Efficiently Inferring Top-k Elephant Flows based on Discrete Tensor Completion

Kun Xie[1], Jiazheng Tian[1*], Xin Wang[2], Gaogang Xie[3,4], Jigang Wen[3], DafangZhang[1*]
[1] College of Computer Science and Electronics Engineering, Hunan University, Changsha, China
[2] Department of Electrical and Computer Engineering, State University of New York at Stony Brook, USA
[3] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
[4] The School of Computer and Control Engineering, University of Chinese Academy of Sciences
*Corresponding author

***Abstract*—Finding top-k elephant flows is a critical task in network measurement, with applications such as congestion control, anomaly detection, and traffic engineering. Traditional top-k flow detection problem focuses on using a small amount of memory to measure the total number of packets or bytes of each flow. Instead, we study a challenging problem of inferring the top-k elephant flows in a practical system with incomplete measurement data as a result of sub-sampling for scalability or data missing. The recent study shows it is promising to more accurately interpolate the missing data with a 3-D tensor compared to that based on a 2-D matrix. Taking full advantage of the multilinear structures, we apply tensor completion to first recover the missing data and then find the top-k elephant flows. To reduce the computational overhead, we propose a novel discrete tensor completion model which uses binary codes to represent the factor matrices. Based on the model, we further propose three novel techniques to speed up the whole top-k flow inference process: a discrete optimization algorithm to train the binary factor matrices, bit operations to facilitate quick missing data inference, and simplifying the finding of top-k elephant flows with binary code partition. In our discrete tensor completion model, only one bit is needed to represent the entry in the factor matrices instead of a real value (32 bits) needed in traditional tensor completion model, thus the storage cost is reduced significantly. Extensive experiments using two real traces demonstrate that compared with the state of art tensor completion algorithms, our discrete tensor completion algorithm can achieve similar data inference accuracy using significantly smaller time and storage space.**

***Index Terms*—Top-k elephant flow inference, Tensor completion**

I. Introduction

Finding the largest k flows, also referred to as the top-k elephant flows, is a fundamental network management function. Elephant flows contribute to a large portion of network traffic. Example management applications that can benefit from the efficient identification of top-k elephant flows include congestion control to dynamically schedule elephant flows [1], network capacity planning [2], anomaly detection [3], and caching of forwarding table entries [4]. The flow of interest may be defined by a source, a destination, a source-destination pair, or by the function specified by TCP, WWW, and P2P. In this paper, we explain our scheme using the flow based on source-destination pair as an example. Our scheme, however, is not constrained by this flow definition.

Literature studies [5]–[11] on the detection of top-k elephant flows attempt to use a small amount of memory to measure each flow's cardinality such as the total number of packets or bytes. Taking a sketch-based algorithm as an example, it relies on a sketch (e.g., CM sketch [5]) to measure the sizes of flows when each packet comes, while using a min-heap to keep track of the top-k flows.

Instead, this paper aims to infer top-k elephant flows that have the largest traffic volumes among data collected by the network controller (or SDN controller). If the traffic volumes of all source-destination flows have been collected in each time slot, the problem is trivial and can be solved by simply sorting the measurement data. However, in a practical system, the traffic measurement data are often incomplete thus sparse for several reasons: 1) Due to the high network monitoring and communication cost, it is impractical to collect full traffic volume information from a very large number of network nodes. Sample-based traffic monitoring is often applied where measurements are only taken between some random node pairs or in some of the periods for a given node pair; 2) Measurement data may get lost under severe communication and system conditions, including network congestion, node misbehavior, monitor failure, a transmission of measurement information through an unreliable transport protocol. *The inference of top-k elephant flows is made extremely difficult and challenging as a result of the incompleteness of traffic measurement data.*

With missing data, one possible approach to achieve the goal is to first recover the missing data, then return the top-k largest flows after sorting the recovered data. Various studies have been made to handle and recover the missing traffic data. Designed based on purely spatial or purely temporal information, the data recovery performance of most known approaches [12]–[14] is low. Recently matrix-completion-based algorithms are proposed to recover the missing traffic data by exploiting both spatial and temporal information [15]–[17]. Although the performance is good when the data missing ratio is low, the performance suffers when the missing ratio is large.

For more accurate missing data inference, a few recent studies [18], [19] try to model the traffic data as a 3-way tensor, and then fill in the missing data of traffic matrices through

tensor completion. Tensors are the higher-order generalization of vectors and matrices. Tensor models can take full advantage of the multilinear structures to achieve higher information precision for better data understanding. Compared with matrix-based data recovery, the tensor-based approach can better handle the missing traffic data and will be used in this paper.

With the capability of more accurate inference of missing measurement data, tensor completion can facilitate the finding of top-k elephant flows using partial measurements data. Top-k elephant flows may be found though three steps: 1) training the factor matrices through tensor factorization using the partial measurement samples, 2) recovering un-observed data using the trained factor matrices, and 3) sorting the complete traffic data descendingly to identify top-k flows. Although this is a promising way, given an $I \times J \times K$ traffic tensor with its tensor rank equal to R, the time complexity of data inference and sorting is $O(IJKR+IJKlog(k))$, which would be a big bottleneck that prevents efficient finding of top-k flows when the size of the traffic monitoring tensor is large.

Recent studies show that binary coding is a promising approach for fast similarity search [20], [21]. Specifically, based on fast bit operations, Hamming distance computation is extremely efficient. If we can represent factor matrices in tensor factorization by binary codes, it may only need simple bit operations to quickly find top-k elephant flows. However, no prior studies on tensor completion have been performed based on binary codes, addressing the challenge may open a new avenue for the tensor-based advanced data processing.

To take advantage of efficient bit operations, we propose a novel discrete tensor completion model for quickly inferring the top-k flows with partial measurement data. Our design includes several novel techniques:

- **Discrete tensor completion model**. To infer the missing measurement data, we propose a novel discrete tensor completion model in which real-valued factor matrices in traditional tensor completion model are represented by binary codes. Only using 1-bit binary value to represent an entry in each factor matrix, the space is largely reduced compared with traditional tensor completion model which uses 32 bits real value number to represent each entry of the factor matrices.
- **Discrete optimization algorithm**. For discrete tensor completion, to learn the binary codes, it needs to solve an optimization problem involving discrete values. This is generally NP-hard due to the discrete constraints. To tackle the problem, we develop a discrete optimization algorithm with the iterative solving of sub-problems of mixed-integer programming.
- **Efficiently inferring missing data through bit operations** Based on the analysis of relationship between CANDECOMP/PARAFAC (CP) decomposition and frontal slices, we transform the missing data recovery problem in each time slot to a problem of dot production of two binary vectors, which is further transformed into the calculation of Hamming distance through lightweight XOR bit operations.
- **Speed-up of top-k inference with binary code partition** To reduce the computation cost in missing data recovery as well as the search space for top-k sorting, we propose to further partition a binary vector into a set of small sub-vectors with the length of each set to that of one machine word of today's CPU. If two binary vectors are not similar, we can quickly determine that corresponding entry is definitely not the top-k entry. With the partition, we only need to compare a small number of sub-vectors rather than long binary vectors in the factor matrices.
- **Extensive experiments** We have conducted extensive experiments using two network measurement traces. Compared with the state of art tensor completion algorithms, our discrete tensor completion algorithm can achieve similar data inference accuracy using significantly smaller time and storage space.

The rest of the paper is organized as follows. We introduce the related work in Section II. The preliminaries of tensor are presented in Section III. We introduce the problem and the tensor completion model in Section IV and Section V, respectively. In Sections VI, VII, and VIII, we present in details our algorithms for discrete tensor completion, quick missing data inference, and binary code partition, respectively. Finally, we evaluate the performance of the proposed algorithm through extensive experiments in Section IX, and conclude the work in Section X.

II. Related work

Network measurements [22], [23] provide critical input for a wide range of network management applications. Finding top-k elephant flows is a critical task in network traffic measurement. It can be applied for congestion control, anomaly detection and traffic engineering. A number of algorithms have been proposed in recent years, including CM sketch [5], Frequent [6], Lossy Counting [7], Space-Saving [8], HeavyKeeper [9], HeavyGuardian [11], and CSS [10]. These algorithms generally focus on measuring the sizes of flows and finding top-k flows more accurately with lower space consumption. Rather than directly measuring flows passing by a node, we consider the inference of top-k elephant flows in the practical environment where measurement data can be incomplete and sparse. This makes the problem much harder.

To address the challenge, we propose a scheme based on tensor completion to infer the top-k elephant flows. Compared with compressive sensing and matrix completion, tensor completion can more accurately recover the missing data taking advantage of the multidimensional data structure. Several tensor completion algorithms [24]–[27] are proposed for recovering the missing data by capturing the global structure of the data via a high-order decomposition such as CANDECOMP/PARAFAC (CP) decomposition and Tucker decomposition. Some recent studies [18], [19] have modeled the traffic matrices of different time slots/days as a tensor to recover the missing data through tensor completion.

Although it is promising to take tensor completion algorithm in the presence of missing measurement data, it would involve a high computation cost to first recover the data and then find the top-k elephant flows. We propose a novel discrete tensor completion model which uses binary codes to represent the factor matrices. With this model, we propose several novel

techniques, including a discrete optimization algorithm to train the binary factor matrices, an algorithm to quickly infer the missing measurement data with bit operations, and a binary code partition algorithm to speed up the top-k inference.

III. Preliminaries

In this paper, scalars are denoted by lowercase letters (a, b, $\cdots$), vectors are written in boldface lowercase ($\mathbf{a}$, $\mathbf{b}$, $\cdots$), and matrices are represented with boldface capitals ($\mathbf{A}$, $\mathbf{B}$, $\cdots$). Higher-order tensors are written as calligraphic letters ($\mathcal{X}$, $\mathcal{Y}$, $\cdots$). The elements of a tensor are denoted by its symbolic name with indexes in subscript. For example, the i-th entry of a vector $\mathbf{a}$ is denoted by a_i, the element (i,j) of a matrix $\mathbf{A}$ is denoted by a_{ij}, and the element (i,j,k) of a third-order tensor $\mathcal{X}$ is denoted by x_{ijk}. The row (column) vectors of a matrix are denoted by the symbolic name of the matrix with indexes in subscript. For example, the i-th row (column) vector of a matrix $\mathbf{A}$ is denoted by $\mathbf{a}_i$ ($\mathbf{a}_{(i)}$). Besides, we denote $\|\bullet\|_F$ as the Frobenius norm of a matrix, and $\operatorname{sgn}(\bullet): R \to \{\pm 1\}$ as the round-off function.

Definition 1. *A* tensor *is a multidimensional array, and is a higher-order generalization of a vector (first-order tensor) and a matrix (second-order tensor). An N-way or Nth-order tensor (denoted as $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$) is an element of the tensor product of N vector spaces, where N is the order of $\mathcal{A}$, also called way or mode.*

The element of $\mathcal{A}$ is denoted by $a_{i_1,i_2,\cdots,i_N}$, $i_n \in \{1,2,\cdots,I_n\}$ with $1 \le n \le N$.

Definition 2. *Slices of a 3-way tensor are two-dimensional sub-arrays, defined by fixing all indexes but two.*

In Fig.1, a 3-way tensor $\mathcal{X}$ has horizontal, lateral and frontal slices, which are denoted by $\mathbf{X}_{i::}$, $\mathbf{X}_{:j:}$ and $\mathbf{X}_{::k}$, respectively. In this paper, we denote the frontal slice $\mathbf{X}_{::k}$ as $\mathbf{X}_k$.

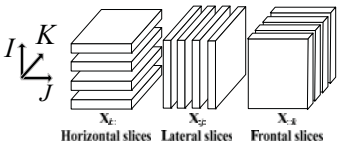

Fig. 1. Tensor slices

Definition 3. *The outer product of two column vectors $\mathbf{a} \circ \mathbf{b}$ is the matrix defined by: $(\mathbf{a} \circ \mathbf{b})_{ij} = a_i b_j$.*

Definition 4. *The outer product $\mathcal{A} \circ \mathcal{B}$ of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N_1}}$ and a tensor $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_{N_2}}$ is the tensor of the order $N_1 + N_2$ defined by*

$$(\mathcal{A} \circ \mathcal{B})_{i_1,i_2,\cdots,i_{N_1},j_1,j_2,\cdots,j_{N_2}} = a_{i_1,i_2,\cdots,i_{N_1}} b_{j_1,j_2,\cdots,j_{N_2}} \quad (1)$$

for all values of the indexes.

Since vectors are first-order tensors, the outer product of three column vectors $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ is a tensor given by:

$$(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})_{ijk} = a_i b_j c_k \quad (2)$$

for all values of the indexes.

Definition 5. *A 3-way tensor $\mathcal{X}$ is a rank one tensor if it can be written as the outer product of three column vectors, i.e. $\mathcal{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$.*

Definition 6. *The rank of a 3-way tensor is the minimal number of rank one tensors, that generate the tensor as their sum, i.e. the smallest R, such that $\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_{(r)} \circ \mathbf{b}_{(r)} \circ \mathbf{c}_{(r)}$ where $\mathbf{a}_{(r)}$, $\mathbf{b}_{(r)}$, and $\mathbf{c}_{(r)}$ are column vectors.*

Definition 7. *The idea of CANDECOMP/PARAFAC (CP) decomposition is to express a tensor as the sum of a finite number of rank one tensors. A 3-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ can be expressed as*

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_{(r)} \circ \mathbf{b}_{(r)} \circ \mathbf{c}_{(r)}, \quad (3)$$

with an entry calculated as

$$x_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr} \quad (4)$$

where $R > 0$, a_{ir}, b_{jr}, c_{kr} are the i-th, j-th, and k-th entry of column vectors $\mathbf{a}_{(r)} \in \mathbb{R}^I$, $\mathbf{b}_{(r)} \in \mathbb{R}^J$, and $\mathbf{c}_{(r)} \in \mathbb{R}^K$, respectively.

By collecting the vectors in the rank one components, we have tensor $\mathcal{X}$'s factor matrices $\mathbf{A} = [\mathbf{a}_{(1)}, \cdots, \mathbf{a}_{(R)}] \in \mathbb{R}^{I \times R}$, $\mathbf{B} = [\mathbf{b}_{(1)}, \cdots, \mathbf{b}_{(R)}] \in \mathbb{R}^{J \times R}$, and $\mathbf{C} = [\mathbf{c}_{(1)}, \cdots, \mathbf{c}_{(R)}] \in \mathbb{R}^{K \times R}$. Using the factor matrices, we can rewrite the CP decomposition as follows.

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_{(r)} \circ \mathbf{b}_{(r)} \circ \mathbf{c}_{(r)} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \quad (5)$$

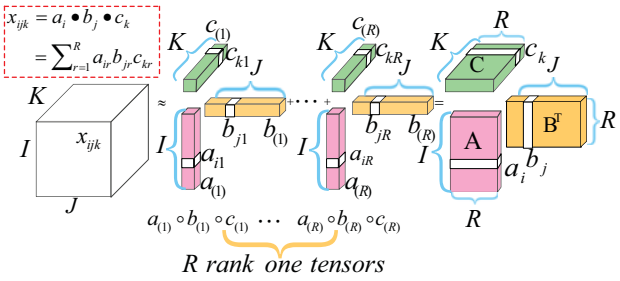

Fig. 2. CP decomposition of three-way tensor.

In Fig.2, a three-way tensor can be represented through CP decomposition as the sum of R outer products (rank one tensors). That is, $\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_{(r)} \circ \mathbf{b}_{(r)} \circ \mathbf{c}_{(r)}$, with column vectors $\mathbf{a}_{(r)} \in \mathbb{R}^I$, $\mathbf{b}_{(r)} \in \mathbb{R}^J$, and $\mathbf{c}_{(r)} \in \mathbb{R}^K$. In addition, an entry x_{ijk} can be calculated as the sum of the dot product of row vectors $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$. That is, $x_{ijk} = \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k$, where $\mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$ is the dot product of the three row vectors $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$ with $\mathbf{a}_i \in \mathbb{R}^{1 \times R}$, $\mathbf{b}_j \in \mathbb{R}^{1 \times R}$, and $\mathbf{c}_k \in \mathbb{R}^{1 \times R}$. In this paper, we design traffic data recovery algorithm based on the CP decomposition.

IV. System Model and Problem

We model the traffic measurement data as a 3-way tensor $\mathcal{M} \in \mathbb{R}^{I \times J \times K}$ (Fig.3), where I, J, and K correspond to the number of origin nodes, the number of destination nodes, and the number of time intervals monitored, respectively. Each entry m_{ijk} indicates the end-to-end traffic volume data from the origin node i to the destination node j in the time slot k.

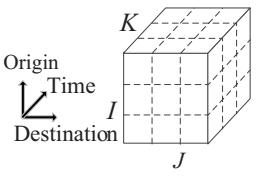

Fig. 3. 3-way traffic tensor.

As partial node pairs are often monitored to reduce the measurement load and also there are an unavoidable data losses upon severe communication conditions, $\mathcal{M}$ is generally a sparse and incomplete tensor. If there are no traffic data between a pair of nodes in a given time interval, it leaves the corresponding entry in $\mathcal{M}$ empty. All observed entries are denoted by $\Omega = \{(i,j,k) \,|\, m_{ijk} \text{ is known}\}$.

The aim of this paper is to efficiently find the top-k elephant flows in each time interval in the face of partial measurement data. The sparseness of data makes the problem much harder to solve.

V. Tensor completion model

Tensor completion is a promising technique to infer the missing data with a multi-dimensional data structure. Its versatility and effectiveness have been proven in a variety of fields, including analysis of web graphs, knowledge bases, chemometrics, signal processing, and computer vision.

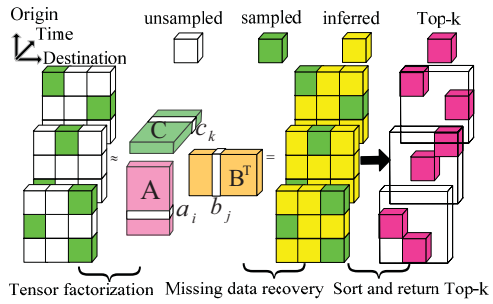

Fig. 4. Solution overview based on tensor completion.

As shown in Fig.4, based on tensor completion, a straightforward way of finding top-k elephant flows includes three steps: 1) training the factor matrices using the partial measurement samples based on tensor factorization, 2) inferring missing tensor entries using the trained factor matrices, and 3) sorting the data entries in each time interval to return the top-k elephant flows in each time slot using the recovered data. As mentioned in the introduction, this method could incur a high computational overhead.

In this section, we first present the real-valued tensor completion model to infer top-k elephant flows. After analyzing the complexity of this method, we propose a novel discrete tensor completion model.

A. Real-valued tensor completion model

Completing the tensor $\mathcal{M}$ based on its partial measurement samples can be performed through the factorization based on the popular CP decomposition method as follows:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \sum_{i,j,k\in\Omega} (m_{ijk} - \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k)^2 + \alpha||\mathbf{A}||^2 + \beta||\mathbf{B}||^2 + \gamma||\mathbf{C}||^2 \tag{6}$$

where $\mathbf{A} \in \mathbb{R}^{I\times R}$, $\mathbf{B} \in \mathbb{R}^{J\times R}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$ are factor matrices with $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$ being the i-th row vector of $\mathbf{A}$, the j-th row vector of $\mathbf{B}$, and the k-th row vector of $\mathbf{C}$. In Eq.(6), α, β, and γ are the regularization coefficients. $\alpha||\mathbf{A}||^2+\beta||\mathbf{B}||^2+\gamma||\mathbf{C}||^2$ is added in the formulation to prevent an over-fitting problem, where the errors corresponding to the tensor elements with observed entries are very small while the errors for the inferred data entries are large.

After getting three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, the original traffic tensor can be recovered by

$$\hat{\mathcal{M}} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \tag{7}$$

with the recovered entry

$$\hat{m}_{ijk} = \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k. \tag{8}$$

As the values of the factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are real, we call Eq. (6) a real-valued tensor completion problem. After obtaining the complete traffic tensor, top-k elephant flows can be found with the descending sort of data in each time interval.

Although this straightforward way can be applied to find the top-k elephant flows in the presence of partial measurements data, the time complexity is $O(IJKR + IJKlog(k))$, where $O(IJKR)$ corresponds to the cost of Eq.(7) and $O(IJKlog(k))$ corresponds to the cost of sorting and selecting top-k elephant flows in each time slot. This will become a computation bottleneck when the size of the network to monitor thus the traffic tensor is large.

B. Discrete tensor completion model

In Eq.(8), the entry m_{ijk} depends on the dot product of $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$, which obviously corresponds to data of origin i, destination j, and time slot k, respectively. In this paper, we call them factor vectors of origin i, destination j, and time slot k.

Recent studies show that binary coding is a promising approach for fast similarity search [20], [21]. Based on fast bit operations, Hamming distance computation is extremely efficient. If we can represent the factor vectors of origins, destinations, and time slots by binary codes, there is a possibility of speeding up the missing data recovery process in Eq.(8) with simple bit operations. This will in turn help expedite the finding of top-k flows. We will introduce additional procedures to further increase the speed of top-k finding in Section VII. Although promising, there are no prior studies on tensor completion based on the construction of binary codes for factor matrices.

We would like to represent the factor matrices using binary codes: $\mathbf{A} = [\mathbf{a}_1; \mathbf{a}_2; \cdots; \mathbf{a}_I]^T \in \{\pm 1\}^{I\times R}$, $\mathbf{B} = [\mathbf{b}_1; \mathbf{b}_2; \cdots; \mathbf{b}_J]^T \in \{\pm 1\}^{J\times R}$, $\mathbf{C} = [\mathbf{c}_1; \mathbf{c}_2; \cdots; \mathbf{c}_K]^T \in \{\pm 1\}^{K\times R}$. In this representation, each row vector of the factor matrix is a length R binary vector. To utilize the partial measurement samples to train the binary factor matrices, similar to Eq.(6), we define a novel discrete tensor completion model, which is formulated as

$$\begin{aligned} &\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \sum_{i,j,k\in\Omega} (\mathrm{m}_{ijk} - \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k)^2 \\ &s.t.\ \ \mathbf{A} \in \{\pm 1\}^{I\times R},\ \mathbf{B} \in \{\pm 1\}^{J\times R},\ \mathbf{C} \in \{\pm 1\}^{K\times R} \end{aligned} \tag{9}$$

Different from Eq.(6), as $\mathbf{A} = [\mathbf{a}_1; \mathbf{a}_2; \cdots; \mathbf{a}_I]^T \in \{\pm 1\}^{I\times R}$, $\mathbf{B} = [\mathbf{b}_1; \mathbf{b}_2; \cdots; \mathbf{b}_J]^T \in \{\pm 1\}^{J\times R}$, $\mathbf{C} = [\mathbf{c}_1; \mathbf{c}_2; \cdots; \mathbf{c}_K]^T \in \{\pm 1\}^{K\times R}$, the regularization $\alpha||\mathbf{A}||^2 + \beta||\mathbf{B}||^2 + \gamma||\mathbf{C}||^2 = \alpha \times I \times R + \beta \times J \times R + \gamma \times K \times R$ is a constant and hence is discarded from Eq.(9).

For any three binary row vectors $\mathbf{a}_i \in \{\pm 1\}^{1\times R}$, $\mathbf{b}_j \in \{\pm 1\}^{1\times R}$, and $\mathbf{c}_k \in \{\pm 1\}^{1\times R}$, the dot product of these vectors $\mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k$ falls into the range of $[-R, R]$. Before we use the observed measurement entries to train the binary factor matrices following Eq.(9), we normalize the traffic data into the value range $[-R, R]$:

$$m_{ijk} = \frac{m_{ijk}}{\frac{1}{R}\left(\max\limits_{1\le i\le I, 1\le j\le J, 1\le k\le K} (m_{ijk})\right)} \tag{10}$$

where $\max\limits_{1\le i\le I, 1\le j\le J, 1\le k\le K} (m_{ijk})$ is the maximum value of all the traffic data.

The operation in Eq.(10) does not impact the correlation hidden in the traffic data or the value order of the traffic data. In our performance evaluation, we will study the impact of R, the dimension of the row vector, on the accuracy and speed of top-k finding.

VI. Discrete tensor completion

Solving the discrete tensor completion problem in Eq.(9) is challenging, since it is generally NP-hard that involves $O(2^{(I+J+K)R})$ combinatorial search for the binary codes. To solve the problem involving binary discrete variables, people usually resort to a two-stage procedure: first solving a relaxed optimization algorithm without the discrete constraints, and then performing direct binary quantization. However, such an oversimplified two-stage procedure results in large quantization loss.

Instead, we directly solve the discrete optimization problem by alternatingly solving three subproblems: to obtain $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. In particular, we show that $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ can be efficiently updated through the discrete optimization.

A-subproblem: In this subproblem, we update $\mathbf{A}$ with $\mathbf{B}$ and $\mathbf{C}$ fixed. We can rewrite the function Eq. (9):

$$\begin{aligned}&\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \sum_{i,j,k\in\Omega} (m_{ijk}^2 - 2m_{ijk}(\mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k) + (\mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k)^2)\\ &s.t.\ \ \mathbf{A} \in \{\pm 1\}^{I\times R},\ \mathbf{B} \in \{\pm 1\}^{J\times R},\ \mathbf{C} \in \{\pm 1\}^{K\times R}\end{aligned} \tag{11}$$

Since the objective function in Eq. (11) is based on summing over $\mathbf{a}_i$, we can update $\mathbf{A}$ by updating each $\mathbf{a}_i$ according to

$$\min_{\mathbf{a}_i\in\{\pm 1\}^R} \sum_{i,j,k\in\Omega} (-2m_{ijk}(\mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k) + (\mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k)^2) \tag{12}$$

The update of different row vectors are independent and can be executed in parallel. Due to the binary constraints, the above minimization in Eq.(12) is generally NP-hard, we propose to update binary code $\mathbf{a}_i$ bit by bit. Denoting a_{ih} as the h-th bit of $\mathbf{a}_i$ and $a_{i\bar{h}}$ as the rest codes excluding a_{ih}, we update a_{ih} while fixing $a_{i\bar{h}}$.

Without loss of generality, let

$$\mathbf{a}_i = [a_{i\bar{h}}\ a_{ih}], \mathbf{b}_j = [b_{j\bar{h}}\ b_{jh}], \mathbf{c}_k = [c_{k\bar{h}}\ c_{kh}] \tag{13}$$

Then Eq.(12) can be rewritten as:

$$\begin{aligned}&\min_{\mathbf{a}_i\in\{\pm 1\}^R} \sum_{i,j,k\in\Omega} (a_{ih}b_{jh}c_{kh})(a_{i\bar{h}} \bullet b_{j\bar{h}} \bullet c_{k\bar{h}}) - 2a_{ih} \sum_{i,j,k\in\Omega} m_{ijk}b_{jh}c_{kh}\\ &= \min_{\mathbf{a}_i\in\{\pm 1\}^R} -2a_{ih} \sum_{i,j,k\in\Omega} (b_{jh}c_{kh}(m_{ijk} - a_{i\bar{h}} \bullet b_{j\bar{h}} \bullet c_{k\bar{h}}))\end{aligned} \tag{14}$$

Let

$$\hat{a}_{ih} = \sum_{i,j,k\in\Omega} (b_{jh}c_{kh})(m_{ijk} - a_{i\bar{h}} \bullet b_{j\bar{h}} \bullet c_{k\bar{h}}) \tag{15}$$

then Eq.(14) is equal to

$$\min_{\mathbf{a}_i\in\{\pm 1\}^R} -2a_{ih}\hat{a}_{ih} \tag{16}$$

It's easy to see, the expressions in $\hat{a}_{ih}$ are all known as we fix $\mathbf{B}$ and $\mathbf{C}$ (thus b_{jh} and c_{kh}) and $a_{i\bar{h}}$ when we update a_{ih}, so $\hat{a}_{ih}$ is also known. From $\min_{\mathbf{a}_i\in\{\pm 1\}^R} -2a_{ih}\hat{a}_{ih}$, the value of a_{ih} is related to the symbol of $\hat{a}_{ih}$. So, we just need to consider symbol of $\hat{a}_{ih}$.

Finally, the update rules of a_{ih} is:

$$a_{ih} \leftarrow \mathrm{sgn}(K(\hat{a}_{ih}, a_{ih})) \tag{17}$$

where $K(x,y) = \begin{cases} x & if\ x \neq 0 \\ y & otherwise \end{cases}$ and $\mathrm{sgn}(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \end{cases}$.

In Eq.(17), when $\hat{a}_{ih} \neq 0$, we update a_{ih} with $a_{ih} = sgn(\hat{a}_{ih})$; otherwise, we don't update it. **B-subproblem**, **C-subproblem** can be solved similar to **A-subproblem**. Through alternating optimization, the complete discrete tensor completion algorithm can be shown in Algorithm 1.

Algorithm 1 Discrete tensor completion

Input: $\{m_{ijk}\,|(i,j,k)\in\Omega\}$: observed entries
R: code length
Output: $\mathbf{A}\in\{\pm1\}^{I\times R}$, $\mathbf{B}\in\{\pm1\}^{J\times R}$, $\mathbf{C}\in\{\pm1\}^{K\times R}$

Initialization: $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$
while not converge **do**
  for $i = 1$ to I **do**
    while not converge **do**
      for $h = 1$ to R **do**
        $\hat{a}_{ih} = \sum_{i,j,k\in\Omega} (b_{jh}c_{kh})(m_{ijk} - a_{i\bar{h}} \bullet b_{j\bar{h}} \bullet c_{k\bar{h}})$
        $a_{ih} \leftarrow \mathrm{sgn}(K(\hat{a}_{ih}, a_{ih}))$
      end for
    end while
  end for
  for $j = 1$ to J **do**
    while not converge **do**
      for $h = 1$ to R **do**
        $\hat{b}_{jh} = \sum_{i,j,k\in\Omega} (a_{ih}c_{kh})(m_{ijk} - a_{i\bar{h}} \bullet b_{j\bar{h}} \bullet c_{k\bar{h}})$
        $b_{jh} \leftarrow \mathrm{sgn}(K(\hat{b}_{jh}, b_{jh}))$
      end for
    end while
  end for
  for $k = 1$ to K **do**
    while not converge **do**
      for $h = 1$ to R **do**
        $\hat{c}_{kh} = \sum_{i,j,k\in\Omega} (a_{ih}b_{jh})(m_{ijk} - a_{i\bar{h}} \bullet b_{j\bar{h}} \bullet c_{k\bar{h}})$
        $c_{kh} \leftarrow \mathrm{sgn}(K(\hat{c}_{kh}, c_{kh}))$
      end for
    end while
  end for
end while
Return $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.

Denote the objective function $(\sum_{i,j,k\in\Omega} (\mathrm{m}_{ijk} - \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k)^2)$ of problem Eq.(9) as $L(\mathbf{A},\mathbf{B},\mathbf{C})$, the convergence of the proposed Algorithm 1 is guaranteed by the following theorem.

Theorem 1. *The sequence $\{\mathbf{A}^{(t)},\mathbf{B}^{(t)},\mathbf{C}^{(t)}\}$ generated by Algorithm 1 monotonically decreases the objective function $L(\mathbf{A},\mathbf{B},\mathbf{C})$ of Eq.(9) where t denotes the iteration step; the objective function sequence $\{L(\mathbf{A}^{(t)},\mathbf{B}^{(t)},\mathbf{C}^{(t)})\}$ converges; the sequence $\{\mathbf{A}^{(t)},\mathbf{B}^{(t)},\mathbf{C}^{(t)}\}$ converges.*

Due to limited space, the proof is abbreviated.

VII. Quick inference of missing data based on Hamming distance

After we obtain the factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ by applying Algorithm 1, the traffic tensor can be recovered by $\hat{\mathcal{M}} = [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$ with $\hat{m}_{ijk} = \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k$ as shown in Eq.(7) and Eq.(8). Using the recovered traffic tensor, after sorting the data entries in each time interval descendingly , the top-k elephant flows can be returned. Although it is promising, as discussed

in Section V-A, the computation cost is large.

In our tensor model, each frontal slice records the volumes of traffic data among all origins and destinations in a time slot. In this section, we first investigate the relationship between tensor CP decomposition and the decomposition of a frontal slice of the tensor, then present our algorithm of quickly inferring top-k elephant flows based on simple bit operations to get the Hamming distance.

A. Relationship between CP decomposition and frontal slice decomposition

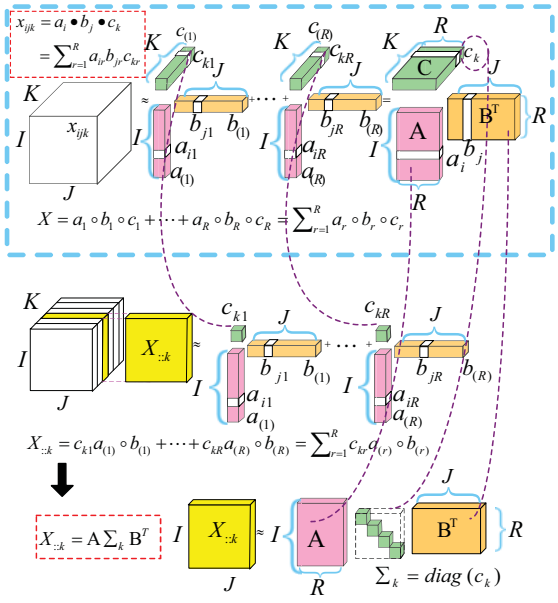

Fig. 5. The relationship between tensor CP decomposition and frontal slice matrix decomposition.

In Fig.5, according to Eq.(5), the CP decomposition of a 3-way tensor $\mathcal{X}$ can be written as follows.

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_{(r)} \circ \mathbf{b}_{(r)} \circ \mathbf{c}_{(r)} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \tag{18}$$

where matrices $\mathbf{A} \in \{\pm 1\}^{I \times R}$, $\mathbf{B} \in \{\pm 1\}^{J \times R}$, $\mathbf{C} \in \{\pm 1\}^{K \times R}$ are the binary factor matrices in the CP decomposition. According to the CP decomposition, in Fig.5, a frontal slice $\mathbf{X}_k$ can be written as

$$\mathbf{X}_k = c_{k1}\mathbf{a}_{(1)} \circ \mathbf{b}_{(1)} + \cdots + c_{kR}\mathbf{a}_{(R)} \circ \mathbf{b}_{(R)} = \sum_{i=1}^{R} c_{ki}\mathbf{a}_{(i)} \circ \mathbf{b}_{(i)} \tag{19}$$

where $c_{k1}, c_{k2}, \cdots, c_{kR}$ are the entries of the k-th row of the factor matrix $\mathbf{C}$.

Eq. (19) shows that each frontal slice $\mathbf{X}_k$ can be expressed as a superposition of R rank-1 matrices $\mathbf{a}_{(i)} \circ \mathbf{b}_{(i)}$ $(1 \le i \le R)$. That is, the traffic data $\mathbf{X}_k$ is represented by the linear combination of R rank-1 matrices $\mathbf{a}_{(i)} \circ \mathbf{b}_{(i)}$. As $\mathbf{A} = \left[\mathbf{a}_{(1)}, \cdots, \mathbf{a}_{(R)}\right] \in \{\pm 1\}^{I \times R}$, $\mathbf{B} = \left[\mathbf{b}_{(1)}, \cdots, \mathbf{b}_{(R)}\right] \in \{\pm 1\}^{J \times R}$, according to (19), $\mathbf{X}_k$ can be rewritten as

$$\mathbf{X}_k = \mathbf{A}\mathbf{\Sigma}_k\mathbf{B}^T \tag{20}$$

where $\mathbf{A} \in \{\pm 1\}^{I \times R}$ and $\mathbf{B} \in \{\pm 1\}^{J \times R}$ are the binary factor matrices in the CP decomposition, $\mathbf{\Sigma}_k = diag\left(\mathbf{C}_{k:}\right)$, and $\mathbf{C}_{k:}$ is the k-th row of the binary factor matrix $\mathbf{C}$.

B. Inferring top-k elephant flows through hamming distance

Let $\mathbf{D} = \mathbf{A}\mathbf{\Sigma}_k$, we have $\mathbf{X}_k = \mathbf{D}\mathbf{B}^T$. As $\mathbf{A} \in \{\pm 1\}^{I \times R}$ and $\mathbf{C} \in \{\pm 1\}^{K \times R}$, we have $\mathbf{D} \in \{\pm 1\}^{I \times R}$.

To recover the traffic data entry m_{ijk}, the straightforward strategy is $m_{ijk} = \mathbf{d}_i \bullet \mathbf{b}_j$, which still results in a high computation cost. Now we will show $\mathbf{d}_i \bullet \mathbf{b}_j$ can be derived using the Hamming distance between $\mathbf{d}_i$ and $\mathbf{b}_j$ at a much smaller computation cost. Let

$$sim(i,j) = \frac{1}{R}\sum_{t=1}^{R} II(d_{it} = b_{jt}) \tag{21}$$

where $II()$ is the indicator function that returns 1 if the statement is true and 0 otherwise. We can easily verify that $sim(i,j) = 0$ if all bits of $\mathbf{d}_i$ and $\mathbf{b}_j$ are different and $sim(i,j) = 1$ if $\mathbf{d}_i = \mathbf{b}_j$. $\sum_{t=1}^{R} II(d_{it} = b_{jt})$ represents the total number of bit locations that have the same bit value in vectors $\mathbf{d}_i$ and $\mathbf{b}_j$, $\sum_{t=1}^{R} II(d_{it} \ne b_{jt})$ is the number of bit locations that have different bit values. Therefore, we have

$$sim(i,j) = \frac{1}{2R}\left(\sum_{t=1}^{R} II(d_{it} = b_{jt}) + R - \sum_{t=1}^{R} II(d_{it} \ne b_{jt})\right) \tag{22}$$

As $\mathbf{d}_i \in \{\pm 1\}^{1 \times R}$ and $\mathbf{b}_j \in \{\pm 1\}^{1 \times R}$, when two bit locations are different, we have $II(d_{it} \ne b_{jt}) = 1$ and $d_{it}b_{jt} = -1$. When two bit locations are the same, we have $II(d_{it} = b_{jt}) = d_{it}b_{jt} = 1$. We further have

$$sim(i,j) = \frac{1}{2R}\left(R + \sum_{t=1}^{R} d_{it}b_{jt}\right) = \frac{1}{2} + \frac{1}{2R}\mathbf{d}_i \bullet \mathbf{b}_j \tag{23}$$

Eq.(23) indicates that when $\mathbf{d}_i$ and $\mathbf{b}_j$ are binary vectors, the order of $\mathbf{d}_i \bullet \mathbf{b}_j$ is the same as the order of $sim(i,j)$, that is, if $\mathbf{d}_i \bullet \mathbf{b}_j > \mathbf{d}_{i'} \bullet \mathbf{b}_{j'}$, we have $sim(i,j) > sim(i',j')$. Hamming distance between two codes $\mathbf{d}_i$ and $\mathbf{b}_j$ is simply the number of bits that are different, which can be computed at extreme efficiency using bit-wise XOR operations. Obviously, $sim(i,j) = R - \text{hamming\_distance}(\mathbf{d}_i, \mathbf{b}_j)$.

VIII. Top-k inferring speedup through binary code partition

To infer the top-k elephant flows, the Hamming distances of $\mathbf{d}_i$ and $\mathbf{b}_j$ for all $1 \le i \le I$ and $1 \le j \le J$ should first be computed, then after sorting $I \times J$ entries, top-k largest entries will be returned. The computation cost is still high for large I and J.

To solve our problem, we only need to know values of large entries rather than all entry values. To speed up the top-k inferring procedure, we can divide each R-bit code $\mathbf{d}_i$ and $\mathbf{b}_j$ into p sub-codes, denoted by $\mathbf{d}_i = \left\{\mathbf{d}_i^1, \mathbf{d}_i^2, \cdots, \mathbf{d}_i^p\right\}$ and $\mathbf{b}_j = \left\{\mathbf{b}_j^1, \mathbf{b}_j^2, \cdots, \mathbf{b}_j^p\right\}$, respectively. We then test the Hamming distance of each sub-codes one by one. If the total Hamming distance of the beginning set of sub-code pairs exceeds a threshold ε, obviously, these two $\mathbf{d}_i$ and $\mathbf{b}_j$ will not be similar and should not be counted as the candidate top-k largest entry. We can stop comparing the remaining bit strings and discard this entry without need of further sub-code comparison and the sorting of this entry. For more efficient calculation, we set the length of the sub-code to be the length of the machine code as in today's CPUs, the read

and XOR operation on one machine word can be done within one memory access.

This partition-based comparison not only helps speed up the phase of missing data inference but also cut short the sorting phase. Fig.6 shows an example, where we set $\varepsilon = 2$. We divide the binary code into four parts. After comparing the first two parts, we find the total Hamming distance reaches 3, which is larger than $\varepsilon = 2$. Therefore, these two binary codes are not similar and the corresponding entry should not be considered as the candidate top-k entries.

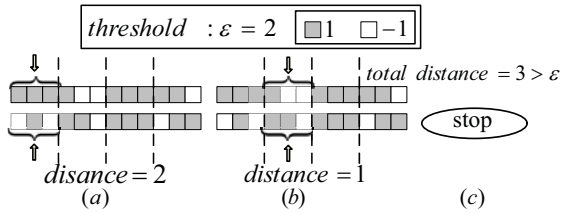

Fig. 6. Binary code partition.

There is a tradeoff to set ε. Small ε will result in that more entries are discarded while increase the risk of missing the top-k entries, large ε will increase the computation cost while decrease the risk of missing the top-k entries. In the experiment, we will set appropriate ε according to the parameter k and R.

IX. Performance Evaluation

Although we use network traffic monitoring as an example to illustrate our scheme (denoted as discrete CP (DCP)), our scheme is a general and can also be applied to infer top-k Origin-Destination (OD) pairs with other network performance data. Therefore, we use two real network monitoring traces (WS-DREAM [28] and Harvard226 [29]) to evaluate our proposed DCP. WS-DREAM records the traffic volume between 142 users and 4,500 Web services over 64 consecutive time slices, at an interval of 15 minutes. Harvard226 contains measurement data of application-level RTTs, with timestamps, between 226 Azureus clients collected in 72 hours. Two performance metrics are utilized to evaluate the proposed DCP, which are defined as follows.

Definition 8. Denote by $y \in \{1, \cdots, h\}$ a vector of entries of the monitoring data in a time slot and π a permutation of the vector. π_i denotes the position of i-th element in the permutation. π_s sorts y in decreasing order. Let positive integer k be a truncation threshold. The Discounted Cumulative Gains ($DCG@k$) score and its normalized variant ($NDCG@k$) are defined respectively as $DCG@k(y,\pi) = \sum_{i=1}^{k} \frac{2^{y_{\pi_i}}-1}{\log(i+2)}$ and $NDCG@k(y,\pi) = \frac{DCG@k(y,\pi)}{DCG@k(y,\pi_s)}$.

$DCG@k$ is maximized when $\pi = \pi_s$. In this paper, the truncation threshold k reflects how many elephant flows are inferred. NDCG is a normalized version of DCG so that the score is bounded by [0,1].

Definition 9. Denote by $y \in \{1, \cdots, h\}$ a vector of entries and let positive integer k be a truncation threshold. $\Phi(k)$ represents the set of entries inferred to be top-k and $\Psi(k)$ represents the set of top-k entries of y. The Precision is defined as $Precision@k = \frac{|\Psi(k)| \cap |\Phi(k)|}{|\Psi(k)|}$.

We empirically set $k = 10$. As we want to infer the top-k elephant flows in each time slot, the resulted $NDCG@10$ and $Precision@10$ are the average $NDCG@10$ and $Precision@10$ of all time slots monitored.

A. Impact of dimension R

According to the Section V-B, we use Eq.(10) to normalize the monitoring data. The dimension R of the binary vector (i.e., the row vector of factor matrices) directly impacts the normalized value range and thus the resolution of the recovered data, which further impacts the top-k inference accuracy. In Fig.7, with the increase of R, NDCG@10 and Precision@10 increase quickly when R is small value. After R reaches 128 in WS-DREAM and 160 in Harvard226, our DCP converges and achieves the stable accuracy performance. Therefore, we set $R = 128$ for WS-DREAM and $R = 160$ for Harvard226 respectively in our rest experiments.

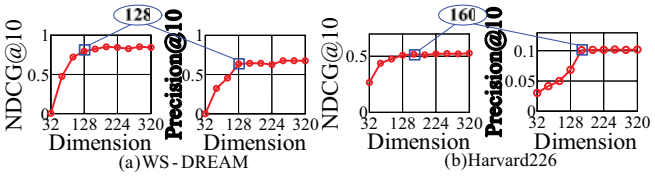

Fig. 7. Impact of dimension R.

B. Impact of threshold ε

In Section VIII, we propose a binary code partition algorithm to speedup the top-k inferring procedure, in which the long binary code is partitioned into several sub-codes. Sub-code pairs are compared one by one until the total Hamming distance of the beginning set of sub-code pairs exceeds a threshold ε, then we are sure the corresponding entry is not top-k and stop the comparison further. Given $k = 10$, we investigate the inference accuracy under different ε. In Fig.8, we find that with the increase of ε, as more data entries are utilized to act as the candidate entries to calculate the top-k flows, the inference accuracy increases initially. After ε reaches a value point, the inference accuracy becomes stable and will not change with the further increase of ε. On the other hand, with the increase of ε, more candidate entries are involved to infer the top-k flows, which results in the increase of computation time. Therefore, we set ε to be the value point, that is $\varepsilon = 49$ for WS-DREAM and $\varepsilon = 65$ for Harvard226, respectively.

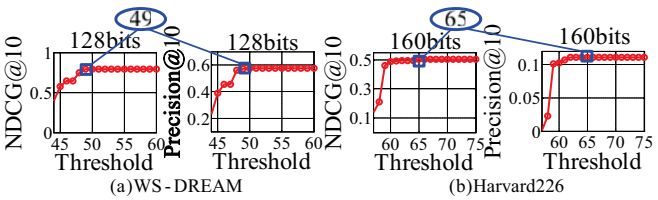

Fig. 8. Threshold ε study.

C. Performance comparison

Besides DCP, we implement other three schemes based on tensor completion to infer top-k elephant flows. The first is

CP in which the tensor completion is achieved though CP decomposition using real valued factor matrices. We use CP as a baseline to show the performance gap between real values and binary codes. CP is the scheme following Section V. The second is DCP-No-Partition which adopts all algorithms in DCP except binary code partition in Section VIII. The third is DCP-Dot in which for the missing data recovery, it directly calculates the missing data through dot operations instead of XOR bit operations adopted in our DCP.

1) Computation time

All the four schemes includes three steps. Step 1: train the factor matrices using the observed entries, Step 2: recover the missing data, Step 3: sort the complete data and return the top-k elephant flows. We denote the computation time of each step as Train time, Recovery time, and Sorting time. The computation time is measured as average number of seconds taken to complete corresponding operations. All experiments are run on a Microstar workstation, which is equipped with two Intel (R) Xeon (R) E5-2620 CPUs with 2GHz processor, 24 Cores and 32 GB RAM. We insert a timer to all schemes implemented .

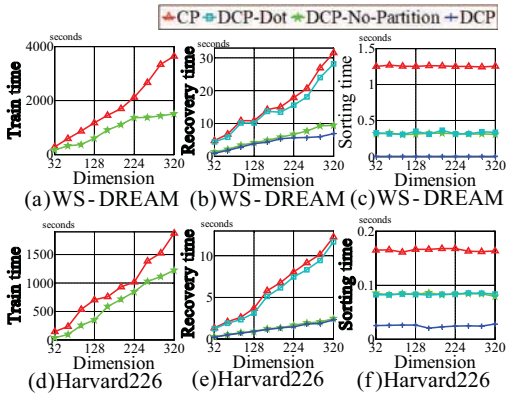

Fig. 9. Computation time.

In Fig.9(a) and Fig.9(d), as DCP, DCP-No-Partition, DCP-Dot adopt same Algorithm 1 to train factor matrices, the training time under these three schemes are same. As a result, we only draw CP and DCP in the figure. Benefited by Algorithm 1, our discrete tensor completion algorithm has good convergence behavior which makes the training time much smaller than that under CP which uses real valued tensor completion.

Fig.9(b) and Fig.9(e) show the recovery time. Among all schemes implemented, our DCP achieves the lowest recovery time. As CP uses real valued factor matrix, for each entry x_{ijk}, recovering it needs the dot product upon real valued row vectors $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$ with $x_{ijk} = \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k$. Such operations are costly, thus the recovery time under CP is the largest one. Our DCP, DCP-No-Partition, and DCP-Dot adopt binary based factor matrices. Recovering entries using DCP and DCP-No-Partition is based on the calculation of Hamming distance through XOR bit operation, thus the recovery time under DCP, DCP-No-Partition are much smaller than that under DCP-Dot, as DCP-Dot adopts dot operation on the binary code. These results demonstrate that missing data inference based on Hamming distance operation is very efficient. The speed gain obtained by binary code partition is different under different traces. In WS-DREAM, the speed gain is larger than that under Harvard226. This is because the threshold ε in WS-DREAM is smaller than that in Harvard226 according to Fig.8, thus the speed gains are different for different traces.

Fig.9(c) and Fig.9(f) show sorting time. The sorting time is independent from the dimension R. Therefore, the curves in Fig.9(c) and Fig.9(f) are parallel to the x-axis. Obviously, the sorting time under CP is the largest one, then the DCP-No-Partition and DCP-Dot, and finally the DCP. Sorting algorithm under CP operates on the real valued entries, while our DCP, DCP-No-Partition, and DCP-Dot operate on integers within the range [-R, R]. Therefore, the sorting time under DCP, DCP-No-Partition, and DCP-Dot is much smaller than that under CP. Moreover, as binary code partition can not only speed up missing data recovery but also reduce the sorting space in step 3, the sorting time under DCP is only half that under DCP-No-Partition and DCP-Dot.

2) Inference accuracy

Fig.10 shows inference accuracy of all the implemented schemes. For fair comparison, the dimension of all tensor completion algorithms are set to 128 for WS-DREAM and 160 for Harvard226. Obviously, the inference accuracy that our DCP, DCP-No-Partition, and DCP-Dot can achieve is similar to that of CP. In Fig.11, we will show that our DCP achieves the accuracy similar to that of CP, but uses significantly smaller storage.

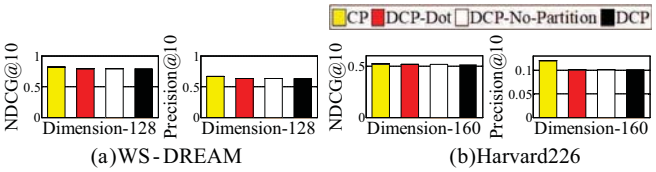

Fig. 10. Inference accuracy.

3) Storage

For the top-k inference with tensor completion, we first utilize the observed data to train the factor matrices, then sort the recovered missing data plus the observed data to find the top-k flows. In the whole process, the factor matrices, the observed sample data, and the recovered data should be stored. We show the storage cost of each part for all schemes in Fig.11.

For the real valued tensor completion, at least 32 bits are needed to store a real number. As only one bit is needed in our scheme to represent one entry in the factor matrices, the storage cost is 1/32 that required in current real valued tensor completion, as shown in Fig.11(a) and Fig.11(d). For all the schemes implemented, we use the same observed sample data to train the factor matrices, therefore, in Fig.11(b) and Fig.11(e), the storage of the sampling data under different schemes are the same.

Under CP, the recovered data are also real numbers, which use 32 bits for each entry. Under DCP, DCP-Dot, and DCP-No-Partition, the recovered data are integers falling into the

range $[-R, R]$ with $R = 128$ and $R = 160$ for traces WS-DREAM and Harvard226, respectively. We can use a short integer which occupies 16 bits to store the recovered entry. Therefore, the recovery data storage under DCP-Dot and DCP-No-Partition is only the half of that under CP. Compared with DCP-Dot and DCP-No-Partition, the recovery storage under our DCP is much lower because our binary partition algorithm in DCP largely reduces the number of entries needed to be recovered.

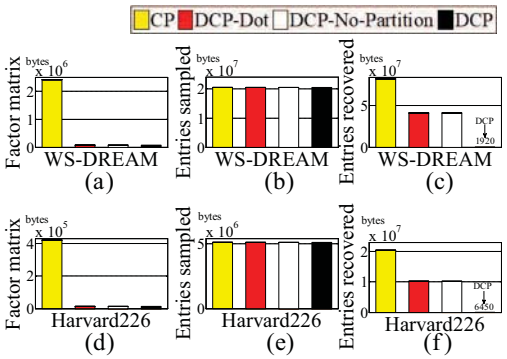

Fig. 11. Storage.

X. Conclusion

We propose a novel discrete tensor completion model to facilitate inferring the top-k elephant flows in the practical environment where measurement data are incomplete and sparse. Several novel techniques are proposed in the model, including a discrete optimization algorithm to train the binary factor matrices, significant cost reduction with bit operations for quick missing data inference, binary code partition to simplify the finding of top-k elephant flows. We have done extensive experiment based on two real measurement traces. The experiment results demonstrate that the computation cost and storage cost is significantly reduced compared with current real valued tensor completion model.

Acknowledgment

The work is supported by the National Natural Science Foundation of China under Grant Nos.61572184 and 61725206, Hunan Provincial Natural Science Foundation of China under Grant No.2017JJ1010, U.S. NSF ECCS 1731238 and NSF CNS 1526843, the open project funding (CASNDST201704) of CAS Key Lab of Network Data Science and Technology, and the open project funding (CARCH201809) of State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences.

References

[1] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, "Programmable packet scheduling at line rate," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 44–57.

[2] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational ip networks: Methodology and experience," in *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4. ACM, 2000, pp. 257–270.

[3] A. Lakhina, M. Crovella, and C. Diot, "Characterization of network-wide anomalies in traffic flows," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 201–206.

[4] O. Rottenstreich and J. Tapolcai, "Optimal rule caching and lossy compression for longest prefix matching," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 2, pp. 864–878, 2017.

[5] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[6] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *European Symposium on Algorithms*. Springer, 2002, pp. 348–360.

[7] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 346–357.

[8] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International Conference on Database Theory*. Springer, 2005, pp. 398–412.

[9] J. Gong, T. Yang, H. Zhang, H. Li, S. Uhlig, S. Chen, L. Uden, and X. Li, "Heavykeeper: An accurate algorithm for finding top-k elephant flows," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, 2018, pp. 909–921.

[10] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows." in *INFOCOM*, 2016, pp. 1–9.

[11] T. Yang, J. Gong, H. Zhang, L. Zou, L. Shi, and X. Li, "Heavyguardian: Separate and guard hot items in data streams," in *ACM SIGKDD*, 2018, pp. 2584–2593.

[12] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," in *ACM SIGMETRICS*, 2003.

[13] Y.Vardi, "Network tomography," *J. Amer. Statist. Assoc.*, vol. vol. 91, no. 433,, p. pp. 365–377, 1996.

[14] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," *ACM IMW*, 2002.

[15] M. Roughan, Y. Zhang, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices (extended version)," *Networking IEEE/ACM Transactions on*, vol. 20, no. 3, pp. 662 – 676, 2012.

[16] G. Gürsun and M. Crovella, "On traffic matrix completion in the internet," in *ACM IMC 2012*.

[17] Y.-C. Chen, L. Qiu, Y. Zhang, G. Xue, and Z. Hu, "Robust network compressive sensing," in *ACM MobiCom*, 2014.

[18] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, and G. Zhang, "Accurate recovery of internet traffic data: A tensor completion approach," in *IEEE INFOCOM*, 2016.

[19] K. Xie, C. Peng, X. Wang, G. Xie, and J. Wen, "Accurate recovery of internet traffic data under dynamic measurements," in *IEEE INFOCOM*, 2017.

[20] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 1–8.

[21] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, "Learning to hash for indexing big data: a survey," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 34–57, 2016.

[22] Q. Liu, H. Dai, A. X. Liu, Q. Li, X. Wang, and J. Zheng, "Cache assisted randomized sharing counters in network measurement," in *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 2018, p. 40.

[23] H. Dai, Y. Zhong, A. X. Liu, W. Wang, and M. Li, "Noisy bloom filters for multi-set membership testing," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 1. ACM, 2016, pp. 139–151.

[24] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 208–220, 2013.

[25] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, 2011.

[26] E. Acar and T. G. Dunlavy, Daniel M.and Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *Journal of Chemometrics*, vol. 25, no. 2, p. 67–86, 2011.

[27] S. Gandy, B. Recht, and I. Yamada, "Tensor completion and low-n-rank tensor recovery via convex optimization," *Inverse Problems*, vol. 27, no. 2, p. 025010, 2011.

[28] http://wsdream.github.io/dataset/wsdream_dataset2.html.

[29] J. Ledlie, P. Gardner, and M. I. Seltzer, "Network coordinates in the wild." in *USENIX NSDI 2007*.