$$

where $\| \cdot \|_1$ denotes 1-norm of a vector.

To find a k-compound star, we first select a k-compound node with probability $w_{\boldsymbol{r}} / \|\boldsymbol{w}\|_1$, where $\boldsymbol{w}$ is the vector containing all $w_{\boldsymbol{r}}$. Then, conditioned on the sampled compound node, we pick one node from each of the peripheral partitions. The node in the n-th peripheral partition is drawn with probability $|e^{(n)}_{i_n \boldsymbol{r}}| / \|\boldsymbol{e}^{(n)}_{*\boldsymbol{r}}\|_1$. After an index tuple $\boldsymbol{i} = (i_1, i_2, \ldots, i_N)$ is obtained, a score $\xi_{\boldsymbol{i},\ell}$ is computed:

$$\xi_{\boldsymbol{i},\ell} \leftarrow sign(e^{(1)}_{i_1 \boldsymbol{r}} \cdots e^{(N)}_{i_N \boldsymbol{r}}) \tag{5}$$

where ℓ denotes the ℓ-th sample. If the index tuple $\boldsymbol{i}$ has not been sampled before, a container is created with $\hat{x}_{\boldsymbol{i}} = \xi_{\boldsymbol{i},\ell}$. Otherwise, we increase $\hat{x}_{\boldsymbol{i}}$ by $\xi_{\boldsymbol{i},\ell}$. The full procedure is shown in Algorithm 1.

3.2 Top-t Extraction

Let $\Psi_p = \{\boldsymbol{i}_p | p \in [P]\}$ denotes the set of index tuples that have been sampled. $P \leq s$ since some $\boldsymbol{i}$ may be picked more than once. Two strategies for identifying the t largest entries from Ψ_p are presented. The first one is directly computing the exact entry value $x_{\boldsymbol{i}}$ for each sample in Ψ_p using Eq.(2) and then extracting the t largest ones. Following the previous works [Ballard *et al.*, 2015; Coh, 1997], in the second strategy, we utilize the scores $\hat{x}_{\boldsymbol{i}}$ obtained during sampling to do prefiltering since the load for computing $\hat{x}_{\boldsymbol{i}}$ is much lighter than that of computing $x_{\boldsymbol{i}}$. Moreover, as we show through theoretical analysis later, the expectation of $\hat{x}_{\boldsymbol{i}}$ is proportional to the k-th power of $x_{\boldsymbol{i}}$. Therefore, $\hat{x}_{\boldsymbol{i}}$ can be used to filter out the relatively small ones in Ψ_p. Specifically, we denote a subset by $\Psi_{t'}$ that contains t' index tuples corresponding to the top-t' largest $\hat{x}_{\boldsymbol{i}}$ after the prefiltering. Then, the exact values are computed only for index tuples in $\Psi_{t'}$, and the t index tuples with the largest $x_{\boldsymbol{i}}$ in $\Psi_{t'}$ are taken as the output of our algorithm.

In the following part, we prove that in Algorithm 1, the probability that an index tuple $\boldsymbol{i}$ is sampled is proportional to $x^k_{\boldsymbol{i}}$ and the expectation of $\hat{x}_{\boldsymbol{i}}$ is also proportional to the k-th power of $x_{\boldsymbol{i}}$.

3.3 Theoretical Analysis

Let $\epsilon_{\boldsymbol{i},\boldsymbol{r}}$ denote the event of choosing compound node $\boldsymbol{r}$ and peripheral nodes $\boldsymbol{i}$ in one iteration, while $\epsilon_{\boldsymbol{i}}$ be that of choosing $\boldsymbol{i}$. We first analyze $p(\epsilon_{\boldsymbol{i}})$, the probability that an entry is sampled, and then compute the expectation of $\hat{x}_{\boldsymbol{i}}$. We also provide error bound of the estimate.

Lemma 1. *Suppose that all elements of the factor matrices* $\boldsymbol{A}^{(n)}, n \in [N]$, *are nonnegative. We have* $p(\epsilon_{\boldsymbol{i}}) = x^k_{\boldsymbol{i}} / \|\boldsymbol{w}\|_1$.

Proof. The probability $p(\epsilon_{\boldsymbol{i}})$ is marginal distribution of $p(\epsilon_{\boldsymbol{i},\boldsymbol{r}})$, which can be computed by:

$$\begin{aligned} p(\epsilon_{\boldsymbol{i}}) &= \sum_{\boldsymbol{r}} p(\epsilon_{\boldsymbol{i},\boldsymbol{r}}) = \sum_{\boldsymbol{r}} \frac{w_{\boldsymbol{r}}}{\|\boldsymbol{w}\|_1} \frac{|e^{(1)}_{i_1 \boldsymbol{r}}|}{\|\boldsymbol{e}^{(n)}_{*\boldsymbol{r}}\|_1} \cdots \frac{|e^{(N)}_{i_N \boldsymbol{r}}|}{\|\boldsymbol{e}^{(N)}_{*\boldsymbol{r}}\|_1} \\ &= \frac{\prod_{p=1}^k (\sum_{r_p} a^{(1)}_{i_1 r_p} \cdots a^{(N)}_{i_N r_p})}{\|\boldsymbol{w}\|_1} = \frac{x^k_{\boldsymbol{i}}}{\|\boldsymbol{w}\|_1} \end{aligned}$$

□

Let $c_{\boldsymbol{i},\ell}$ be a random variable that if $\boldsymbol{i}$ has been picked in the ℓ-th iteration, $c_{\boldsymbol{i},\ell} = \xi_{\boldsymbol{i},\ell}$. Otherwise $c_{\boldsymbol{i},\ell} = 0$. The final score $\hat{x}_{\boldsymbol{i}}$ can be written as $\hat{x}_{\boldsymbol{i}} = \sum_{\ell=1}^s c_{\boldsymbol{i},\ell}$.

Lemma 2. *The expectation of* $\hat{x}_{\boldsymbol{i}}$ *equals to* $s x^k_{\boldsymbol{i}} / \|\boldsymbol{w}\|_1$.

Proof. Since $c_{\boldsymbol{i},\ell}$ are i.i.d for fixed $\boldsymbol{i}$ and varying ℓ, the expectation of $\hat{x}_{\boldsymbol{i}}$ is:

$$\mathbb{E}[\hat{x}_{\boldsymbol{i}}] = \sum_{\ell=1}^s \mathbb{E}[c_{\boldsymbol{i},\ell}] = \sum_{\ell=1}^s (\sum_{\boldsymbol{r}} p(\epsilon_{\boldsymbol{i},\boldsymbol{r}}) \xi_{\boldsymbol{i},\ell}) = \frac{s x^k_{\boldsymbol{i}}}{\|\boldsymbol{w}\|_1}$$

□

Theorem 1. *Fix* $\delta > 0$ *and error probability* $\sigma \in (0, 1)$. *Assume that all entries in the factor matrices are nonnegative. If the number of samples*

$$s \geq 3 \|\boldsymbol{w}\|_1 \log(2/\sigma) / (\delta^2 x^k_{\boldsymbol{i}}) \tag{6}$$

then

$$Pr(|\frac{\hat{x}_{\boldsymbol{i}} \cdot \|\boldsymbol{w}\|_1}{s} - x^k_{\boldsymbol{i}}| > \delta x^k_{\boldsymbol{i}}) \leq \sigma$$

Proof. Since $c_{\boldsymbol{i},1}, \cdots, c_{\boldsymbol{i},s}$ are independent random variables taking values in $\{0, 1\}$. The Chernoff bounds on the sum of Poisson trials shows $Pr(|\hat{x}_{\boldsymbol{i}} - \mu| \geq \delta\mu) \leq 2\exp(-\mu\delta^2/3), \delta \in (0, 1)$, where $\mu = sx_{\boldsymbol{i}}^k / \|\boldsymbol{w}\|_1$. By the choice of s, we have $\mu \leq 3\log(2/\sigma)/\delta^2$. Then, $Pr(|\hat{x}_{\boldsymbol{i}} - sx_{\boldsymbol{i}}^k/\|\boldsymbol{w}\|_1| \geq \delta s x_{\boldsymbol{i}}^k/\|\boldsymbol{w}\|_1) \leq \sigma$. Multiplying $\|\boldsymbol{w}\|_1/s$ inside $\Pr[\cdot]$ proofs the bound. □

4 Sampling for Recommender Systems

In this section, we present an algorithm for efficiently identifying top entries in sub-arrays of a tensor, which is of particular interest for recommender systems. Without loss of generality, we assume that the first factor matrix $\boldsymbol{A}^{(1)}$ corresponds to users, with each row characterizing a single user. For any user $\boldsymbol{u} = \boldsymbol{a}^{(1)}_{i_1*}$, we are interested in top entries in the sub-tensor $\mathcal{X}_{\boldsymbol{u}}$, whose CP decomposition is

$$\mathcal{X}_{\boldsymbol{u}} \approx [\![\boldsymbol{u}, \boldsymbol{A}^{(2)}, \cdots, \boldsymbol{A}^{(N)}]\!] \quad (7)$$

We find that for different users, only the probabilities for sampling k-compound nodes from the core partition are different. In the following step, they share the same distribution for sampling the peripheral nodes. Based on this observation, for each k-compound node in the core partition, we build a pool containing sub-index tuples $\boldsymbol{i}' = (i_2, \ldots, i_N)$ that have been picked given that node. For a new user, when a k-compound node is chosen, we can directly use the indices kept in its pool and only sample new $\boldsymbol{i}'$ when necessary. By sharing among users, a huge number of samples can be saved. This procedure is illustrated in Algorithm 2.

Algorithm 2 Finding top-t entries for multiple users

Given factor matrix $\boldsymbol{A}^{(n)} \in \mathbb{R}^{L_n \times R}, n = 1, 2, \ldots, N$. $\boldsymbol{A}^{(1)}$ contains vectors characterizing users. Let s be the number of samples, and $m = R^k$.

Initialize m empty pools $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_m$
Initialize $f_{\boldsymbol{r}} = 0$ for all compound nodes $\boldsymbol{r}$.
for $i_1 = 1, 2, \ldots, L_1$ **do**
  Let the current user be $\boldsymbol{u} = \boldsymbol{a}^{(1)}_{i_1*}$
  for all $\boldsymbol{r} = (r_1, \ldots, r_k)$ **do**
    $w_{\boldsymbol{r}} \leftarrow |u_{r_1} \cdots u_{r_k}| \|\boldsymbol{e}^{(2)}_{*\boldsymbol{r}}\|_1 \cdots \|\boldsymbol{e}^{(N)}_{*\boldsymbol{r}}\|_1$
  end for
  for $\ell = 1, \ldots, s$ **do**
    Sample $\boldsymbol{r}$ with probability $w_{\boldsymbol{r}}/\|\boldsymbol{w}\|_1$.
    Increase $f_{\boldsymbol{r}}$ by 1.
  end for
  for all $\boldsymbol{r}$ **do**
    if $f_{\boldsymbol{r}} > |\boldsymbol{g}_{\boldsymbol{r}}|$ **then**
      Sample $f_{\boldsymbol{r}} - |\boldsymbol{g}_{\boldsymbol{r}}|$ index tuples $\boldsymbol{i}'$ and append them to $\boldsymbol{g}_{\boldsymbol{r}}$.
    end if
    Use $f_{\boldsymbol{r}}$ index tuples $\boldsymbol{i}'$ in $\boldsymbol{g}_{\boldsymbol{r}}$ and compute scores.
  end for
  Post-processing for finding top-t entries of $\boldsymbol{u}$.
end for

Data	L_1	L_2	L_3	Top-1	Top-10^3
ML-S	456	1,973	1,222	81.564	43.385
ML-L	993	3,298	2,555	88.737	50.595
LastFM	1,348	6,927	2,132	234.730	101.795
Delicious	1,681	29,540	7,251	95.472	40.828
dblp	43,928	2,572	17	4.190	2.452

Table 1: Statistics for each data set. The last two columns show the largest and 1000-th largest entry value in the tensors.

5 Implementation Details

In this section, we discuss some details that could improve the performance. Follow the previous work [Coh, 1997], instead of first picking a node from core partition and then choosing N peripheral nodes in each iteration, we directly get s core nodes in one loop. Specifically, we compute the expected number of occurrence for each compound node. Let $\mu_{\boldsymbol{r}} = sw_{\boldsymbol{r}}/\|\boldsymbol{w}\|_1$, the count for $\boldsymbol{r}$ is:

$$f_{\boldsymbol{r}} = \begin{cases} \lfloor \mu_{\boldsymbol{r}} \rfloor, & \text{with probability } p = \lceil \mu_{\boldsymbol{r}} \rceil - \mu_{\boldsymbol{r}} \\ \lceil \mu_{\boldsymbol{r}} \rceil, & \text{with probability } p = \lfloor \mu_{\boldsymbol{r}} \rfloor - \mu_{\boldsymbol{r}} \end{cases} \quad (8)$$

This requires $O(R^k)$ work while sampling $\boldsymbol{r}$ s times (line 10 of Algorithm 1) requires $O(sk\log R)$ work.

To conduct Core^k sampling, we may extend the factor matrices to $\boldsymbol{E}^{(n)}$ whose elements are from Eq.(3). Then, we can just conduct Core^1 sampling with $\boldsymbol{E}^{(n)}$ which requires $O(L_n R^k)$ space for each $n \in [N]$. This is costly in practice when k is large. On the other hand, given a core node $\boldsymbol{r}$, the sampling of the peripheral nodes will only depend on one column in $\boldsymbol{E}^{(n)}$ indexed by $\boldsymbol{r}$. Therefore, instead of storing the whole $\boldsymbol{E}^{(n)}$, we only keep one column of it at a time. This requires $O(L_1 + \cdots + L_N)$ space in total. When k is small, we can directly store $\boldsymbol{E}^{(n)}$ in exchange for speed.

6 Experiments

We evaluate our algorithms on five real-world data sets: Delicious Bookmarks[1](Delicious), Last.FM[2](LastFM), MovieLens[3]+IMDb[4]/Rotten Tomatoes[5](ML-S), a larger MovieLens data set (ML-L) and dblp[6]. The first four are from the tag recommendation application [Cantador *et al.*, 2011; Harper and Konstan, 2016]. Following previous practice [Jäschke *et al.*, 2008], users, items and tags that occur at least 5 times are used. The last is for link prediction. We follow [Acar *et al.*, 2009] to extract only the proceedings from 1991 to 2007.

To learn the factor matrices of CP decomposition, for the tag data sets, we use the Bayesian Probabilistic Tensor Factorization (BPTF) method and optimize the pair-wise ranking function of Bayesian Personalized Ranking (BPR) [Rendle *et al.*, 2009b; 2009a]. And for the dblp data set, Alternating Least Squares (ALS), provided in the Tensor Toolbox

[1]http://www.delicious.com
[2]http://www.lastfm.com
[3]http://www.grouplens.org
[4]http://www.imdb.com
[5]http://www.rottentomatoes.com
[6]http://dblp.uni-trier.de/db/

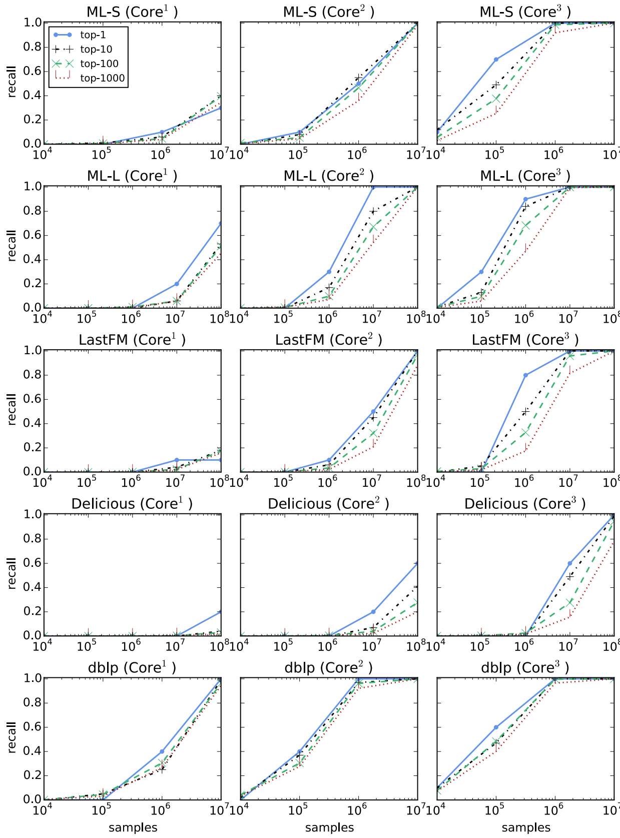


Figure 3: Recall of Core^k for $k \in \{1, 2, 3\}$.

of [Bader *et al.*, 2012; Bader and Kolda, 2007], is used. R is chosen following previous works, i.e. 64 for the first four data sets and 50 for the last one. We give the summary statistics in Table 1.

6.1 Accuracy and Running Time

We experiment with k equals 1,2 and 3 respectively. In Figure 3, we plot recall, i.e. the percentage of true top-t entries identified, versus the number of samples s. The corresponding running time are illustrated in Figure 4. The time for exhaustive computing is shown as the baseline. Here we set $t' = s$ and focus on the effectiveness of the sampling stage firstly. For each s, we run 10 times and the average result is reported.

We can see that with the increase of k, much higher recall can be obtained with the same number of samples. For the tag recommendation data sets, starting from $s = 10^5$, the time consumed by Core^2 becomes similar with Core^1. For dblp, they cost almost the same time from $s = 10^6$. Core^3 consumes much more time than the other two when the number of samples are small due to its relatively heavy initialization. However, when the number of samples reaches 10^7, its running time become quite similar with the other two. All three sampling methods are orders of magnitude faster than exhaustive computing.

To show more details on running time, we separate it into four stages: initialization, sampling, scoring and filtering. Initialization consists of computation of the weights for

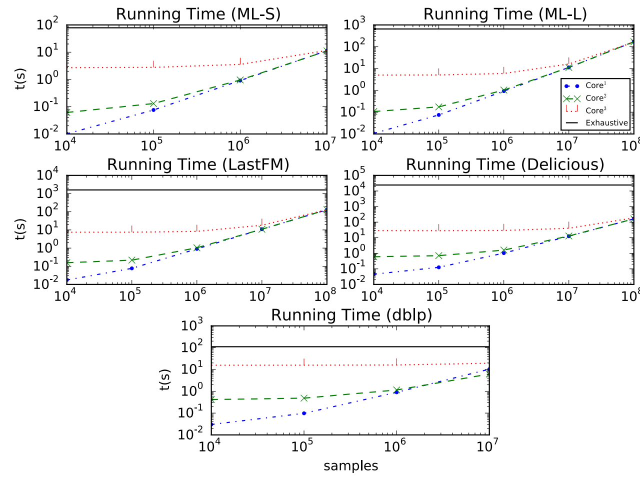
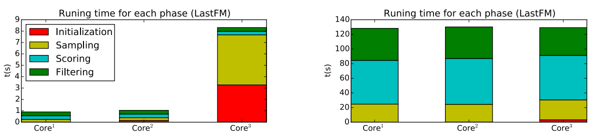


Figure 4: Running time on each data set.



(a) Sampling with $s = 1e6$ (b) Sampling with $s = 1e8$

Figure 5: Running time in details for LastFM.

core nodes and some preprocessing. Sampling corresponds to sampling index tuples and may also include the computation of the distributions for peripheral partitions when extension matrices are not pre-saved. The scoring phase shows the time consumed on computing and saving scores for each sampled index tuple. The last is to identify the top-t entries through filtering. We plot them for LastFM in Figure 5.

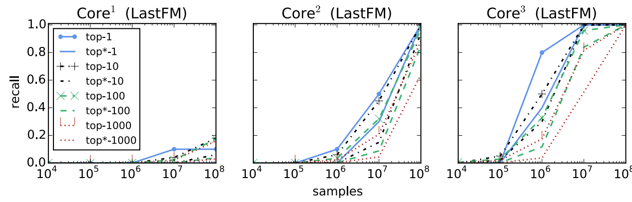
The running time for initialization is dependent on k and will be less influential with the increase of s. When s is relatively small (Figure 5 (a)), the sampling phase in Core^3 is dominated by its online computing of the probability vectors, which leads to considerable more running time when compared with Core^1 and Core^2. And when s is large (Figure 5 (b)), all sampling phases are dominated by the sampling procedure, and are therefore similar in run time. When $k = 3$, the probability for top entries are amplified, leading to a smaller size of Ψ_p and a saving of computation during the filtering phase. We notice that with the increase of s, the running time of Core^3 becomes quite similar with the other two.

6.2 Use Prefiltering

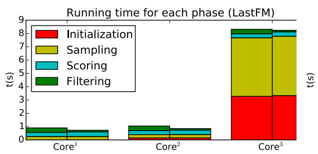
In this part, we evaluate the performance of using prefiltering during top-t extraction (Section 3.2). Due to space limit, we only show results on LastFM with $t' = s/10$ in Figure 6. We notice that prefiltering leads to a slightly lower recall but save considerable computing time.

6.3 Sampling for Collaborative Filtering

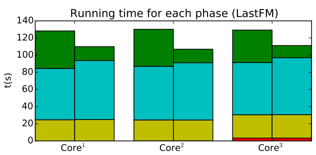
In Figure 7, we show the performance of Algorithm 2 applied for collaborative filtering, where the samples are shared among different users. Here Core^2 is used to find the top-100



(a) Recalls of the two strategies. top*-t and top-t are results with and without prefiltering respectively.



(b) $s = 1e6$.



(c) $s = 1e8$.

Figure 6: Performance of using prefiltering. The left bars in (b) and (c) are for $t' = s$ and the right are for $t' = s/10$.

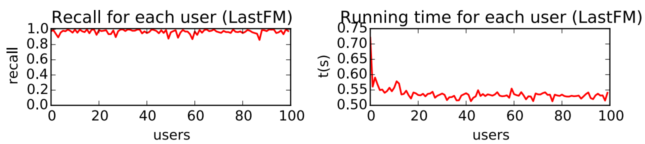


Figure 7: Recall and running time with multiple users on LastFM. For better legibility, we only plot results for the first 100 users.

largest entries on LastFM data set and s is set to 10^6. We can see that only for the first few users we need some time to get the samples and build the sample pools. After that, the computation time for a new user is much reduced.

6.4 Comparison with Diamond Sampling

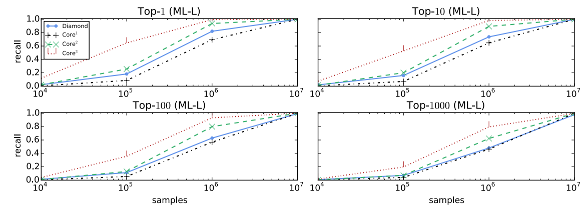
Since this is the first study of efficiently identifying the top entries in factorized tensors, there is no previous approach that we can compare with. On the other hand, when $N = 2$, the problem reduces to the MAD problem which finds the top entries in the product of two matrices. We therefore apply our algorithms to this problem and compare their performance with the diamond sampling method of [Ballard *et al.*, 2015].

For each user, we can multiply its characterizing vector into the item matrix to get a new user-dependent item matrix. Note that the searchings for different users are conducted independently, i.e. we run Algorithm 1 instead of Algorithm 2. In Figure 8, we show the results on ML-L. We set $t' = s/10$ and evaluate recall for $t \in \{1, 10, 100, 1000\}$. Since there are L_1 pairs of matrices, the average results are drawn.

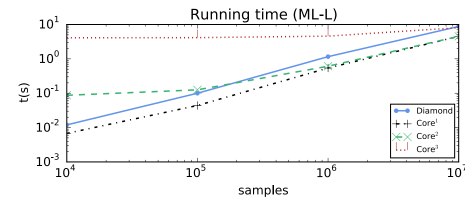
The performance of Core^1 is slightly worse than diamond sampling since the score $\hat{x}_{\boldsymbol{i}}$ in the latter is proportional to the square of entry magnitude. With the same number of samples, Core^2 can achieve higher recall than diamond sampling with less computing time. With Core^3, the recall can be further improved with the cost of some more time. This illustrates the advantage of augmenting the sampling probability to higher power of the entry magnitude.

6.5 Accuracy of the Scores

To show the accuracy of the estimation $\hat{x}_{\boldsymbol{i}}$, we run Core^3 on ML-S with $s = 10^8$ and plot the 10^4 pairs of $(x^3_{\boldsymbol{i}}, \|\boldsymbol{w}\|_1 \hat{x}_{\boldsymbol{i}}/s)$



(a) Recall on top-t



(b) Running time

Figure 8: Comparison with Diamond sampling on ML-L.

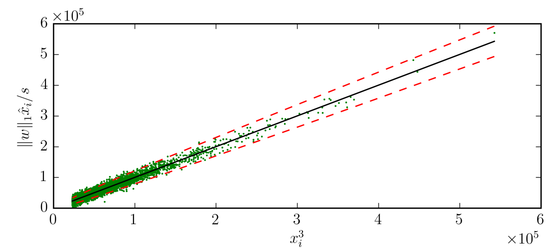


Figure 9: Plot of pairs $(\|\boldsymbol{w}\|_1 \hat{x}_{\boldsymbol{i}}/s, x^3_{\boldsymbol{i}})$ on ML-S. The solid line is the reference for equality. Two dashed lines are for $x^3_{\boldsymbol{i}} \pm \delta x^3_{\boldsymbol{i}}$.

which have the largest $x^3_{\boldsymbol{i}}$ in Ψ_p after sampling in Figure 9. Given $s = 10^8$, we set $\sigma = 0.1$ and compute the δ for each $x^k_{\boldsymbol{i}}$ using the equality in Eq.(6) from Theorem 1. Then two dashed lines for $x^k_{\boldsymbol{i}} \pm \delta x^k_{\boldsymbol{i}}$ can be plotted As expected, the points concentrate around the diagonal and most pairs fall within the two dashed lines, which confirmed with the theoretical result.

7 Conclusion

In this work, we proposed a novel sampling-based algorithm for approximate maximum search in tensor which is factorized by CP decomposition. It is an extension of MAD and this is the first study of this important problem. Experimental results indicate that our method is orders of magnitude faster than exhaustive search. When applied to the MAD problem, it is also more efficient than other state-of-the-art method.

References

[Acar *et al.*, 2009] Evrim Acar, Daniel M Dunlavy, and Tamara G. Kolda. Link prediction on evolving data using matrix and tensor factorizations. In *the IEEE International Conference on Data Mining Workshops*, pages 262–269, 2009.

[Adomavicius and Tuzhilin, 2015] Gediminas Adomavicius and Alexander Tuzhilin. *Context-aware recommender systems*, pages 191–226. Springer, 2015.

[Andoni and Indyk, 2008] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, pages 117–122, 2008.

[Bader and Kolda, 2007] Brett W. Bader and Tamara G. Kolda. Efficient matlab computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, pages 205–231, 2007.

[Bader *et al.*, 2012] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 2.5. *Available online*, 2012.

[Ballard *et al.*, 2015] Grey Ballard, Tamara G. Kolda, Ali Pinar, and C Seshadhri. Diamond sampling for approximate maximum all-pairs dot-product (MAD) search. In *the IEEE International Conference on Data Mining*, pages 11–20, 2015.

[Bertalmio *et al.*, 2000] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 417–424, 2000.

[Cantador *et al.*, 2011] Ivan Cantador, Peter Brusilovsky, and Tsvi Kuflik. Second workshop on information heterogeneity and fusion in recommender systems (HetRec 2011). In *RecSys*, pages 387–388, 2011.

[Coh, 1997] *Approximating Matrix Multiplication for Pattern Recognition Tasks*, 1997.

[Drineas *et al.*, 2006] Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast monte carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.

[Dunlavy *et al.*, 2011] Daniel M Dunlavy, Tamara G. Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data*, pages 10:1–10:27, 2011.

[Harper and Konstan, 2016] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, pages 19:1–19:19, 2016.

[Higham and Relton, 2016] Nicholas J Higham and Samuel D Relton. Estimating the largest elements of a matrix. *SIAM Journal on Scientific Computing*, pages C584–C601, 2016.

[Holodnak and Ipsen, 2015] John T Holodnak and Ilse CF Ipsen. Randomized approximation of the gram matrix: Exact computation and probabilistic bounds. *SIAM Journal on Matrix Analysis and Applications*, pages 110–137, 2015.

[Hu *et al.*, 2015] Yang Hu, Xi Yi, and Larry S Davis. Collaborative fashion recommendation: A functional tensor factorization approach. In *Proceedings of the 23rd ACM International Conference on Multimedia*, pages 129–138, 2015.

[Jäschke *et al.*, 2008] Robert Jäschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in social bookmarking systems. *AI Communications*, pages 231–247, 2008.

[Ke *et al.*, 2004] Yan Ke, Rahul Sukthankar, and Larry Huston. Efficient near-duplicate detection and sub-image retrieval. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, pages 869–876, 2004.

[Kolda and Bader, 2009] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, pages 455–500, 2009.

[Koren *et al.*, 2009] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[Rendle *et al.*, 2009a] Steffen Rendle, Leandro Balby Marinho, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data mining*, pages 727–736. ACM, 2009.

[Rendle *et al.*, 2009b] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 452–461, 2009.

[Salton and Harman, 2003] Gerard Salton and Donna Harman. *Information retrieval*. John Wiley and Sons Ltd., 2003.

[Shrivastava and Li, 2014] Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014.

[Xiong *et al.*, 2010] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the SIAM International Conference on Data Mining*, pages 211–222, 2010.