Out-of-Core and Distributed Algorithms for Dense Subtensor Mining

Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos

Abstract—How can we detect fraudulent lockstep behavior in large-scale multi-aspect data (i.e., tensors)? Can we detect it when data are too large to fit in memory or even on a disk? Past studies have shown that dense subtensors in real-world tensors (e.g., social media, Wikipedia, TCP dumps, etc.) signal anomalous or fraudulent behavior such as retweet boosting, bot activities, and network attacks. Thus, various approaches, including tensor decomposition and search, have been proposed for detecting dense subtensors rapidly and accurately. However, existing methods have low accuracy, or they assume that tensors are small enough to fit in main memory, which is unrealistic in many real-world applications such as social media and web.

To overcome these limitations, we propose D-CUBE, a disk-based dense-subtensor detection method, which also can run in a distributed manner across multiple machines. Compared to state-of-the-art methods, D-CUBE is (1) **Memory Efficient**: requires up to *1,600× less memory* and handles *1,000× larger* data (*2.6TB*), (2) **Fast**: up to *7× faster* due to its near-linear scalability, (3) **Provably Accurate**: gives a guarantee on the densities of the detected subtensors, and (4) **Effective**: spotted network attacks from TCP dumps and synchronized behavior in rating data most accurately.

Index Terms—Tensor, Dense Subtensor, Anomaly Detection, Fraud Detection, Out-of-core Algorithm, Distributed Algorithm

1 INTRODUCTION

G IVEN a tensor that is too large to fit in memory, how can we detect dense subtensors? Especially, can we spot dense subtensors without sacrificing speed and accuracy provided by in-memory algorithms?

A common application of this problem is review fraud detection, where we aim to spot suspicious lockstep behavior among groups of fraudulent user accounts who review suspiciously similar sets of products. Previous work [1], [2], [3] has shown the benefit of incorporating extra information, such as timestamps, ratings, and review keywords, by modeling review data as a tensor. Tensors allow us to consider additional dimensions in order to identify suspicious behavior of interest more accurately and specifically. That is, extraordinarily dense subtensors indicate groups of users with lockstep behaviors both in the products they review and along the additional dimensions (e.g., multiple users reviewing the same products at the exact same time).

In addition to review-fraud detection, spotting dense subtensors has been found effective for many anomaly-detection tasks. Examples include network-intrusion detection in TCP dumps [1], [2], retweet-boosting detection in on-line social networks [3], bot-activity detection in Wikipedia [1], and genetics applications [2], [4].

Due to these wide applications, several methods have been proposed for rapid and accurate dense-subtensor detection, and search-based methods have shown the best performance. Specifically, search-based methods [1], [3] outperforms methods based on tensor decomposition, such as CP Decomposition and HOSVD [2], in terms of accuracy

- *K. Shin and C. Faloutsos are with Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 15213.*
  E-mail: {kijungs, christos}@cs.cmu.edu
- *B. Hooi and J. Kim are with Statistics Department, Carnegie Mellon University, Pittsburgh, PA, 15213.*
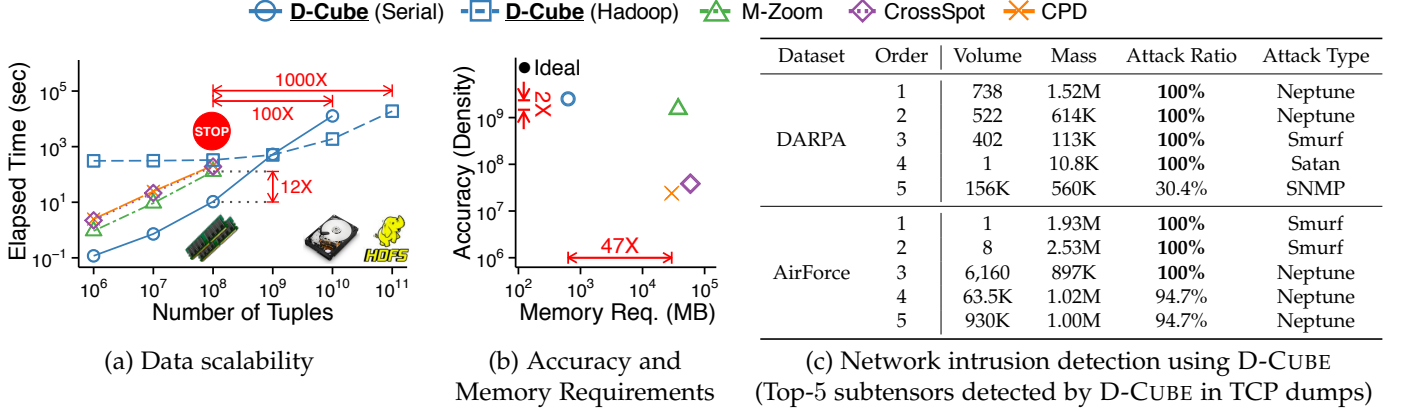 *E-mail: {bhooi, jisuk1}@andrew.cmu.edu*

TABLE 1: Comparison of D-CUBE and state-of-the-art dense-subtensor detection methods. ✓denotes 'supported'.

	M-ZOOM [1]	CROSSSPOT [3]	MAF [2]	FRAUDAR [5]	D-CUBE
High-order Tensors	✓	✓	✓		✓
Flexibility in Density Measures	✓	✓		✓	✓
Accuracy Guarantees	✓			✓	✓
Out-of-core Computation					✓
Distributed Computation					✓

and flexibility with regard to the choice of density metrics. Moreover, the latest search method [1] provides a guarantee on the densities of the subtensors it finds, while methods based on tensor decomposition do not.

However, existing search methods for dense-subtensor detection assume that input tensors are small enough to fit in memory. Moreover, they are not directly applicable to tensors stored in disk since using them for such tensors incurs too many disk I/Os due to their highly iterative nature. However, real applications, such as social media and web, often involve disk-resident tensors with terabytes or even petabytes, which in-memory algorithms cannot handle. This leaves a growing gap that needs to be filled.

To overcome these limitations, we propose D-CUBE[1], a dense-subtensor detection method for disk-resident tensors. D-CUBE works under the W-Stream model [7], where data are only sequentially read and written during computation. As seen in Table 1, only D-CUBE supports out-of-core computation, which allows it to process data too large to fit in main memory. D-CUBE is optimized for this setting by

1. The preliminary version of D-CUBE appeared in [6].

(a) Data scalability

(b) Accuracy and Memory Requirements

(c) Network intrusion detection using D-Cube (Top-5 subtensors detected by D-Cube in TCP dumps)

Dataset	Order	Volume	Mass	Attack Ratio	Attack Type
DARPA	1	738	1.52M	**100%**	Neptune
	2	522	614K	**100%**	Neptune
	3	402	113K	**100%**	Smurf
	4	1	10.8K	**100%**	Satan
	5	156K	560K	30.4%	SNMP
AirForce	1	1	1.93M	**100%**	Smurf
	2	8	2.53M	**100%**	Smurf
	3	6,160	897K	**100%**	Neptune
	4	63.5K	1.02M	94.7%	Neptune
	5	930K	1.00M	94.7%	Neptune

Fig. 1: **Strengths of D-Cube.** The red stop sign denotes 'out of memory'. (a) **Fast & Scalable**: D-Cube was *12× faster* and successfully handled *1,000× larger data (2.6TB)* than its best competitors. (b) **Efficient & Accurate**: D-Cube required *47× less memory* and found denser subtensors than its best competitors from English Wikipedia revision history. (c) **Effective**: D-Cube accurately spotted network attacks from TCP dumps. See Section 4 for the detailed experimental settings.

carefully minimizing the amount of disk I/O and the number of steps requiring disk accesses, without losing accuracy guarantees it provides. Moreover, we present a distributed version of D-Cube using the MapReduce framework [8], specifically its open source implementation Hadoop [9].

The main strengths of D-Cube are as follows:

- **Memory Efficient:** D-Cube requires up to *1,600×* less memory and successfully handles *1,000×* larger data (*2.6TB*) than its best competitors (Figures 1(a) and 1(b)).
- **Fast:** D-Cube detects dense subtensors up to *7×* faster in real-world tensors and *12 ×* faster in synthetic tensors than than its best competitors due to its near-linear scalability with all aspects of tensors (Figure 1(a)).
- **Provably Accurate:** D-Cube provides a guarantee on the densities of the subtensors it finds (Theorem 3), and it shows similar or higher accuracy than its best competitors on real-world tensors (Figure 1(b)).
- **Effective:** D-Cube successfully spotted network attacks from TCP dumps, and lockstep behavior in rating data, with the highest accuracy (Figure 1(c)).

Reproducibility: The code and data used in the paper are available at **http://www.cs.cmu.edu/~kijungs/codes/dcube**.

In Section 2, we provide notations and a formal problem definition. In Section 3, we propose D-Cube, a disk-based dense-subtensor detection method. In Section 4, we present experimental results. After discussing related work in Section 5, we offer conclusions in Section 6.

2 NOTATIONS AND DEFINITIONS

In this section, we first introduce notations and concepts used in the paper. Then, we define density measures and the problem of top-k dense-subtensor detection.

2.1 Notations and Concepts

Table 2 lists the symbols frequently used in the paper. We use $[x] = \{1, 2, \cdots, x\}$ for brevity. Let $\mathcal{R}(A_1, \cdots, A_N, X)$ be a relation with N dimension attributes, denoted by $A_1, \cdots, A_N$, and a nonnegative measure attribute, denoted by X (see Example 1 for a running example). For each tuple $t \in \mathcal{R}$ and for each $n \in [N]$, $t[A_n]$ and $t[X]$ indicate the values of A_n and X, resp., in t. For each $n \in [N]$, we use

TABLE 2: Table of symbols.

Symbol	Definition
$\mathcal{R}(A_1, \cdots, A_N, X)$	relation representing an N-way tensor
N	number of the dimension attributes in $\mathcal{R}$
A_n	n-th dimension attribute in $\mathcal{R}$
X	measure attribute in $\mathcal{R}$
$t[A_n]$ (or $t[X]$)	value of attribute A_n (or X) in tuple t in $\mathcal{R}$
$\mathcal{B}$	a subtensor in $\mathcal{R}$
$\rho(\mathcal{B}, \mathcal{R})$	density of subtensor $\mathcal{B}$ in $\mathcal{R}$
$\mathcal{R}_n$ (or $\mathcal{B}_n$)	set of distinct values of A_n in $\mathcal{R}$ (or $\mathcal{B}$)
$M_\mathcal{R}$ (or $M_\mathcal{B}$)	mass of $\mathcal{R}$ (or $\mathcal{B}$)
$\mathcal{B}(a, n)$	set of tuples with attribute $A_n = a$ in $\mathcal{B}$
$M_{\mathcal{B}(a,n)}$	attribute-value mass of a in A_n
k	number of subtensors we aim to find
θ	mass-threshold parameter in D-Cube
$[x]$	$\{1, 2, \cdots, x\}$

$\mathcal{R}_n = \{t[A_n] : t \in \mathcal{R}\}$ to denote the set of distinct values of A_n in $\mathcal{R}$. The relation $\mathcal{R}$ is naturally represented as an N-way tensor of size $|\mathcal{R}_1| \times \cdots \times |\mathcal{R}_N|$. The value of each entry in the tensor is $t[X]$, if the corresponding tuple t exists, and 0 otherwise. Let $\mathcal{B}_n$ be a subset of $\mathcal{R}_n$. Then, a *subtensor* $\mathcal{B}$ in $\mathcal{R}$ is defined as $\mathcal{B}(A_1, ..., A_N, X) = \{t \in \mathcal{R} : \forall n \in [N], t[A_n] \in \mathcal{B}_n\}$, the set of tuples where each attribute A_n has a value in $\mathcal{B}_n$. The relation $\mathcal{B}$ is a 'subtensor' because it forms a subtensor of size $|\mathcal{B}_1| \times \cdots \times |\mathcal{B}_N|$ in the tensor representation of $\mathcal{R}$, as in Figure 2(b). We define the mass of $\mathcal{R}$ as $M_\mathcal{R} = \sum_{t \in \mathcal{R}} t[X]$, the sum of attribute X in the tuples of $\mathcal{R}$. We denote the set of tuples of $\mathcal{B}$ whose attribute $A_n = a$ by $\mathcal{B}(a, n) = \{t \in \mathcal{B} : t[A_n] = a\}$ and its mass, called the *attribute-value mass of a in A_n*, by $M_{\mathcal{B}(a,n)} = \sum_{t \in \mathcal{B}(a,n)} t[X]$.

Example 1 (Wikipedia Revision History). *As in Figure 2, assume a relation $\mathcal{R}(user, page, date, count)$, where each tuple (u, p, d, c) in $\mathcal{R}$ indicates that user u revised page p, c times, on date d. The first three attributes, A_1=user, A_2=page, and A_3=date, are dimension attributes, and the other one, X=count, is the measure attribute. Let $\mathcal{B}_1$=\{Alice, Bob\}, $\mathcal{B}_2$=\{A, B\}, and $\mathcal{B}_3$=\{May-29\}. Then, $\mathcal{B}$ is the set of tuples regarding the revision of page A or B by Alice or Bob on May-29, and its mass $M_\mathcal{B}$ is 19, the total number of such revisions. The attribute-value mass of Alice (i.e., $M_{\mathcal{B}(Alice,1)}$) is 9, the number of revisions on A or B by exactly Alice on May-29. In the tensor representation,*

User	Page	Date	Count
Alice	A	May-29	4
Alice	B	May-29	5
Bob	A	May-29	7
Bob	B	May-29	3
Carol	C	May-30	1
⋮	⋮	⋮	⋮

(a) Relation $\mathcal{R}$

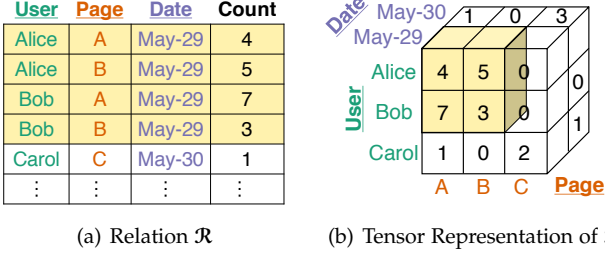(b) Tensor Representation of $\mathcal{R}$

Fig. 2: Pictorial description of Example 1. (a) Relation $\mathcal{R}$ where the colored tuples compose relation $\mathcal{B}$. (b) Tensor representation of $\mathcal{R}$ where the relation $\mathcal{B}$ forms a subtensor.

$\mathcal{B}$ composes a subtensor in $\mathcal{R}$, as depicted in Figure 2(b).

2.2 Density Measures

We present density measures proven useful for anomaly detection in past studies. We use them throughout the paper although our dense-subtensor detection method, explained in Section 3, is flexible and not restricted to specific measures. Below, we slightly abuse notations to emphasize that the density measures are the functions of $M_{\mathcal{B}}$, $\{|\mathcal{B}_n|\}_{n=1}^N$, $M_{\mathcal{R}}$, and $\{|\mathcal{R}_n|\}_{n=1}^N$, where $\mathcal{B}$ is a subtensor of a relation $\mathcal{R}$.

Arithmetic Average Mass (Definition 1) and Geometric Average Mass (Definition 2), which were used for detecting network intrusions and bot activities [1], are the extensions of density measures widely-used for graphs [10], [11].

Definition 1 (Arithmetic Average Mass ρ_{ari} [1]). *The arithmetic average mass of a subtensor $\mathcal{B}$ of a relation $\mathcal{R}$ is defined as*

$$\rho_{ari}(\mathcal{B}, \mathcal{R}) = \rho_{ari}(M_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n=1}^N, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N) = \frac{M_{\mathcal{B}}}{\frac{1}{N}\sum_{n=1}^N |\mathcal{B}_n|}.$$

Definition 2 (Geometric Average Mass ρ_{geo} [1]). *The geometric average mass of a subtensor $\mathcal{B}$ of a relation $\mathcal{R}$ is defined as*

$$\rho_{geo}(\mathcal{B}, \mathcal{R}) = \rho_{geo}(M_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n=1}^N, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N) = \frac{M_{\mathcal{B}}}{(\prod_{n=1}^N |\mathcal{B}_n|)^{\frac{1}{N}}}.$$

Suspiciousness (Definition 3), which was used for detecting 'retweet-boosting' activities [12], is the negative log-likelihood that $\mathcal{B}$ has mass $M_{\mathcal{B}}$ under the assumption that each entry of $\mathcal{R}$ is i.i.d from a Poisson distribution.

Definition 3 (Suspiciousness ρ_{susp} [3]). *The suspiciousness of a subtensor $\mathcal{B}$ of a relation $\mathcal{R}$ is defined as*

$$\rho_{susp}(\mathcal{B}, \mathcal{R}) = \rho_{susp}(M_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n=1}^N, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N)$$
$$= M_{\mathcal{B}}\left(\log \frac{M_{\mathcal{B}}}{M_{\mathcal{R}}} - 1\right) + M_{\mathcal{R}}\prod_{n=1}^N \frac{|\mathcal{B}_n|}{|\mathcal{R}_n|} - M_{\mathcal{B}}\log\left(\prod_{n=1}^N \frac{|\mathcal{B}_n|}{|\mathcal{R}_n|}\right).$$

Entry Surplus (Definition 4) is the observed mass of $\mathcal{B}$ subtracted by α times the expected mass, under the assumption that the value of each entry (in the tensor representation) in $\mathcal{R}$ is i.i.d. It is a multi-dimensional extension of edge surplus [13], a density metric for graphs.

Definition 4 (Entry Surplus [13]). *The entry surplus of a subtensor $\mathcal{B}$ of a relation $\mathcal{R}$ is defined as*

$$\rho_{es(\alpha)}(\mathcal{B}, \mathcal{R}) = \rho_{es(\alpha)}(M_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n=1}^N, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N)$$
$$= M_{\mathcal{B}} - \alpha M_{\mathcal{R}}\prod_{n=1}^N \frac{|\mathcal{B}_n|}{|\mathcal{R}_n|}.$$

Subtensors with high entry surplus are configurable by adjusting α. With high α values, relatively small compact subtensors have higher entry surplus than large sparse subtensors, while the opposite happens with small α values. We show this tendency experimentally in Section 4.7.

2.3 Problem Definition

Based on the concepts and density measures in the previous sections, we define the problem of top-k dense-subtensor detection in a large-scale tensor in Definition 5.

Definition 5 (Large-scale Top-k Densest Subtensor Detection). **(1) Given:** *a large-scale relation $\mathcal{R}$ not fitting in memory, the number of subtensors k, and a density measure ρ,* **(2) Find:** *the top-k subtensors of $\mathcal{R}$ with the highest density in terms of ρ.*

Even when we restrict our attention to finding one subtensor in a matrix fitting in memory (i.e., $k = 1$ and $N = 2$), obtaining an exact solution takes $O((\sum_{n=1}^N |\mathcal{R}_n|)^6)$ time [14], [15], which is infeasible for large-scale tensors. Thus, our focus in this work is to design an approximate algorithm with (1) near-linear scalability with all aspects of $\mathcal{R}$, which does not fit in memory, (2) an approximation guarantee at least for some density measures, and (3) meaningful results on real-world data.

3 PROPOSED METHOD

In this section, we propose D-CUBE, a disk-based dense-subtensor detection method. We first describe D-CUBE in Section 3.1. Then, we prove its theoretical properties in Section 3.2. Lastly, we present our MAPREDUCE implementation of D-CUBE in Section 3.3. Throughout these subsections, we assume that the entries of tensors (i.e., the tuples of relations) are stored on disk and read/written only in a sequential way. However, all other data (e.g., distinct attribute-value sets and the mass of each attribute value) are assumed to be stored in memory.

3.1 Algorithm

D-CUBE is a search method that starts with the given relation and removes attribute values (and the tuples with the attribute values) sequentially so that a dense subtensor is left. Contrary to previous approaches, D-CUBE removes multiple attribute values (and the tuples with the attribute values) at a time to reduce the number of iterations and also disk I/Os. In addition to this advantage, D-CUBE carefully chooses attribute values to remove to give the same accuracy guarantee as if attribute values were removed one by one, and shows similar or even higher accuracy empirically.

3.1.1 Overall Structure of D-Cube (Algorithm 1)

Algorithm 1 describes the overall structure of D-CUBE. It first copies and assigns the given relation $\mathcal{R}$ to $\mathcal{R}^{ori}$ (line 1); and computes the sets of distinct attribute values composing $\mathcal{R}$ (line 2). Then, it finds k dense subtensors one by one from $\mathcal{R}$ (line 6) using its mass as a parameter (line 5). The detailed procedure for detecting a single dense subtensor from $\mathcal{R}$ is explained in Section 3.1.2. After each subtensor $\mathcal{B}$ is found, the tuples included in $\mathcal{B}$ are removed from $\mathcal{R}$ (line 7) to prevent the same subtensor from being found

---

Algorithm 1: D-CUBE

Input : relation: $\mathcal{R}$, density measure: ρ,
threshold: $\theta(\geq 1)$,
the number of subtensors we aim to find: k
Output: k dense subtensors
1 $\mathcal{R}^{ori} \leftarrow copy(\mathcal{R})$
2 compute $\{\mathcal{R}_n\}_{n=1}^N$
3 $results \leftarrow \emptyset$ ▷ *list of dense subtensors*
4 **for** $i \leftarrow 1..k$ **do**
5    $M_{\mathcal{R}} \leftarrow \sum_{t\in\mathcal{R}} t[X]$
6    $\{\mathcal{B}_n\}_{n=1}^N \leftarrow find\_one(\mathcal{R}, \{\mathcal{R}_n\}_{n=1}^N, M_{\mathcal{R}}, \rho, \theta)$
         ▷ *see Algorithm 2*
7    $\mathcal{R} \leftarrow \{t \in \mathcal{R} : \exists n \in [N], t[A_n] \notin \mathcal{B}_n\}$ ▷ $\mathcal{R} \leftarrow \mathcal{R} - \mathcal{B}$
8    $\mathcal{B}^{ori} \leftarrow \{t \in \mathcal{R}^{ori} : \forall n \in [N], t[A_n] \in \mathcal{B}_n\}$
9    $results \leftarrow results \cup \{\mathcal{B}^{ori}\}$
10 **return** $results$

---

again. Due to this change in $\mathcal{R}$, subtensors found from $\mathcal{R}$ are not necessarily the subtensors of the original relation $\mathcal{R}^{ori}$. Thus, instead of $\mathcal{B}$, the subtensor in $\mathcal{R}^{ori}$ formed by the same attribute values forming $\mathcal{B}$ is added to the list of k dense subtensors (lines 8-9). Notice that, due to this step, D-CUBE can detect overlapping dense subtensors. That is, a tuple can be included in multiple dense subtensors.

Based on our assumption that the sets of distinct attribute values (i.e., $\{\mathcal{R}_n\}_{n=1}^N$ and $\{\mathcal{B}_n\}_{n=1}^N$) are stored in memory and can be randomly accessed, all the steps in Algorithm 1 can be performed by sequentially reading and writing tuples in relations (i.e., tensor entries) in disk without loading all the tuples in memory at once. For example, the filtering steps in lines 7-8 can be performed by sequentially reading each tuple from disk and writing the tuple to disk only if it satisfies the given condition.

Note that this overall structure of D-CUBE is similar with that of M-ZOOM [1] except the fact that the tuples are stored on disk. However, the methods differ significantly in the way each dense subtensor is found from $\mathcal{R}$, which is explained in the following section.

3.1.2 Single Subtensor Detection (Algorithm 2)

Algorithm 2 describes how D-CUBE detects each dense subtensor from the given relation $\mathcal{R}$. It first initializes a subtensor $\mathcal{B}$ to $\mathcal{R}$ (lines 1-2) then repeatedly removes attribute values and the tuples of $\mathcal{B}$ with those attribute values until all values are removed (line 5).

Specifically, in each iteration, D-CUBE first chooses a dimension attribute A_i that attribute values are removed from (line 7), Then, it computes D_i, the set of attribute values whose masses are less than $\theta(\geq 1)$ times the average (line 8). We explain how the dimension attribute is chosen, in Section 3.1.3 and analyze the effects of θ on the accuracy and the time complexity, in Section 3.2. The tuples whose attribute values of A_i are in D_i are removed from $\mathcal{B}$ at once within a single scan of $\mathcal{B}$ (line 16). However, deleting a subset of D_i may achieve higher value of the metric ρ. Hence, D-CUBE computes the changes in the density of $\mathcal{B}$ (line 11) as if the attribute values in D_i were removed one by one, in an increasing order of their masses. This allows D-CUBE to optimize ρ as if we removed attributes one by one, while still benefiting from the computational speedup of removing multiple attributes in each scan. Note

---

Algorithm 2: $find\_one$ in D-CUBE

Input : relation: $\mathcal{R}$, attribute-value sets: $\{\mathcal{R}_n\}_{n=1}^N$,
mass: $M_{\mathcal{R}}$, density measure: ρ, threshold: $\theta(\geq 1)$
Output: attribute values forming a dense subtensor
1 $\mathcal{B} \leftarrow copy(\mathcal{R})$, $M_{\mathcal{B}} \leftarrow M_{\mathcal{R}}$ ▷ *initialize the subtensor $\mathcal{B}$*
2 $\mathcal{B}_n \leftarrow copy(\mathcal{R}_n), \forall n \in [N]$
3 $\tilde{\rho} \leftarrow \rho(M_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n=1}^N, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N)$ ▷ $\tilde{\rho}$: *max ρ so far*
4 $r, \tilde{r} \leftarrow 1$ ▷ r: *current order of attribute values, $\tilde{r}$: r with $\tilde{\rho}$*
5 **while** $\exists n \in [N], \mathcal{B}_n \neq \emptyset$ **do**   ▷ *until all values are removed*
6   compute $\{\{M_{\mathcal{B}(a,n)}\}_{a\in\mathcal{B}_n}\}_{n=1}^N$
7    $i \leftarrow select\_dimension()$ ▷ *see Algorithms 3 and 4*
8    $D_i \leftarrow \{a \in \mathcal{B}_i : M_{\mathcal{B}(a,i)} \leq \theta\frac{M_{\mathcal{B}}}{|\mathcal{B}_i|}\}$ ▷ D_i: *set to be removed*
9   sort D_i in an increasing order of $M_{\mathcal{B}(a,i)}$
10   **for** each value $a \in D_i$ **do**
11      $\mathcal{B}_i \leftarrow \mathcal{B}_i - \{a\}$, $M_{\mathcal{B}} \leftarrow M_{\mathcal{B}} - M_{\mathcal{B}(a,i)}$
12      $\rho' \leftarrow \rho(M_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n=1}^N, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N)$
         ▷ ρ' : *ρ when a is removed*
13      $order(a,i) \leftarrow r, r \leftarrow r+1$
14   **if** $\rho' > \tilde{\rho}$ **then**
15        $\tilde{\rho} \leftarrow \rho', \tilde{r} \leftarrow r$ ▷ *update max ρ so far*
16    $\mathcal{B} \leftarrow \{t \in \mathcal{B} : t[A_i] \notin D_i\}$ ▷ *remove tuples*
17 $\tilde{\mathcal{B}}_n \leftarrow \{a \in \mathcal{R}_n : order(a,n) \geq \tilde{r}\}, \forall n \in [N]$ ▷ *reconstruct*
18 **return** $\{\tilde{\mathcal{B}}_n\}_{n=1}^N$

Algorithm 3: $select\_dimension$ by cardinality

Input : attribute-value sets: $\{\mathcal{B}_n\}_{n=1}^N$
Output: a dimension in $[N]$
1 **return** $n \in [N]$ with maximum $|\mathcal{B}_n|$

---

that these changes in ρ can be computed exactly without actually removing the tuples from $\mathcal{B}$ or even accessing the tuples in $\mathcal{B}$ since its mass (i.e., $M_{\mathcal{B}}$) and the number of distinct attribute values (i.e., $\{|\mathcal{B}_n|\}_{n=1}^N$) are maintained up-to-date (lines 11-12). This is because removing an attribute value from a dimension attribute does not affect the masses of the other values of the same attribute. The orders that attribute values are removed and when the density of $\mathcal{B}$ is maximized are maintained (lines 13-15) so that the subtensor $\mathcal{B}$ maximizing the density can be restored and returned (lines 17-18), as the result of Algorithm 2.

Note that, in each iteration (lines 5-16) of Algorithm 2, the tuples of $\mathcal{B}$, which are stored on disk, need to be scanned only twice, once in line 6 and once in line 16. Moreover, both steps can be performed by simply sequentially reading and/or writing tuples in $\mathcal{B}$ without loading all the tuples in memory at once. For example, to compute attribute-value masses in line 6, D-CUBE increases $M_{\mathcal{B}(t[A_n],n)}$ by $t[X]$ for each dimension attribute A_n after reading each tuple t in $\mathcal{B}$ sequentially from disk.

3.1.3 Dimension Selection (Algorithms 3 and 4)

We discuss two policies for choosing a dimension attribute that attribute values are removed from. They are used in line 7 of Algorithm 2 offering different advantages.

Maximum cardinality policy (Algorithm 3): The dimension attribute with the largest cardinality is chosen, as described in Algorithm 3. This simple policy, however, provides an accuracy guarantee (see Theorem 3 in Section 3.2.2).

Maximum density policy (Algorithm 4): The density of $\mathcal{B}$ when attribute values are removed from each dimension

Algorithm 4: *select_dimension* by density

Input : attribute-value sets: $\{\mathcal{B}_n\}_{n=1}^N$ and $\{\mathcal{R}_n\}_{n=1}^N$,
 attribute-value masses: $\{\{M_{\mathcal{B}(a,n)}\}_{a\in\mathcal{B}_n}\}_{n=1}^N$,
 masses: $M_{\mathcal{B}}$ and $M_{\mathcal{R}}$, density measure: ρ,
 threshold: $\theta(\geq 1)$

Output: a dimension in $[N]$

1 $\tilde{\rho} \leftarrow -\infty, \tilde{i} \leftarrow 1$ ▷ $\tilde{\rho}$: max ρ so far, $\tilde{i}$: dimension with $\tilde{\rho}$

2 for each dimension $i \in [N]$ **do**

3   **if** $\mathcal{B}_i \neq \emptyset$ **then**

4 $D_i \leftarrow \{a \in \mathcal{B}_i : M_{\mathcal{B}(a,i)} \leq \theta \frac{M_{\mathcal{B}}}{|\mathcal{B}_i|}\}$
            ▷ D_i : set to be removed

5 $M'_{\mathcal{B}} \leftarrow M_{\mathcal{B}} - \sum_{a\in D_i} M_{\mathcal{B}(a,i)}$

6 $\mathcal{B}'_i \leftarrow \mathcal{B}_i - D_i$

7 $\rho' \leftarrow \rho(M'_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n\neq i} \cup \{|\mathcal{B}'_i|\}, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N)$
            ▷ ρ' : ρ when D_i are removed

8   **if** $\rho' > \tilde{\rho}$ **then**

9 $\tilde{\rho} \leftarrow \rho', \tilde{i} \leftarrow i$ ▷ update max ρ so far

10 return $\tilde{i}$

attribute is computed. Then, the dimension attribute leading to the highest density is chosen. Note that the tuples in $\mathcal{B}$, stored on disk, do not need to be accessed for this computation, as described in Algorithm 4. Although this policy does not provide the accuracy guarantee given by the maximum cardinality policy, this policy works well with various density measures and tends to spot denser subtensors than the maximum cardinality policy in our experiments with real-world data.

3.1.4 *Efficient Implementation*

We present the optimization techniques used for the efficient implementation of D-CUBE.

Combining Disk-Accessing Steps. The amount of disk I/O can be reduced by combining multiple steps involving disk accesses. In Algorithm 1, updating $\mathcal{R}$ (line 7) in an iteration can be combined with computing the mass of $\mathcal{R}$ (line 5) in the next iteration. That is, if we aggregate the values of the tuples of $\mathcal{R}$ while they are written for the update, we do not need to scan $\mathcal{R}$ again for computing its mass in the next iteration. Likewise, in Algorithm 2, updating $\mathcal{B}$ (line 16) in an iteration can be combined with computing attribute-value masses (line 6) in the next iteration. This optimization reduces the amount of disk I/O in D-CUBE about 30%.

Caching Tensor Entries in Memory. Although we assume that tuples are stored on disk, storing them in memory up to the memory capacity speeds up D-CUBE up to 3 times in our experiments (see Section 4.4). We cache the tuples in $\mathcal{B}$, which are more frequently accessed than those in $\mathcal{R}$ or $\mathcal{R}^{ori}$, in memory with the highest priority.

3.2 Analyses

In this section, we prove the time and space complexities of D-CUBE and the accuracy guarantee provided by D-CUBE.

3.2.1 *Complexity Analyses*

Theorem 1 states the worst-case time complexity, which equals to the worst-case I/O complexity, of D-CUBE.

Lemma 1 (Maximum Number of Iterations in Algorithm 2).
Let $L = \max_{n\in[N]} |\mathcal{R}_n|$. *Then, the number of iterations (lines 5-16) in Algorithm 2 is at most*

$$N \min(\log_\theta L, L).$$

Proof. In each iteration (lines 5-16) of Algorithm 2, among the values of the chosen dimension attribute A_i, attribute values whose masses are at most $\theta\frac{M_{\mathcal{B}}}{|\mathcal{B}_i|}$, where $\theta \geq 1$, are removed. The set of such attribute values is denoted by D_i. We will show that, if $|\mathcal{B}_i| > 0$, then

$$|\mathcal{B}_i \backslash D_i| < |\mathcal{B}_i|/\theta. \tag{1}$$

Note that, when $|\mathcal{B}_i \backslash D_i| = 0$, Eq. (1) trivially holds. When $|\mathcal{B}_i \backslash D_i| > 0$, $M_{\mathcal{B}}$ can be factorized and lower bounded as

$$M_{\mathcal{B}} = \sum\nolimits_{a\in\mathcal{B}_i\backslash D_i} M_{\mathcal{B}(a,i)} + \sum\nolimits_{a\in D_i} M_{\mathcal{B}(a,i)}$$
$$\geq \sum\nolimits_{a\in\mathcal{B}_i\backslash D_i} M_{\mathcal{B}(a,i)} > |\mathcal{B}_i\backslash D_i| \cdot \theta\frac{M_{\mathcal{B}}}{|\mathcal{B}_i|},$$

where the last strict inequality is from the definition of D_i and that $|\mathcal{B}_i\backslash D_i| > 0$. This strict inequality implies $M_{\mathcal{B}} > 0$, and thus dividing both sides by $\theta\frac{M_{\mathcal{B}}}{|\mathcal{B}_i|}$ gives Eq. (1). Now, Eq. (1) implies that the number of remaining values of the chosen attribute after each iteration is less than $1/\theta$ of that before the iteration. Hence each attribute can be chosen at most $\log_\theta L$ times before all of its values are removed. Thus, the maximum number of iterations is at most $N\log_\theta L$. Also, by Eq. (1), at least one attribute value is removed per iteration. Hence, the maximum number of iterations is at most the number of attribute values, which is upper bounded by NL. Hence the number of iterations is upper bounded by $N\max(\log_\theta L, L)$. ∎

Theorem 1 (Worst-case Time Complexity). *Let* $L = \max_{n\in[N]} |\mathcal{R}_n|$. *If* $\theta = O\left(e^{(\frac{N|\mathcal{R}|}{L})}\right)$, *which is a weaker condition than* $\theta = O(1)$, *the worst-case time complexity of Algorithm 1 is*

$$O(kN^2|\mathcal{R}| \min(\log_\theta L, L)).$$

Proof. From Lemma 1, the number of iterations (lines 5-16) in Algorithm 2 is $O(N\min(\log_\theta L, L))$. Executing lines 6 and 16 $O(N\min(\log_\theta L, L))$ times takes $O(N^2|\mathcal{R}| \min(\log_\theta L, L))$, which dominates the time complexity of the other parts. For example, repeatedly executing line 9 takes $O(NL\log_2 L)$, and by our assumption, it is dominated by $O(N^2|\mathcal{R}| \min(\log_\theta L, L))$. Thus, the worst-case time complexity of Algorithm 2 is $O(N^2|\mathcal{R}| \min(\log_\theta L, L))$, and that of Algorithm 1, which executes Algorithm 2, k times, is $O(kN^2|\mathcal{R}| \min(\log_\theta L, L))$. ∎

However, this worst-case time complexity, which allows the worst distributions of the measure attribute values of tuples, is too pessimistic. In Section 4.4, we experimentally show that D-CUBE scales linearly with k, N, and $\mathcal{R}$; and sub-linearly with L even when θ is its smallest value 1.

Theorem 2 states the memory requirement of D-CUBE. Since the tuples do not need to be stored in memory all at once in D-CUBE, its memory requirement does not depend on the number of tuples (i.e., $|\mathcal{R}|$).

Theorem 2 (Memory Requirements). *The amount of memory space required by* D-CUBE *is* $O(\sum_{n=1}^N |\mathcal{R}_n|)$.

Proof. D-CUBE stores $\{\{M_{\mathcal{B}(a,n)}\}_{a\in\mathcal{B}_n}\}_{n=1}^N$, $\{\mathcal{R}_n\}_{n=1}^N$, and $\{\mathcal{B}_n\}_{n=1}^N$ in memory. Each has at most $\sum_{n=1}^N |\mathcal{R}_n|$ values. Thus, the memory requirement is $O(\sum_{n=1}^N |\mathcal{R}_n|)$. ∎

3.2.2 Accuracy Guarantee

We show that D-CUBE gives the same accuracy guarantee with in-memory algorithms [1], if we set θ to 1, although accesses to tuples (stored on disk) are restricted in D-CUBE to reduce disk I/Os. Specifically, Theorem 3 states that the subtensor found by Algorithm 2 with the maximum cardinality policy has density at least $\frac{1}{\theta N}$ of the optimum when ρ_{ari} is used as the density measure.

Theorem 3 (θN-Approximation Guarantee). *Let $\mathcal{B}^*$ be the subtensor $\mathcal{B}$ maximizing $\rho_{ari}(\mathcal{B}, \mathcal{R})$ in the given relation $\mathcal{R}$. Let $\tilde{\mathcal{B}}$ be the subtensor returned by Algorithm 2 with ρ_{ari} and the maximum cardinality policy. Then,*

$$\rho_{ari}(\tilde{\mathcal{B}}, \mathcal{R}) \geq \frac{1}{\theta N}\rho_{ari}(\mathcal{B}^*, \mathcal{R}).$$

Proof. First, the maximal subtensor $\mathcal{B}^*$ satisfies that, for any $i \in [N]$ and for any attribute value $a \in \mathcal{B}_i^*$, its attribute-value mass $M_{\mathcal{B}^*(a,i)}$ is at least $\frac{1}{N}\rho_{ari}(\mathcal{B}^*, \mathcal{R})$. This is since the maximality of $\rho_{ari}(\mathcal{B}^*, \mathcal{R})$ implies $\rho_{ari}(\mathcal{B}^* - \mathcal{B}^*(a,i), \mathcal{R}) \leq \rho_{ari}(\mathcal{B}^*, \mathcal{R})$, and plugging in Definition 1 to ρ_{ari} gives $\frac{M_{\mathcal{B}^*} - M_{\mathcal{B}^*(a,i)}}{\frac{1}{N}((\sum_{n=1}^N |\mathcal{B}_n^*|)-1)} = \rho_{ari}(\mathcal{B}^* - \mathcal{B}^*(a,i), \mathcal{R}) \leq \rho_{ari}(\mathcal{B}^*, \mathcal{R}) = \frac{M_{\mathcal{B}^*}}{\frac{1}{N}\sum_{n=1}^N |\mathcal{B}_n^*|}$, which reduces to

$$M_{\mathcal{B}^*(a,i)} \geq \frac{1}{N}\rho_{ari}(\mathcal{B}^*, \mathcal{R}). \qquad (2)$$

Consider the earliest iteration (lines 5-16) in Algorithm 2 where an attribute value a of $\mathcal{B}^*$ is included in D_i. Let $\mathcal{B}'$ be $\mathcal{B}$ in the beginning of the iteration. Our goal is to prove $\rho_{ari}(\tilde{\mathcal{B}}, \mathcal{R}) \geq \frac{1}{\theta N}\rho_{ari}(\mathcal{B}^*, \mathcal{R})$, which we will show as $\rho_{ari}(\tilde{\mathcal{B}}, \mathcal{R}) \geq \rho_{ari}(\mathcal{B}', \mathcal{R}) \geq \frac{M_{\mathcal{B}'(a,i)}}{\theta} \geq \frac{M_{\mathcal{B}^*(a,i)}}{\theta} \geq \frac{1}{\theta N}\rho_{ari}(\mathcal{B}^*, \mathcal{R})$.

First, $\rho_{ari}(\tilde{\mathcal{B}}, \mathcal{R}) \geq \rho_{ari}(\mathcal{B}', \mathcal{R})$ is from the maximality of $\rho_{ari}(\tilde{\mathcal{B}}, \mathcal{R})$ among the densities of the subtensors generated in the iterations (lines 13-15 in Algorithm 2). Second, applying $|\mathcal{B}_i'| \geq \frac{1}{N}\sum_{n=1}^N |\mathcal{B}_n'|$ from the maximum cardinality policy (Algorithm 3) to Definition 1 of ρ_{ari} gives $\rho_{ari}(\mathcal{B}', \mathcal{R}) = \frac{M_{\mathcal{B}'}}{\frac{1}{N}\sum_{n=1}^N |\mathcal{B}_n'|} \geq \frac{M_{\mathcal{B}'}}{|\mathcal{B}_i'|}$. And $a \in D_i$ gives $\theta\frac{M_{\mathcal{B}'}}{|\mathcal{B}_i'|} \geq M_{\mathcal{B}'(a,i)}$. So combining these gives $\rho_{ari}(\mathcal{B}', \mathcal{R}) \geq \frac{M_{\mathcal{B}'(a,i)}}{\theta}$. Third, $\frac{M_{\mathcal{B}'(a,i)}}{\theta} \geq \frac{M_{\mathcal{B}^*(a,i)}}{\theta}$ is from $\mathcal{B}' \supset \mathcal{B}^*$. Fourth, $\frac{M_{\mathcal{B}^*(a,i)}}{\theta} \geq \frac{1}{\theta N}\rho_{ari}(\mathcal{B}^*, \mathcal{R})$ is from Eq. (2). Hence, $\rho_{ari}(\tilde{\mathcal{B}}, \mathcal{R}) \geq \frac{1}{\theta N}\rho_{ari}(\mathcal{B}^*, \mathcal{R})$ holds. ∎

3.3 MapReduce Implementation

We present our MAPREDUCE implementation of D-CUBE, assuming that tuples in relations are stored in a distributed file system. Specifically, we describe four MAPREDUCE algorithms that cover the steps of D-CUBE accessing tuples.

(1) Filtering Tuples. In lines 7-8 of Algorithm 1 and line 16 of Algorithm 2, D-CUBE filters the tuples satisfying the given conditions. These steps are done by the following map-only algorithm, where we broadcast the data used in each condition (e.g., $\{\mathcal{B}_n\}_{n=1}^N$ in line 7 of Algorithm 1) to mappers using the distributed cache functionality.

- Map-stage: Take a tuple t (i.e., $\langle t[A_1], ..., t[A_N], t[X]\rangle$) and emit t if t satisfies the given condition. Otherwise, the tuple is ignored.

(2) Computing Attribute-value Masses. Line 6 of Algorithm 2 is performed by the following algorithm, where we reduce the amount of shuffled data by combining the intermediate results within each mapper.

- Map-stage: Take a tuple t (i.e., $\langle t[A_1], ..., t[A_N], t[X]\rangle$) and emit N key/value pairs $\{\langle (n, t[A_n]), t[X]\rangle\}_{n=1}^N$.
- Combine-stage/Reduce-stage: Take $\langle (n, a), \text{values}\rangle$ and emit $\langle (n, a), \text{sum(values)}\rangle$.

Each tuple $\langle (n, a), \text{value}\rangle$ of the final output indicates that $M_{\mathcal{B}(a,n)} = \text{value}$.

(3) Computing Mass. Line 5 of Algorithm 1 can be performed by the following algorithm, where we reduce the amount of shuffled data by combining the intermediate results within each mapper.

- Map-stage: Take a tuple t (i.e., $\langle t[A_1], ..., t[A_N], t[X]\rangle$) and emit $\langle 0, t[X]\rangle$.
- Combine-stage/Reduce-stage: Take $\langle 0, \text{values}\rangle$ and emit $\langle 0, \text{sum(values)}\rangle$.

The value of the final tuple corresponds to $M_{\mathcal{R}}$.

(4) Computing Attribute-value Sets. Line 2 of Algorithm 1 can be performed by the following algorithm, where we reduce the amount of shuffled data by combining the intermediate results within each mapper.

- Map-stage: Take a tuple t (i.e., $\langle t[A_1], ..., t[A_N], t[X]\rangle$) and emit N key/value pairs $\{\langle (n, t[A_n]), 0\rangle\}_{n=1}^N$.
- Combine-stage/Reduce-stage: Take $\langle (n, a), \text{values}\rangle$ and emit $\langle (n, a), 0\rangle$.

Each tuple $\langle (n, a), 0\rangle$ of the final output indicates that a is a member of $\mathcal{R}_n$.

4 EXPERIMENTS

We designed and conducted experiments to answer the following questions:

- **Q1. Memory Efficiency**: How much memory space does D-CUBE require for analyzing real-world tensors? How large tensors can D-CUBE handle?
- **Q2. Speed and Accuaracy**: How fast and accurately does D-CUBE spot dense subtensors?
- **Q3. Scalability**: Does D-CUBE scale linearly with all aspects of data? Does D-CUBE scale out?
- **Q4. Effectiveness**: Which anomalies and fraud does D-CUBE detect in real-world tensors?
- **Q5. Effect of θ**: How does the mass-threshold parameter θ affect the speed and accuracy of D-CUBE?
- **Q6. Effect of α**: How does the parameter α in density metric $\rho_{es(\alpha)}$ affect subtensors that D-CUBE detects?

4.1 Experimental Settings

4.1.1 Machines

We ran all serial algorithms on a machine with 2.67GHz Intel Xeon E7-8837 CPUs and 1TB memory. We ran MAPREDUCE algorithms on a 40-node Hadoop cluster, where each node has an Intel Xeon E3-1230 3.3GHz CPU and 32GB memory.

4.1.2 Datasets

We describe the real-world and synthetic tensors used in our experiments. Real-world tensors are categorized into four groups: (a) Rating data (SWM, Yelp, Android, Netflix, and YahooM.), (b) Wikipedia revision histories (KoWiki and EnWiki), (c) Temporal social networks (Youtube and SMS), and (d) TCP dumps (DARPA and AirForce). Some statistics of these datasets are summarized in Table 3.

Rating data. Rating data are relations with schema (<u>user</u>, <u>item</u>, <u>timestamp</u>, <u>score</u>, #ratings). Each tuple (u,i,t,s,r) indicates that user u gave item i score s, r times, at timestamp t. In SWM Dataset [16], the timestamps are in dates, and the items are entertaining software from a popular online software marketplace. In Yelp Dataset [17], the timestamps are in dates, and the items are businesses listed on Yelp, a review site. In Android Dataset [18], the timestamps are hours, and the items are Android apps on Amazon, an online store. In Netflix Dataset [19], the timestamps are in dates, and the items are movies listed on Netflix, a movie rental and streaming service. In YahooM. Dataset [20], the timestamps are in hours, and the items are musical items listed on Yahoo! Music, a provider of various music services.

Wikipedia revision history. Wikipedia revision histories are relations with schema (<u>user</u>, <u>page</u>, <u>timestamp</u>, #revisions). Each tuple (u,p,t,r) indicates that user u revised page p, r times, at timestamp t (in hour) in Wikipedia, a crowdsourcing online encyclopedia. In KoWiki Dataset [1], the pages are from Korean Wikipedia. In EnWiki Dataset [1], the pages are from English Wikipedia.

Temporal social networks. Temporal social networks are relations with schema (<u>source</u>, <u>destination</u>, <u>timestamp</u>, #interactions). Each tuple (s,d,t,i) indicates that user s interacts with user d, i times, at timestamp t. In Youtube Dataset [21], the timestamps are in hours, and the interactions are becoming friends on Youtube, a video-sharing website. In SMS Dataset, the timestamps are in hours, and the interactions are sending text messages.

TCP Dumps. DARPA Dataset [22], collected by the Cyber Systems and Technology Group in 1998, is a relation with schema (<u>source IP</u>, <u>destination IP</u>, <u>timestamp</u>, #connections). Each tuple (s,d,t,c) indicates that c connections were made from IP s to IP d at timestamp t (in minutes). AirForce [23] Dataset, used for KDD Cup 1999 [23], is a relation with schema (<u>protocol</u>, <u>service</u>, <u>src bytes</u>, <u>dst bytes</u>, <u>flag</u>, <u>host count</u>, <u>srv count</u>, #connections). The description of each attribute is as follows:

- protocol: type of protocol (tcp, udp, etc.).
- service: service on destination (http, telnet, etc.).
- src bytes: bytes sent from source to destination.
- dst bytes: bytes sent from destination to source.
- flag: normal or error status.
- host count: number of connections made to the same host in the past two seconds.
- srv count: number of connections made to the same service in the past two seconds.
- #connections: number of connections with the given dimension attribute values.

Synthetic Tensors: We used synthetic tensors for scalability tests. Each tensor was created by generating a random binary tensor and injecting ten random dense subtensors,

TABLE 3: Summary of real-world datasets.

Name	Volume	#Tuples
Rating data (<u>user</u>, <u>item</u>, <u>timestamp</u>, <u>rating</u>, #reviews)		
SWM [16]	967K × 15.1K × 1.38K × 5	1.13M
Yelp [17]	552K × 77.1K × 3.80K × 5	2.23M
Android [18]	1.32M × 61.3K × 1.28K × 5	2.64M
Netflix [19]	480K × 17.8K × 2.18K × 5	99.1M
YahooM. [20]	1.00M × 625K × 84.4K × 101	253M
Wiki revision histories (<u>user</u>, <u>page</u>, <u>timestamp</u>, #revisions)		
KoWiki [1]	470K × 1.18M × 101K	11.0M
EnWiki [1]	44.1M × 38.5M × 129K	483M
Social networks (<u>user</u>, <u>user</u>, <u>timestamp</u>, #interactions)		
Youtube [21]	3.22M × 3.22M × 203	18.7M
SMS	1.25M × 7.00M × 4.39K	103M
TCP dumps (<u>src IP</u>, <u>dst IP</u>, <u>timestamp</u>, #connections)		
DARPA [22]	9.48K × 23.4K × 46.6K	522K
TCP dumps (<u>protocol</u>, <u>service</u>, <u>src bytes</u>, ···, #connections)		
AirForce [23]	3 × 70 × 11 × 7.20K × 21.5K × 512 × 512	648K

whose volumes are 10^N and densities (in terms of ρ_{ari}) are between $10\times$ and $100\times$ of that of the entire tensor.

4.1.3 Implementations

We explain the implementations of the dense-subtensor detection methods used in our experiments.

- D-CUBE (Proposed): We implemented D-CUBE in Java with Hadoop 1.2.1. We set the mass-threshold parameter θ to 1 and used the maximum density policy for dimension selection, unless otherwise stated.
- M-ZOOM [1]: We used the open-source Java implementation of M-ZOOM[2].
- CROSSSPOT [3]: We used a Java implementation of the open-source implementation of CROSSSPOT[3]. Although CROSSSPOT was originally designed to maximize ρ_{susp}, we used its variants that directly maximize the density metric compared in each experiment. We used CPD as the seed selection method of CROSSSPOT as in [1].
- CPD (CP Decomposition): Let $\{\mathbf{A}^{(n)}\}_{n=1}^{N}$ be the factor matrices obtained by CP Decomposition [24]. The i-th dense subtensor is composed by every attribute value a_n whose corresponding element in the i-th column of $A^{(n)}$ is greater than or equal to $1/\sqrt{|\mathcal{R}_n|}$. We used Tensor Toolbox [25] for CP Decomposition.
- MAF [2]: We used Tensor Toolbox [25] for CP Decomposition, which MAF is largely based on.

4.2 Q1. Memory Efficiency

We compare the amount of memory required by different methods for handling the real-world datasets. As seen in Figure 4, D-CUBE, which does not require tuples to be stored in memory, needed up to **1,600× less memory** than the second best method, which stores tuples in memory.

Due to its memory efficiency, D-CUBE successfully handled **1,000× larger data** than its competitors within a memory budget. We ran methods on 3-way synthetic tensors

2. https://github.com/kijungs/mzoom
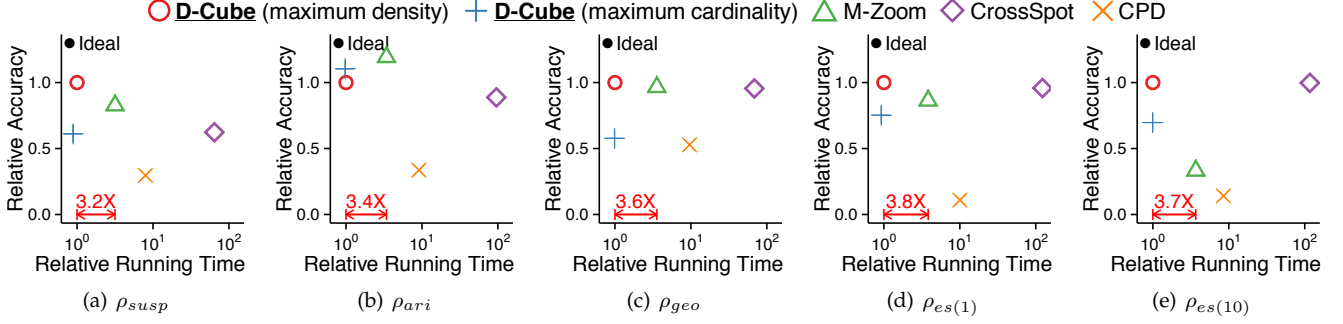3. https://github.com/mjiang89/CrossSpot

Fig. 3: **D-CUBE achieves both speed and accuracy (on average).** In each plot, points represent the speed and accuracy of different methods averaged over all real-world tensors. Upper-left region indicates better performance. D-CUBE is about **3.6× faster** than the second best method M-ZOOM. Moreover, D-CUBE with the maximum density consistently shows high accuracy regardless of density measures, while the other methods do not.
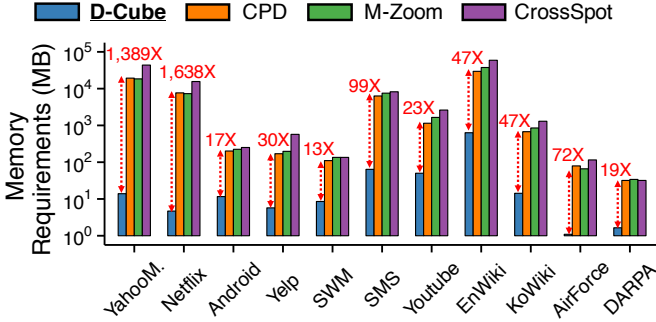

Fig. 4: **D-CUBE is memory efficient.** D-CUBE requires up to **1,600× less memory** than its competitors.

with different numbers of tuples (i.e., $|\mathcal{R}|$), with a memory budget of 16GB per machine. In every tensor, the cardinality of each dimension attribute was $1/1000$ of the number of tuples, i.e., $|\mathcal{R}_n| = |\mathcal{R}|/1000$, $\forall n \in [N]$. Figure 1(a) in Section 1 shows the result. The HADOOP implementation of D-CUBE successfully spotted dense subtensors in a tensor with 10^{11} tuples (**2.6TB**), and the serial version of D-CUBE successfully spotted dense subtensors in a tensor with 10^{10} tuples (**240GB**), which was the largest tensor that can be stored on a disk. However, all other methods ran out of memory even on a tensor with 10^{9} tuples (21GB).

4.3 Q2. Speed and Accuracy

We compare how rapidly and accurately D-CUBE (the serial version) and its competitors detect dense subtensors in the real-world datasets. We measured the wall-clock time (average over three runs) taken for detecting three subtensors by each method, and we measured the maximum density of the three subtensors found by each method using different density measures in Section 2.2. For this experiment, we did not limit the memory budget so that every method can handle every dataset. D-CUBE also utilized extra memory space by caching tuples in memory, as explained in Section 3.1.4.

Figure 3 shows the results averaged over all datasets [4], and Figure 9 shows the results in each dataset. D-CUBE provided the best trade-off between speed and accuracy.

---

4. We computed the relative running time and relative accuracy of each method (compared to the running time and accuracy of D-CUBE with the maximum density policy) in each dataset. Then, we averaged them over all datasets.

Specifically, **D-CUBE was up to 7× faster** (on average 3.6× faster) than the second fastest method M-ZOOM. Moreover, **D-CUBE (with the maximum density policy) consistently spotted high-density subtensors**, while the accuracies of the other methods varied on density measures. Specifically, on average, D-CUBE (with the maximum density policy) showed the highest accuracy with all the density measures except ρ_{ari}, which M-ZOOM and D-CUBE (with the maximum cardinality policy) were more accurate with. Although MAF does not appear in Figures 3 and 9, it consistently provided sparser subtensors than CPD with similar speed.

4.4 Q3. Scalability

We show that D-CUBE scales (sub-)linearly with every input factor, i.e., the number of tuples, the number of dimension attributes, and the cardinality of dimension attributes, and the number of subtensors that we aim to find. To measure the scalability with each factor, we started with finding a dense subtensor in a synthetic tensor with 10^{8} tuples and 3 dimension attributes each of whose cardinality is 10^{5}. Then, we measured the running time as we changed one factor at a time while fixing the other factors. The threshold parameter θ was fixed to 1. As seen in Figure 5, D-CUBE scaled linearly with every factor and sub-linearly with the cardinality of attributes even when θ was set to its minimum value 1. This supports our claim in Section 3.2.1 that the worst-case time complexity of D-CUBE (Theorem 1) is too pessimistic. This linear scalability of D-CUBE held both with enough memory budget (blue solid lines in Figure 5) to store all tuples and with minimum memory budget (red dashed lines in Figure 5) to barely meet the requirements although D-CUBE was up to 3× faster in the former case.

We also evaluate the machine scalability of the MAPRE-DUCE implementation of D-CUBE. We measured its running time taken for finding a dense subtensor in a synthetic tensor with 10^{10} tuples and 3 dimension attributes each of whose cardinality is 10^{7}, as we increased the number of machines running in parallel from 1 to 40. Figure 6 shows the changes in the running time and the speed-up, which is defined as T_1/T_M where T_M is the running time with M machines. The speed-up increased near linearly when a small number of machines were used, while it flattened as more machines were added due to the overhead in the distributed system.
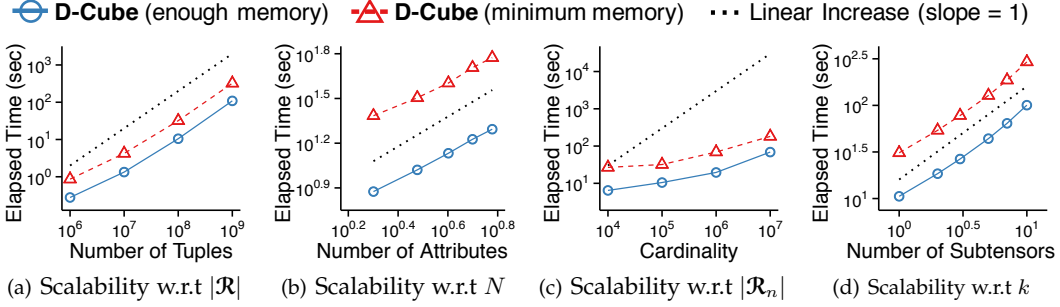
Fig. 5: **D-CUBE scales (sub-)linearly with all input factors** regardless of memory budgets.
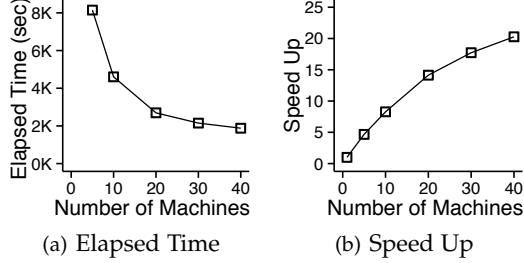

(a) Elapsed Time   (b) Speed Up

Fig. 6: **D-CUBE scales out.** The MAPREDUCE implementation of D-CUBE was speeded up 8× with 10 machines, and 20× with 40 machines.

4.5 Q4. Effectiveness

We demonstrate the effectiveness of D-CUBE in four applications using real-world tensors.

Network Intrusion Detection from TCP Dumps. D-CUBE detected network attacks from TCP dumps accurately by spotting corresponding dense subtensors. We consider two TCP dumps that are modeled differently. DARPA Dataset is a 3-way tensor where the dimension attributes are source IPs, destination IPs, and timestamps in minutes; and the measure attribute is the number of connections. AirForce Dataset, which does not include IP information, is a 7-way tensor where the measure attribute is the same but the dimension attributes are the features of the connections, including protocols and services. Both datasets include labels indicating whether each connection is malicious or not.

Figure 1(c) in Section 1 lists the five densest subtensors (in terms of ρ_{geo}) found by D-CUBE in each dataset. Notice that the dense subtensors are mostly composed of various types of network attacks. Based on this observation, we classified each connection as malicious or benign based on the density of the densest subtensor including the connection (i.e., the denser the subtensor including a connection is, the more suspicious the connection is). This led to high accuracy as seen in Table 4, which reports the accuracy when each method (with the density measure giving the highest accuracy) was used for dense-subtensor detection. In both datasets, using D-CUBE resulted in the highest accuracy.

Synchronized Behavior Detection in Rating Data. D-CUBE spotted suspicious synchronized behavior accurately in rating data. Specifically, we assume an attack scenario where fraudsters in a review site, who aim to boost (or lower) the ratings of the set of items, create multiple user accounts and give the same score to the items within a short period of time. This lockstep behavior forms a dense

TABLE 4: **D-CUBE spots network attacks fastest and most accurately** from TCP dumps.

Datasets	AirForce		DARPA	
	Elapsed Time (sec)	Accuracy (AUC)	Elapsed Time (sec)	Accuracy (AUC)
CPD [24]	413.2	0.854	105.0	0.926
MAF [2]	486.6	0.912	102.4	0.514
CROSSSPOT [3]	575.5	0.924	132.2	0.923
M-ZOOM [1]	27.7	0.975	22.7	0.923
D-CUBE	**15.6**	**0.987**	**9.1**	**0.930**

TABLE 5: **D-CUBE detects synchronized behavior fastest and most accurately** in rating datasets.

Datasets	Android		Yelp	
	Elapsed Time (sec)	Recall @ Top-10	Elapsed Time (sec)	Recall @ Top-10
CPD [24]	59.9	0.54	47.5	0.52
MAF [2]	95.0	0.54	49.4	0.52
CROSSSPOT [3]	71.3	0.54	56.7	0.52
M-ZOOM [1]	28.4	0.70	17.7	0.30
D-CUBE	**7.0**	**0.90**	**4.9**	**0.60**

subtensor with volume (# fake accounts × # target items × 1 × 1) in the rating dataset, whose dimension attributes are users, items, timestamps, and rating scores.

We injected 10 such random dense subtensors whose volumes varied from 15×15×1×1 to 60×60×1×1 in Yelp and Android Datasets. We compared the number of the injected subtensors detected by each dense-subtensor detection method. We considered each injected subtensor as overlooked by a method, if the subtensor did not belong to any of the top-10 dense subtensors spotted by the method or it was hidden in a natural dense subtensor at least 10 times larger than the injected subtensor. We repeated this experiment 10 times, and the averaged results are summarized in Table 5. For each method, we report the results with the density measure giving the highest accuracy. In both datasets, D-CUBE detected a largest number of the injected subtensors. Especially, in Android Dataset, D-CUBE detected 9 out of the 10 injected subtensors, while the second best method detected only 7 injected subtensors on average.

Spam-Review Detection in Rating Data. D-CUBE successfully spotted spam reviews in SWM Dataset, which contains reviews from an online software marketplace. We modeled SWM Dataset as a 4-way tensor whose dimension attributes are users, software, ratings, and timestamps in dates, and we applied D-CUBE (with $\rho = \rho_{ari}$) to the dataset. Table 7 shows the statistics of the top-3 dense subtensors. Although ground-truth labels were not available,

TABLE 6: **D-CUBE successfully detects spam reviews** in SWM Dataset.

Subtensor 1 (100% spam)			Subtensor 2 (100% spam)			Subtensor 3 (at least 48% spam)		
User	Review	Date	User	Review	Date	User	Review	Date
Ti*	type in *** and you will get ...	Mar-4	Sk*	invite code***, referral— ...	Apr-18	Mr*	entered this code and got ...	Nov-23
Fo*	type in for the bonus code: ...	Mar-4	fu*	use my code for bonus ...	Apr-18	Max*	enter the bonus code: *** ...	Nov-23
dj*	typed in the code: *** ...	Mar-4	Ta*	enter the code *** for ...	Apr-18	Je*	enter *** when it asks...	Nov-23
Di*	enter this code to start with ...	Mar-4	Ap*	bonus code *** for points ...	Apr-18	Man*	just enter *** for a boost ...	Nov-23
Fe*	enter code: *** to win even ...	Mar-4	De*	bonus code: ***, be one ...	Apr-18	Ty*	enter *** ro receive a ...	Nov-23

TABLE 7: Summary of the dense subtensors that D-CUBE detects in real-world datasets.

Dataset	Order	Volume	Mass	ρ_{ari}	Type
SWM	1	120	308	44.0	Spam reviews
	2	612	435	31.6	Spam reviews
	3	231,240	771	20.3	Spam reviews
KoWiki	1	8	546	273.0	Edit war
	2	80	1,011	233.3	Edit war
	3	270	1,126	168.9	Edit war
EnWiki	1	9.98M	1.71M	7,931	Bot activities
	2	541K	343K	4,211	Bot activities
	3	23.5M	973K	3,395	Bot activities

TABLE 8: **D-CUBE successfully spots bot activities** in EnWiki Dataset.

Subtensor #	Users in each subtensor (100% bots)
1	WP 1.0 bot
2	AAlertBot
3	AlexNewArtBot, VeblenBot, InceptionBot
4	WP 1.0 bot
5	Cydebot, VeblenBot

as the examples in Table 6 show, all the reviews composing the first and second dense subtensors were obvious spam reviews. In addition, at least 48% of the reviews composing the third dense subtensor were obvious spam reviews.

Anomaly Detection in Wikipedia Revision Histories. D-CUBE detected interesting anomalies in Wikipedia revision histories, which we model as 3-way tensors whose dimension attributes are users, pages, and timestamps in hours. Table 7 gives the statistics of the top-3 dense subtensors detected by D-CUBE (with $\rho = \rho_{ari}$ and the maximum cardinality policy) in KoWiki Dataset and by D-CUBE (with $\rho = \rho_{geo}$ and the maximum density policy) in EnWiki Dataset. All three subtensors detected in KoWiki Dataset indicated edit wars. For example, the second subtensor corresponded to an edit war where 4 users changed 4 pages, 1,011 times, within 5 hours. On the other hand, all three subtensors detected in Enwiki Dataset indicated bot activities. For example, the third subtensor corresponded to 3 bots which edited 1,067 pages 973,747 times. The users composing the top-5 dense subtensors in EnWiki Dataset are listed in Table 8. Notice that all of them are bots.

4.6 Q5. Effects of Parameter θ on Speed and Accuracy

We investigate the effects of the mass-threshold parameter θ on the speed and accuracy of D-CUBE in the real-world datasets. We used the serial version of D-CUBE with a memory budget of 16GB, and we measured its relative accuracy and speed as in Section 4.3. Figure 7 shows the results averaged over all datasets. Different θ values provided a

trade-off between speed and accuracy. Specifically, increasing θ tended to make D-CUBE faster but less accurate. This tendency is consistent with our theoretical analyses (Theorems 1-3 in Section 3.2). The sensitivity of the accuracy to θ depended on the used density measures. Specifically, the sensitivity was lower with $\rho_{es(\alpha)}$ than with the other density measures.

4.7 Q6. Effects of Parameter α in $\rho_{es(\alpha)}$ on Subtensors detected by D-CUBE

We show that the dense subtensors detected by D-CUBE are configurable by the parameter α in density measure $\rho_{es(\alpha)}$. Figure 8 shows the volumes and masses of subtensors detected in Youtube and Yelp Datasets by D-CUBE when $\rho_{es(\alpha)}$ with different α values were used as the density metrics. With large α values, D-CUBE tended to spot relatively small but compact subtensors. With small α values, however, D-CUBE tended to spot relatively sparse but large subtensors. Similar tendencies were obtained with the other datasets.

5 RELATED WORK

We discuss previous work on (a) dense-subgraph detection, (b) dense-subtensor detection, (c) large-scale tensor decomposition, and (d) other anomaly/fraud detection methods.

Dense Subgraph Detection. Dense-subgraph detection in graphs has been extensively studied in theory; see [26] for a survey. Exact algorithms [14], [15] and approximate algorithms [10], [15] have been proposed for finding subgraphs with maximum average degree. These have been extended for incorporating size restrictions [27], alternative metrics for denser subgraphs [13], evolving graphs [28], subgraphs with limited overlap [29], [30], and streaming or distributed settings [31], [32]. Dense subgraph detection has been applied to fraud detection in social or review networks [5], [12], [33], [34], [35].

Dense Subtensor Detection. Extending dense subgraph detection to tensors [1], [3] incorporates additional dimensions, such as time, to identify dense regions of interest with greater accuracy and specificity. CROSSSPOT [3], which starts from a seed subtensor and adjusts it in a greedy way until it reaches a local optimum, shows high accuracy in practice but does not provide any theoretical guarantees on its running time and accuracy. M-ZOOM [1], which starts from the entire tensor and only shrinks it by removing attributes one by one in a greedy way, improves CROSSSPOT in terms of speed and approximation guarantees. Both methods, however, require the tuples of relations to be loaded into memory at once and to be randomly accessed, which limit their applicability to large-scale datasets. Dense-subtensor detection in tensors has been found useful for
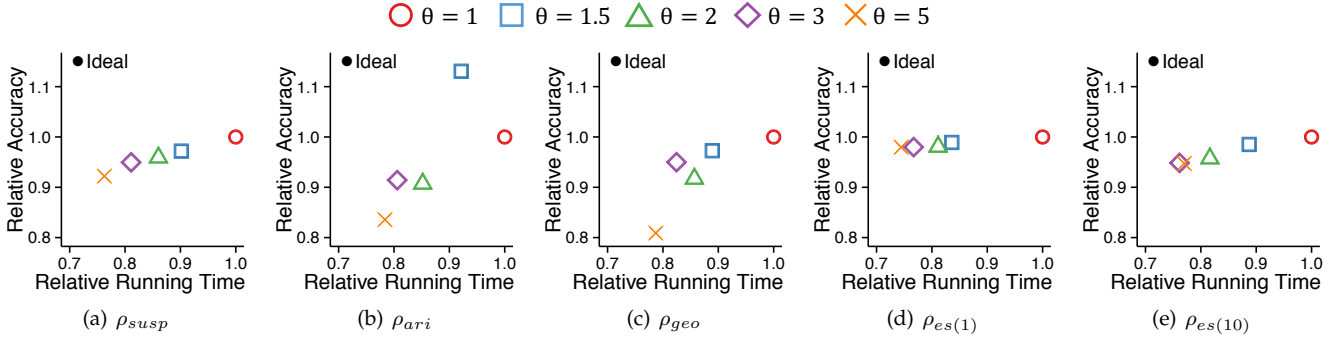
$\bigcirc\ \theta = 1$ $\square\ \theta = 1.5$ $\triangle\ \theta = 2$ $\diamond\ \theta = 3$ $\times\ \theta = 5$

(a) ρ_{susp} (b) ρ_{ari} (c) ρ_{geo} (d) $\rho_{es(1)}$ (e) $\rho_{es(10)}$

Fig. 7: **The mass-threshold parameter θ gives a trade-off between the speed and accuracy of D-Cube.** We report the accuracy and speed averaged over all real-world datasets. As θ increases, D-Cube tends to be faster but less accurate.
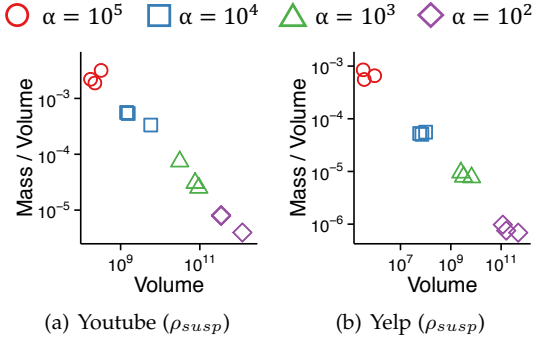

$\bigcirc\ \alpha = 10^5$ $\square\ \alpha = 10^4$ $\triangle\ \alpha = 10^3$ $\diamond\ \alpha = 10^2$

(a) Youtube (ρ_{susp})   (b) Yelp (ρ_{susp})

Fig. 8: **Subtensors detected by D-Cube are configurable** by the parameter α in density metric $\rho_{es(\alpha)}$. As α increases, D-Cube spots smaller but more compact subtensors.

detecting retweet boosting [3], network attacks [1], [2], bot activities [1], and for genetics applications [2], [4].

Large-Scale Tensor Decomposition. Tensor decomposition such as HOSVD and CP decomposition [24] can be used to spot dense subtensors [2]. Scalable algorithms for tensor decomposition have been developed, including disk-based algorithms [36], [37], distributed algorithms [37], [38], [39], and approximate algorithms based on sampling [40] and count-min sketch [41]. However, dense-subtensor detection based on tensor decomposition has serious limitations: it usually detects subtensors with significantly lower density (see Section 4.3) than search-based methods, provides no flexibility with regard to the choice of density metric, and does not provide any approximation guarantee.

Other Anomaly/Fraud Detection Methods. In addition to dense-subtensor detection, many approaches, including those based on egonet features [42], coreness [35], and behavior models [43], have been used for anomaly and fraud detection in graphs. See [44] for a survey.

6 CONCLUSION

In this work, we propose D-Cube, a disk-based dense-subtensor detection method, to deal with disk-resident tensors too large to fit in main memory. D-Cube is optimized to minimize disk I/Os while providing a guarantee on the quality of the subtensors it finds. Moreover, we present the distributed version of D-Cube running on MapReduce for terabyte-scale or larger data distributed across multiple machines. In summary, D-Cube achieves the following advantages over its state-of-the-art competitors:

- **Memory Efficient:** D-Cube handles *1,000×* larger data (*2.6TB*) by reducing memory usage up to *1,600×* compared to in-memory algorithms (Section 4.2).
- **Fast:** Even when data fit in memory, D-Cube is up to *7×* faster than its competitors (Section 4.3) with near-linear scalability (Section 4.4).
- **Provably Accurate:** D-Cube is one of the methods giving the best approximation guarantee (Theorem 3) and the densest subtensors in practice (Section 4.3).
- **Effective:** D-Cube was most accurate in two applications: detecting network attacks from TCP dumps and lockstep behavior in rating data (Section 4.5).

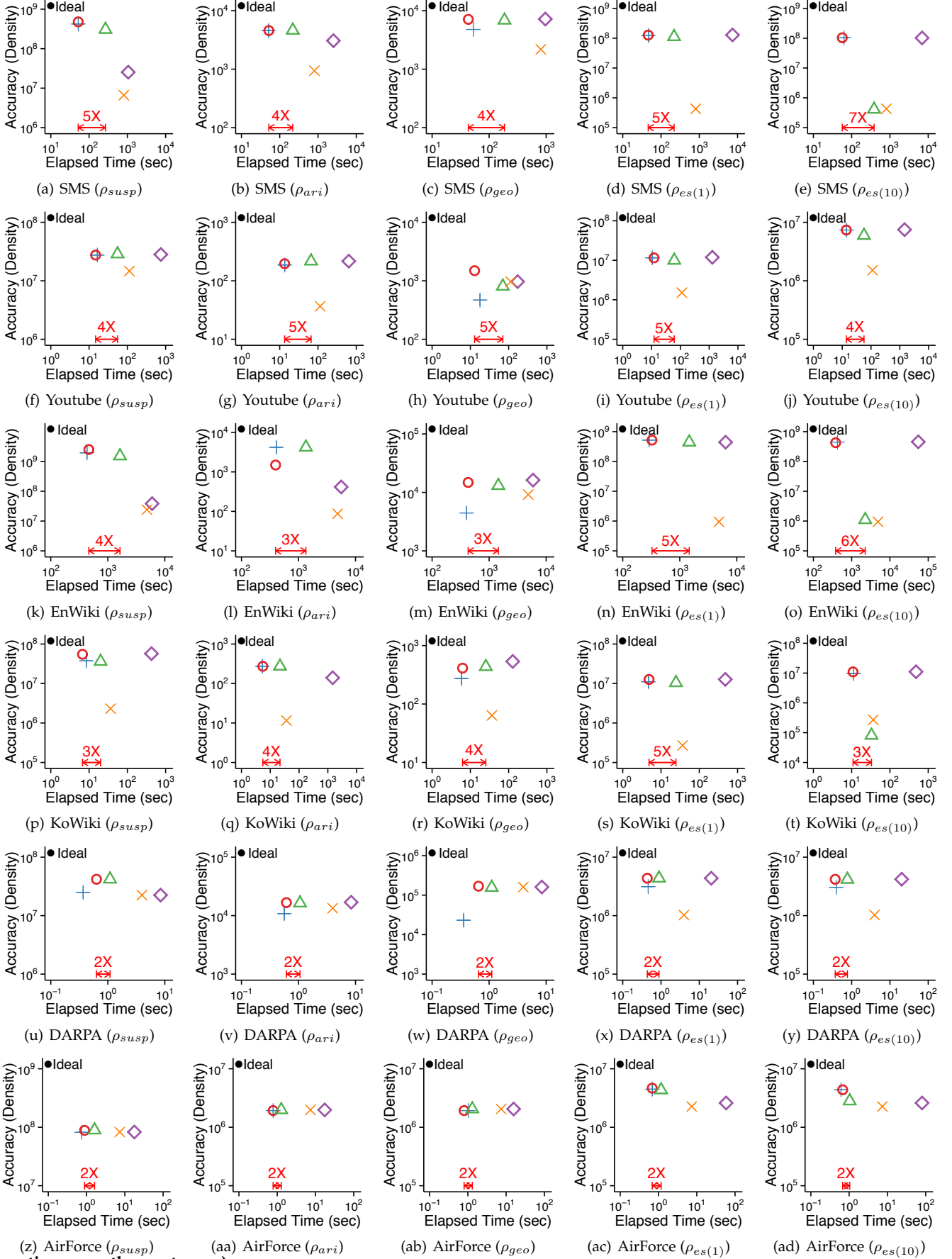**Reproducibility:** The code and data used in the paper are available at **http://www.cs.cmu.edu/~kijungs/codes/dcube**.

REFERENCES

[1] K. Shin, B. Hooi, and C. Faloutsos, "M-zoom: Fast dense-block detection in tensors with quality guarantees," in *ECML/PKDD*, 2016.

[2] K. Maruhashi, F. Guo, and C. Faloutsos, "Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis," in *ASONAM*, 2011.

[3] M. Jiang, A. Beutel, P. Cui, B. Hooi, S. Yang, and C. Faloutsos, "A general suspiciousness metric for dense blocks in multimodal data," in *ICDM*, 2015.

[4] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang, "Dense subgraphs with restrictions and applications to gene annotation graphs," in *RECOMB*, 2010.

[5] B. Hooi, K. Shin, H. A. Song, A. Beutel, N. Shah, and C. Faloutsos, "Graph-based fraud detection in the face of camouflage," *ACM Transactions on Knowledge Discovery from Data*, vol. 11, no. 4, p. 44, 2017.

[6] K. Shin, B. Hooi, J. Kim, and C. Faloutsos, "D-cube: Dense-block detection in terabyte-scale tensors," in *WSDM*, 2017.

Legend: ○ **D-Cube** (maximum density) + **D-Cube** (maximum cardinality) △ M-Zoom ◇ CrossSpot ✕ CPD

(a) SMS (ρ_{susp})   (b) SMS (ρ_{ari})   (c) SMS (ρ_{geo})   (d) SMS ($\rho_{es(1)}$)   (e) SMS ($\rho_{es(10)}$)

(f) Youtube (ρ_{susp})   (g) Youtube (ρ_{ari})   (h) Youtube (ρ_{geo})   (i) Youtube ($\rho_{es(1)}$)   (j) Youtube ($\rho_{es(10)}$)

(k) EnWiki (ρ_{susp})   (l) EnWiki (ρ_{ari})   (m) EnWiki (ρ_{geo})   (n) EnWiki ($\rho_{es(1)}$)   (o) EnWiki ($\rho_{es(10)}$)

(p) KoWiki (ρ_{susp})   (q) KoWiki (ρ_{ari})   (r) KoWiki (ρ_{geo})   (s) KoWiki ($\rho_{es(1)}$)   (t) KoWiki ($\rho_{es(10)}$)

(u) DARPA (ρ_{susp})   (v) DARPA (ρ_{ari})   (w) DARPA (ρ_{geo})   (x) DARPA ($\rho_{es(1)}$)   (y) DARPA ($\rho_{es(10)}$)

(z) AirForce (ρ_{susp})   (aa) AirForce (ρ_{ari})   (ab) AirForce (ρ_{geo})   (ac) AirForce ($\rho_{es(1)}$)   (ad) AirForce ($\rho_{es(10)}$)
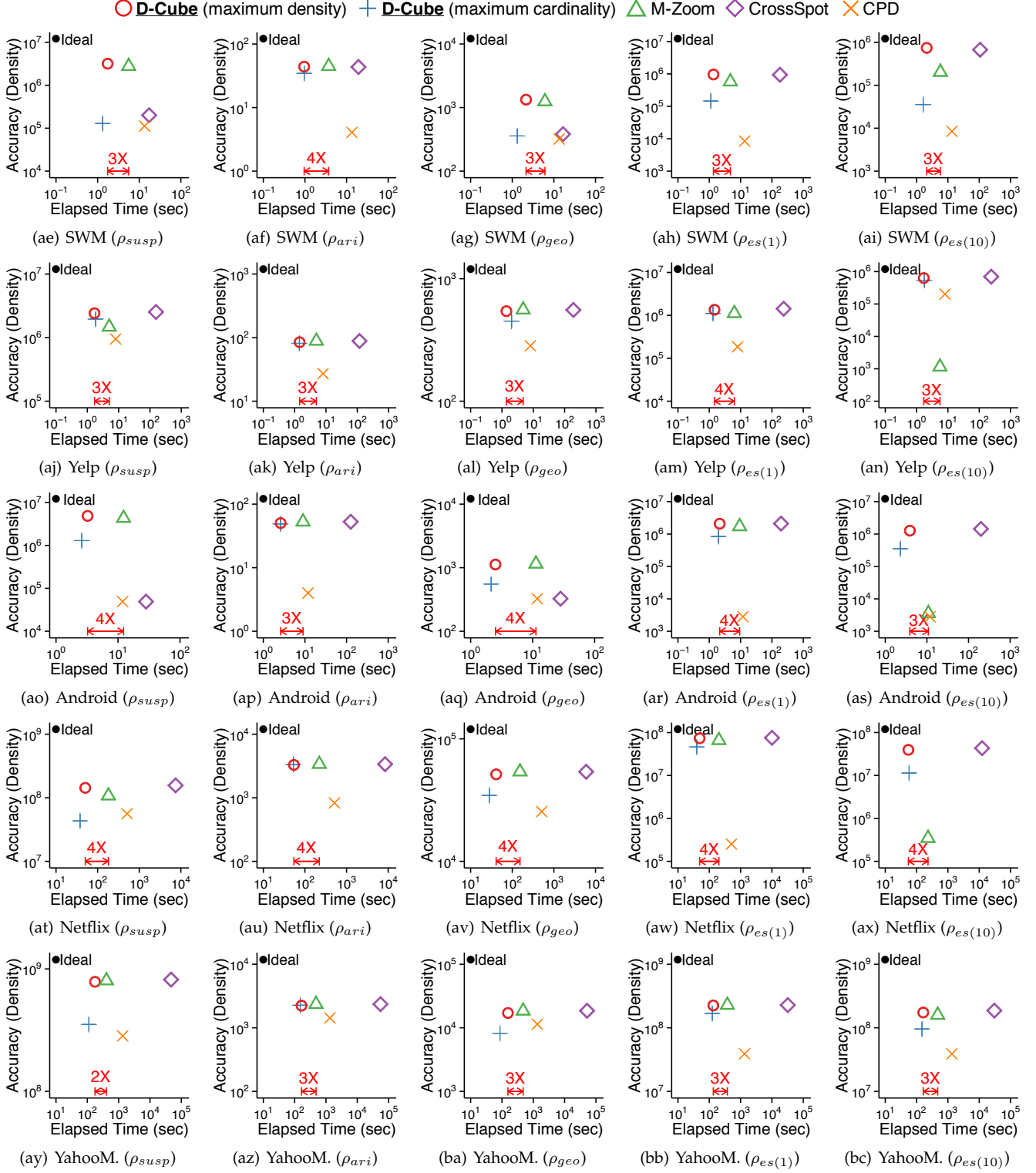
(continues on the next page)

Fig. 9: **D-CUBE achieves both speed and accuracy (in every dataset).** In each plot, points represent the speed and accuracy of different methods. Upper-left region indicates better performance. D-CUBE is up to $7\times$ **faster** than the second fast method M-ZOOM. Moreover, D-CUBE with the maximum density policy is the only method that is consistently accurate regardless of density measures.

[7] J. M. Ruhl, "Efficient algorithms for new computational models," Ph.D. dissertation, 2003.

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[9] "Apache hadoop." [Online]. Available: http://hadoop.apache.org/

[10] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph," in *APPROX*, 2000.

[11] R. Kannan and V. Vinay, *Analyzing the structure of large graphs*. Technical Report, 1999.

[12] M. Jiang, P. Cui, A. Beutel, C. Faloutsos, and S. Yang, "Catchsync: catching synchronized behavior in large directed graphs," in *KDD*, 2014.

[13] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli, "Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees," in *KDD*, 2013.

[14] A. V. Goldberg, *Finding a maximum density subgraph*. Technical Report, 1984.

[15] S. Khuller and B. Saha, "On finding dense subgraphs," in *ICALP*, 2009, pp. 597–608.

[16] L. Akoglu, R. Chandy, and C. Faloutsos, "Opinion fraud detection in online reviews by network effects." *ICWSM*, 2013.

[17] "Yelp dataset challenge." [Online]. Available: https://www.yelp.com/dataset_challenge

[18] J. McAuley, R. Pandey, and J. Leskovec, "Inferring networks of substitutable and complementary products," in *KDD*, 2015.

[19] J. Bennett and S. Lanning, "The netflix prize," in *KDD Cup*, 2007.

[20] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer, "The yahoo! music dataset and kdd-cup'11." in *KDD Cup*, 2012.

[21] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," in *IMC*, 2007.

[22] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," in *DISCEX*, 2000.

[23] "Kdd cup 1999 data." [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[24] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[25] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.6," Available online. [Online]. Available: http://www.sandia.gov/~tgkolda/TensorToolbox/

[26] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal, "A survey of algorithms for dense subgraph discovery," in *Managing and Mining Graph Data*, 2010, pp. 303–336.

[27] R. Andersen and K. Chellapilla, "Finding dense subgraphs with size bounds," in *WAW*, 2009.

[28] A. Epasto, S. Lattanzi, and M. Sozio, "Efficient densest subgraph computation in evolving graphs," in *WWW*, 2015.

[29] O. D. Balalau, F. Bonchi, T. Chan, F. Gullo, and M. Sozio, "Finding subgraphs with maximum total density and limited overlap," in *WSDM*, 2015.

[30] E. Galbrun, A. Gionis, and N. Tatti, "Top-k overlapping densest subgraphs," *Data Mining and Knowledge Discovery*, vol. 30, no. 5, pp. 1134–1165, 2016.

[31] B. Bahmani, R. Kumar, and S. Vassilvitskii, "Densest subgraph in streaming and mapreduce," *PVLDB*, vol. 5, no. 5, pp. 454–465, 2012.

[32] B. Bahmani, A. Goel, and K. Munagala, "Efficient primal-dual graph algorithms for mapreduce," in *WAW*, 2014.

[33] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos, "Copycatch: stopping group attacks by spotting lockstep behavior in social networks," in *WWW*, 2013.

[34] N. Shah, A. Beutel, B. Gallagher, and C. Faloutsos, "Spotting suspicious link behavior with fbox: An adversarial perspective," in *ICDM*, 2014.

[35] K. Shin, T. Eliassi-Rad, and C. Faloutsos, "Corescope: Graph mining using k-core analysis - patterns, anomalies and algorithms," in *ICDM*, 2016.

[36] J. Oh, K. Shin, E. E. Papalexakis, C. Faloutsos, and H. Yu, "S-hot: Scalable high-order tucker decomposition," in *WSDM*, 2017.

[37] K. Shin and U. Kang, "Distributed methods for high-dimensional and large-scale tensor factorization," in *ICDM*, 2014.

[38] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *KDD*, 2012, pp. 316–324.

[39] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *ICDE*, 2015, pp. 1047–1058.

[40] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," in *PKDD*, 2012.

[41] Y. Wang, H.-Y. Tung, A. J. Smola, and A. Anandkumar, "Fast and guaranteed tensor decomposition via sketching," in *NIPS*, 2015.

[42] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *PAKDD*, 2010.

[43] R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson, "Modeling dynamic behavior in large evolving graphs," in *WSDM*, 2013.

[44] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 626–688, 2015.

Kijung Shin is a Ph.D. student in Computer Science Department of Carnegie Mellon University. He received B.S. in Computer Science and Engineering from Seoul National University. His research interests include graph mining and scalable machine learning.

Bryan Hooi received the B.S. and M.S. degrees from Stanford University. He is currently working toward the Ph.D. degree in statistics and machine learning joint PhD Program at Carnegie Mellon University. His main research interests include high-dimensional inference, network analysis, biomedical, and social science applications.

Jisu Kim is a joint Ph.D. student in Statistics and Machine Learning at Carnegie Mellon University. He received B.S. in Mathematics at Seoul National University. His research interest lies in the inference on topological or geometrical features of data, such as topological data analysis.

Christos Faloutsos is a Professor at Carnegie Mellon University. He has received the Presidential Young Investigator Award by the National Science Foundation (1989), the Research Contributions Award in ICDM 2006, the SIGKDD Innovations Award (2010), 24 "best paper" awards (including 5 "test of time" awards), and four teaching awards. Eight of his advisees have attracted KDD or SCS dissertation awards, He is an ACM Fellow, he has served as a member of the executive committee of SIGKDD; he has published over 350 refereed articles, 17 book chapters and two monographs. He holds seven patents (and 2 pending), and he has given over 40 tutorials and over 20 invited distinguished lectures. His research interests include large-scale data mining with emphasis on graphs and time sequences; anomaly detection, tensors, and fractals.