



Detecting Group Anomalies in Tera-Scale Multi-Aspect Data via Dense-Subtensor Mining

Kijung Shin^{1*}, Bryan Hooi², Jisu Kim³ and Christos Faloutsos⁴

¹Graduate School of AI and School of Electrical Engineering, KAIST, Daejeon, South Korea, ²School of Computing and Institute of Data Science, National University of Singapore, Singapore, Singapore, ³DataShape, Inria Saclay, Palaiseau, France, ⁴School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, United States

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 suffer from 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 $1,561\times$ less memory and handles $1,000\times$ larger data (2.6TB), (2) Fast: up to $7\times$ 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.

Keywords: tensor, dense subtensor, anomaly detection, fraud detection, out-of-core algorithm, distributed algorithm

1 INTRODUCTION

Given 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 (Maruhashi et al., 2011; Jiang et al., 2015; Shin et al., 2018) 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

OPEN ACCESS

Edited by:

Meng Jiang,
University of Notre Dame,
United States

Reviewed by:

Kai Shu,
Illinois Institute of Technology,
United States

Kun Kuang,
Zhejiang University, China
Tong Zhao,
University of Notre Dame,
United States

*Correspondence:

Kijung Shin
kijungs@kaist.ac.kr

Specialty section:

This article was submitted to
Big Data Networks,
a section of the journal
Frontiers in Big Data

Received: 13 August 2020

Accepted: 17 December 2020

Published: 29 April 2021

Citation:

Shin K, Hooi B, Kim J and Faloutsos C
(2021) Detecting Group Anomalies in
Tera-Scale Multi-Aspect Data via
Dense-Subtensor Mining.
Front. Big Data 3:594302.
doi: 10.3389/fdata.2020.594302

(Maruhashi et al., 2011; Shin et al., 2018), retweet-boosting detection in online social networks (Jiang et al., 2015), bot-activity detection in Wikipedia (Shin et al., 2018), and genetics applications (Saha et al., 2010; Maruhashi et al., 2011).

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 (Jiang et al., 2015; Shin et al., 2018) outperform methods based on tensor decomposition, such as CP Decomposition and HOSVD (Maruhashi et al., 2011), in terms of accuracy and flexibility with regard to the choice of density metrics. Moreover, the latest search-based methods (Shin et al., 2018) provide 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.

1.1 Our Contributions

To overcome these limitations, we propose D-CUBE a dense-subtensor detection method for disk-resident tensors. D-CUBE works under the W-Stream model (Ruhl, 2003), 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 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 (Dean and Ghemawat, 2008), specifically its open source implementation HADOOP.

The main strengths of D-CUBE are as follows:

- **Memory Efficient:** D-CUBE requires up to $1,561\times$ less memory and successfully handles $1,000\times$ larger data ($2.6TB$) than its best competitors (**Figures 1A,B**).
- **Fast:** D-CUBE detects dense subtensors up to $7\times$ faster in real-world tensors and $12\times$ faster in synthetic tensors than its best competitors due to its near-linear scalability with all aspects of tensors (**Figure 1A**).
- **Provably Accurate:** D-CUBE provides a guarantee on the densities of the subtensors it finds (Theorem 3), and it shows similar or higher accuracy in dense-subtensor detection than its best competitors on real-world tensors (**Figure 1B**).
- **Effective:** D-CUBE successfully spotted network attacks from TCP dumps, and lockstep behavior in rating data, with the highest accuracy (**Figure 1C**).

Reproducibility: The code and data used in the paper are available at <http://dmlab.kaist.ac.kr/dcube>.

1.2 Related Work

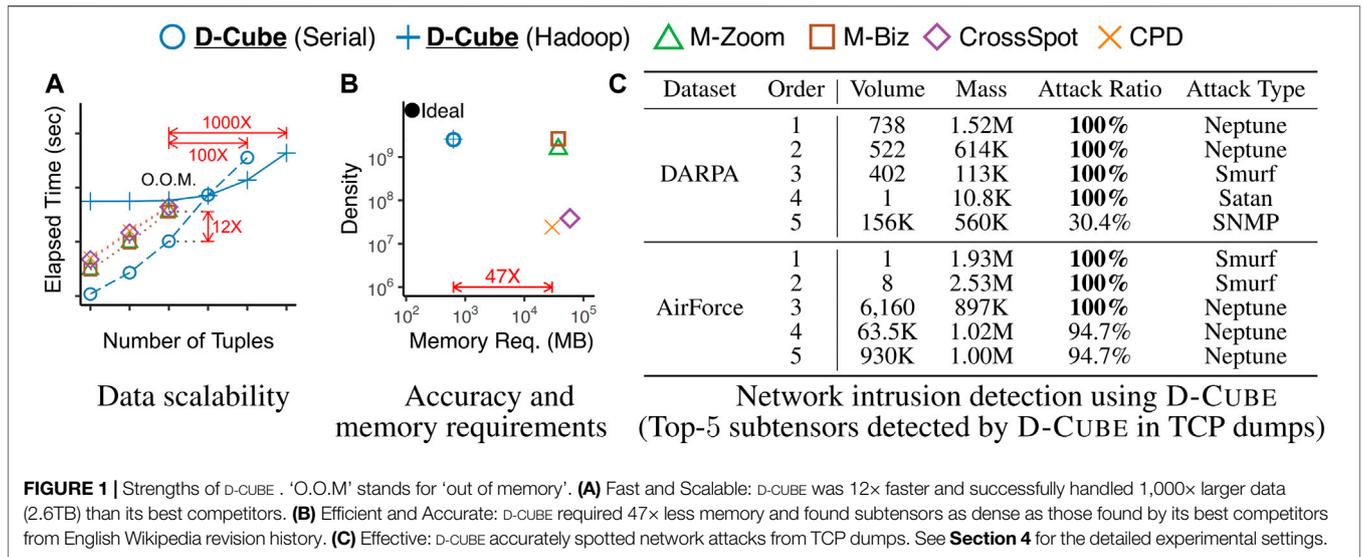
We discuss previous work on (a) dense-subgraph detection, (b) dense-subtensor detection, (c) large-scale tensor decomposition, and (d) other anomaly/fraud detection methods.

Dense Subgraph Detection. Dense-subgraph detection in graphs has been extensively studied in theory; see Lee et al. (2010) for a survey. Exact algorithms (Goldberg, 1984; Khuller and Saha, 2009) and approximate algorithms (Charikar, 2000; Khuller and Saha, 2009) have been proposed for finding subgraphs with maximum average degree. These have been extended for incorporating size restrictions (Andersen and Chellapilla, 2009), alternative metrics for denser subgraphs (Tsourakakis et al., 2013), evolving graphs (Epasto et al., 2015), subgraphs with limited overlap (Balalau et al., 2015; Galbrun et al., 2016), and streaming or distributed settings (Bahmani et al., 2012, 2014). Dense subgraph detection has been applied to fraud detection in social or review networks (Beutel et al., 2013; Jiang et al., 2014; Shah et al., 2014; Shin et al., 2016; Hooi et al., 2017).

Dense Subtensor Detection. Extending dense subgraph detection to tensors (Jiang et al., 2015; Shin et al., 2017a, 2018) incorporates additional dimensions, such as time, to identify dense regions of interest with greater accuracy and specificity. Jiang et al. (2015) proposed CROSSPOT, 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. Shin et al. (2018) proposed M-ZOOM, which starts from the entire tensor and only shrinks it by removing attributes one by one in a greedy way, improves CROSSPOT in terms of speed and approximation guarantees. M-BIZ, which was proposed in Shin et al. (2018), starts from the output of M-ZOOM and repeats adding or removing an attribute greedily until a local optimum is reached. Given a dynamic tensor, DENSEALERT and DENSESTREAM,

TABLE 1 | Comparison of D-CUBE and state-of-the-art dense-subtensor detection methods. ✓denotes ‘supported’.

	M-ZOOM and M-BIZ (Shin et al., 2018)	DENSESTREAM and DENSEALERT (Shin et al., 2017a)	CROSSPOT (Jiang et al., 2015)	MAF (Maruhashi et al., 2011)	FRAUDAR (Hooi et al., 2017)	D-CUBE (proposed)
High-order tensors	✓	✓	✓	✓		✓
Flexibility in density measures	✓		✓		✓	✓
Accuracy guarantees	✓	✓			✓	✓
Out-of-core computation						✓
Distributed computation						✓



which were proposed in Shin et al. (2017a), incrementally compute a single dense subtensor in it. CROSSSPOT, M-ZOOM, M-BIZ, and DENSESTREAM require all tuples of relations to be loaded into memory at once and to be randomly accessed, which limit their applicability to large-scale datasets. DENSEALERT maintains only the tuples created within a time window, and thus it can find a dense subtensor only within the window. Dense-subtensor detection in tensors has been found useful for detecting retweet boosting (Jiang et al., 2015), network attacks (Maruhashi et al., 2011; Shin et al., 2017a, 2018), bot activities (Shin et al., 2018), and vandalism on Wikipedia (Shin et al., 2017a), and also for genetics applications (Saha et al., 2010; Maruhashi et al., 2011).

Large-Scale Tensor Decomposition. Tensor decomposition such as HOSVD and CP decomposition (Kolda and Bader, 2009) can be used to spot dense subtensors, as shown in Maruhashi et al. (2011). Scalable algorithms for tensor decomposition have been developed, including disk-based algorithms (Shin and Kang, 2014; Oh et al., 2017), distributed algorithms (Kang et al., 2012; Shin and Kang, 2014; Jeon et al., 2015), and approximate algorithms based on sampling (Papalexakis et al., 2012) and count-min sketch (Wang et al., 2015). 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 (Akoglu et al., 2010), coreness (Shin et al., 2016), and behavior models (Rossi et al., 2013), have been used for anomaly and fraud detection in graphs. See Akoglu et al. (2015) for a survey.

1.3 Organization of the Paper

In **Section 2**, we provide notations and a formal problem definition. In **Section 3**, we propose D-CUBE, a disk-based

TABLE 2 | Table of symbols.

Symbol	Definition
$\mathcal{R}(A_1, \dots, A_N, X)$	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
$[x]$	$\{1, 2, \dots, x\}$

dense-subtensor detection method. In **Section 4**, we present experimental results and discuss them. In **Section 5**, we offer conclusions.

2 PRELIMINARIES AND PROBLEM DEFINITION

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 $[x] = \{1, 2, \dots, x\}$ for brevity. Let $\mathcal{R}(A_1, \dots, A_N, X)$ be a relation with N dimension attributes, denoted by A_1, \dots, 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 $\mathcal{R}_n = \{t[A_n] : t \in \mathcal{R}\}$ to denote the

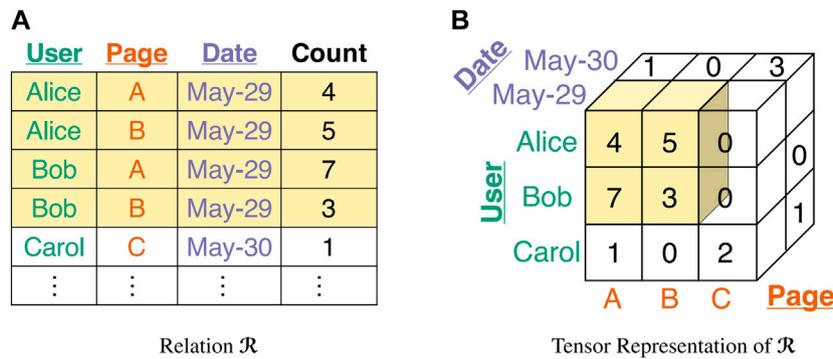


FIGURE 2 | Pictorial description of Example 1. **(A)** Relation \mathcal{R} where the colored tuples compose relation \mathcal{B} . **(B)** Tensor representation of \mathcal{R} where the relation \mathcal{B} forms a subtensor.

set of distinct values of A_n in \mathcal{R} . The relation \mathcal{R} is naturally represented as an N -way tensor of size $|\mathcal{R}_1| \times \dots \times |\mathcal{R}_N|$. 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 $\mathcal{B}(A_1, \dots, A_N, X) = \{t \in \mathcal{R} : \forall n \in [N], t[A_n] \in \mathcal{B}_n\}$, 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 $|\mathcal{B}_1| \times \dots \times |\mathcal{B}_N|$ in the tensor representation of \mathcal{R} , as in **Figure 2B**. 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 $\mathbf{R}(\text{user}, \text{page}, \text{date}, \text{count})$, where each tuple (u, p, d, c) in \mathbf{R} indicates that user u revised page p , c times, on date d . The first three attributes, $A_1 = \text{user}$, $A_2 = \text{page}$, and $A_3 = \text{date}$, are dimension attributes, and the other one, $X = \text{count}$, is the measure attribute. Let $\mathcal{B}_1 = \{\text{Alice}, \text{Bob}\}$, $\mathcal{B}_2 = \{A, B\}$, and $\mathcal{B}_3 = \{\text{May} - 29\}$. 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., $M_{\mathcal{B}(\text{Alice}, 1)}$) is 9, the number of revisions on A or B by exactly Alice on May-29. In the tensor representation, \mathcal{B} composes a subtensor in \mathbf{R} , as depicted in **Figure 2B**.

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 in Shin et al. (2018), are the extensions of density measures widely-used for graphs (Kannan and Vinay, 1999; Charikar, 2000).

DEFINITION 1 (Arithmetic Average Mass ρ_{ari}). The arithmetic average mass of a subtensor \mathcal{B} of a relation \mathbf{R} is defined as

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

DEFINITION 2 (Geometric Average Mass ρ_{geo}). The geometric average mass of a subtensor \mathcal{B} of a relation \mathbf{R} is defined as

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

Suspiciousness (Definition 3), which was used for detecting ‘retweet-boosting’ activities in Jiang et al. (2014), 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}). The suspiciousness of a subtensor \mathcal{B} of a relation \mathbf{R} is defined as

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

Entry Surplus (Definition 4) is the observed mass of \mathcal{B} subtracted by a 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, which was proposed in Tsourakakis et al. (2013) as a density metric for graphs.

DEFINITION 4 (Entry Surplus). The entry surplus of a subtensor \mathcal{B} of a relation \mathbf{R} is defined as

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

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 1.

Problem 1 (Large-scale Top- k Densest Subtensor Detection). **(1) Given:** a large-scale relation \mathbf{R} not fitting in memory, the number of subtensors k , and a density measure ρ , **(2) Find:** the top- k subtensors of \mathbf{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 (Goldberg, 1984; Khuller and Saha, 2009), 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 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.

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  $results \leftarrow \emptyset$   $\triangleright$  list of dense subtensors
4 for  $i \leftarrow 1..k$  do
5    $M_{\mathcal{R}} \leftarrow \sum_{t \in \mathcal{R}} t[X]$ 
6    $\{\mathcal{B}_n\}_{n=1}^N \leftarrow find\_one(\mathcal{R}, \{\mathcal{R}_n\}_{n=1}^N, M_{\mathcal{R}}, \rho, \theta)$   $\triangleright$  see Algorithm 2
7    $\mathcal{R} \leftarrow \{t \in \mathcal{R} : \exists n \in [N], t[A_n] \notin \mathcal{B}_n\}$   $\triangleright \mathcal{R} \leftarrow \mathcal{R} - \mathcal{B}$ 
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  $results$ 

```

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 $\mathcal{B} \leftarrow \text{copy}(\mathcal{R}), M_{\mathcal{B}} \leftarrow M_{\mathcal{R}}$ \triangleright initialize the subtensor \mathcal{B}
- 2 $\mathcal{B}_n \leftarrow \text{copy}(\mathcal{R}_n), \forall n \in [N]$
- 3 $\tilde{\rho} \leftarrow \rho(M_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n=1}^N, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N)$ $\triangleright \tilde{\rho}$: max ρ so far
- 4 $r, \tilde{r} \leftarrow 1$ $\triangleright r$: current order of attribute values, \tilde{r} : r with $\tilde{\rho}$
- 5 **while** $\exists n \in [N], \mathcal{B}_n \neq \emptyset$ \triangleright until all values are removed
- 6 compute $\{M_{\mathcal{B}(a,n)}\}_{a \in \mathcal{B}_n, n=1}^N$
- 7 $i \leftarrow \text{select_dimension}()$ \triangleright see Algorithms 3 and 4
- 8 $D_i \leftarrow \{a \in \mathcal{B}_i : M_{\mathcal{B}(a,i)} \leq \theta \frac{M_{\mathcal{B}}}{|\mathcal{B}_i|}\}$ $\triangleright D_i$: 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 $\rho' \leftarrow \rho(M_{\mathcal{B}}, \{|\mathcal{B}_n|\}_{n=1}^N, M_{\mathcal{R}}, \{|\mathcal{R}_n|\}_{n=1}^N)$ $\triangleright \rho'$: ρ when a is removed
- 13 $\text{order}(a, i) \leftarrow r, r \leftarrow r + 1$
- 14 **if** $\rho' > \tilde{\rho}$ **then**
- 15 $\tilde{\rho} \leftarrow \rho', \tilde{r} \leftarrow r$ \triangleright update max ρ so far
- 16 $\mathcal{B} \leftarrow \{t \in \mathcal{B} : t[A_i] \notin D_i\}$ \triangleright remove tuples
- 17 $\tilde{\mathcal{B}}_n \leftarrow \{a \in \mathcal{R}_n : \text{order}(a, n) \geq \tilde{r}\}, \forall n \in [N]$ \triangleright reconstruct
- 18 **return** $\{\tilde{\mathcal{B}}_n\}_{n=1}^N$

Note that this overall structure of D-CUBE is similar to that of M-ZOOM (Shin et al., 2018) except that 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 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 (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.

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 $|\mathcal{B}_n|$

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  $\tilde{\rho} \leftarrow -\infty, \tilde{i} \leftarrow 1$ 
2 for each dimension  $i \in [N]$  do
3   if  $\mathcal{B}_i \neq \emptyset$  then
4      $D_i \leftarrow \{a \in \mathcal{B}_i : M_{\mathcal{B}(a,i)} \leq \theta \frac{M_{\mathcal{B}}}{|\mathcal{B}_i|}\}$ 
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      $\rho' \leftarrow \rho(M'_{\mathcal{B}}, \{\mathcal{B}_n\}_{n \neq i} \cup \{\mathcal{B}'_i\}, M_{\mathcal{R}}, \{\mathcal{R}_n\}_{n=1}^N)$ 
8     if  $\rho' > \tilde{\rho}$  then
9        $\tilde{\rho} \leftarrow \rho', \tilde{i} \leftarrow i$ 
10 return  $\tilde{i}$ 

```

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 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. Then, we theoretically compare D-CUBE with M-ZOOM and M-BIZ (Shin et al., 2018).

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 $|\mathcal{B}_i \setminus D_i| > 0$, then

$$|\mathcal{B}_i \setminus D_i| < |\mathcal{B}_i| / \theta \quad (1)$$

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

$$\begin{aligned} M_{\mathcal{B}} &= \sum_{a \in \mathcal{B}_i \setminus D_i} M_{\mathcal{B}(a,i)} + \sum_{a \in D_i} M_{\mathcal{B}(a,i)} \\ &\geq \sum_{a \in \mathcal{B}_i \setminus D_i} M_{\mathcal{B}(a,i)} > |\mathcal{B}_i \setminus D_i| \cdot \theta \frac{M_{\mathcal{B}}}{|\mathcal{B}_i|}, \end{aligned}$$

where the last strict inequality is from the definition of D_i and that $|\mathcal{B}_i \setminus D_i| > 0$. 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 $\theta = O\left(e^{\left(\frac{N|L|}{L}\right)}\right)$, which is a weaker condition than $\theta = O(1)$, the worst-case time complexity of Algorithm 1 is*

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

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 in Algorithm 1 is $O(\sum_{n=1}^N |\mathcal{R}_n|)$.*

PROOF In Algorithm 1, $\{\{M_{\mathcal{B}(a,n)}\}_{a \in \mathcal{B}_n}\}_{n=1}^N$, $\{\mathcal{R}_n\}_{n=1}^N$, and $\{\mathcal{B}_n\}_{n=1}^N$ need to be loaded into memory at once. 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 in Dense-Subtensor Detection

We show that $D-CUBE$ gives the same accuracy guarantee with in-memory algorithms proposed in Shin et al. (2018), if we set θ to 1,

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

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 $\frac{M_{\mathcal{B}^*} - M_{\mathcal{B}^*(a,i)}}{\frac{1}{N} \left(\left(\sum_{n=1}^N |\mathcal{B}_n^*| \right) - 1 \right)} = \rho_{ari}$

$$(\mathcal{B}^* - \mathcal{B}^*(a, i), \mathcal{R}) \leq \rho_{ari}(\mathcal{B}^*, \mathcal{R}) = \frac{M_{\mathcal{B}^*}}{\frac{1}{N} \sum_{n=1}^N |\mathcal{B}_n^*|}, \text{ which reduces to}$$

$$M_{\mathcal{B}^*(a,i)} \geq \frac{1}{N} \rho_{ari}(\mathcal{B}^*, \mathcal{R}). \quad (3)$$

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 1:line:single:order1-1:line:single:order2 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 $\rho_{ari}(\mathcal{B}', \mathcal{R}) \geq \frac{M_{\mathcal{B}'(a,i)}}{\theta}$.

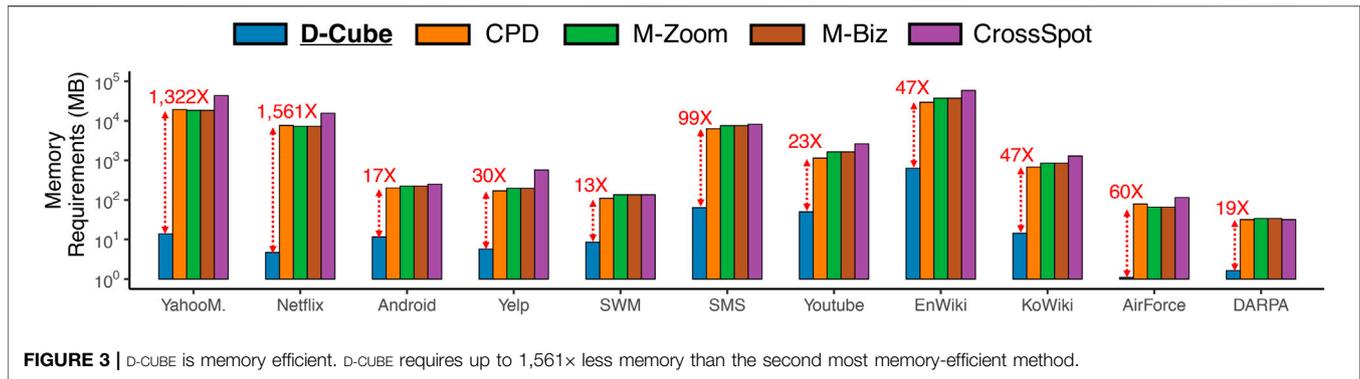


FIGURE 3 | D-CUBE is memory efficient. D-CUBE requires up to 1,561× less memory than the second most memory-efficient method.

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

3.2.3 Theoretical Comparison with M-ZOOM and M-BIZ (Shin et al., 2018)

While D-CUBE requires only $O(\sum_{n=1}^N |\mathcal{R}_n|)$ memory space (see Theorem 2), which does not depend on the number of tuples (i.e., $|\mathcal{R}|$), M-ZOOM and M-BIZ require additional $O(N|\mathcal{R}|)$ space for storing all tuples in main memory. The worst-case time complexity of D-CUBE is $O(kN^2|\mathcal{R}|\min(\log_{\theta}L, L))$ (see Theorem 1), and it is slightly higher than that of M-ZOOM, which is $O(kN|\mathcal{R}|\log L)$. Empirically, however, D-CUBE is up to 7× faster than M-ZOOM, as we show in Section 4. The main reason is that D-CUBE reads and writes tuples only sequentially, allowing efficient caching based on spatial locality. On the other hand, M-ZOOM requires tuples to be stored and accessed in hash tables, making efficient caching difficult.¹ The time complexity of M-BIZ depends on the number of iterations until reaching a local optimum, and there is no known upper bound on the number of iterations tighter than $O(2^{\sum_{n=1}^N |\mathcal{R}_n|})$. If ρ_{ari} is used, M-ZOOM and M-BIZ² give an approximation ratio of N, which is the approximation ratio of D-CUBE when θ is set to 1 (see Theorem 3).

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 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., $\langle t[A_1], \dots, t[A_N], t[X] \rangle$) 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., $\langle t[A_1], \dots, t[A_N], t[X] \rangle$) 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 $\langle (n, a), \text{sum}(\text{values}) \rangle$.

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

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., $\langle t[A_1], \dots, t[A_N], t[X] \rangle$) 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 $\langle (n, a), 0 \rangle$.

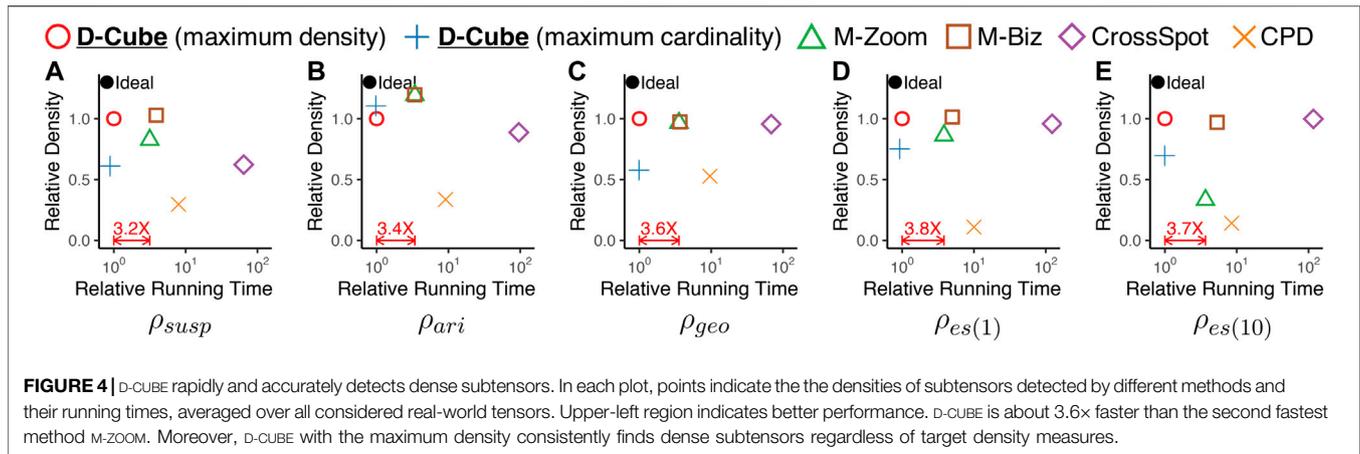
Each tuple $\langle (n, a), 0 \rangle$ of the final output indicates that a is a member of \mathcal{R}_n .

4 RESULTS AND DISCUSSION

We designed and conducted experiments to answer the following questions:

¹M-Zoom repeats retrieving all tuples with a given attribute value, and thus it requires storing and accessing tuples in hash tables for quick retrievals.

²We assume that M-Biz uses the outputs of M-Zoom as its initial states, as suggested in Shin et al. (2018).



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 Accuracy in Dense-subtensor Detection: How rapidly and accurately does D-CUBE identify dense subtensors? Does D-CUBE outperform its best competitors?

Q3. Scalability: Does D-CUBE scale linearly with all aspects of data? Does D-CUBE scale out?

Q4. Effectiveness in Anomaly Detection: Which anomalies 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 in dense-subtensor detection?

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 (user, item, timestamp, score, #ratings). Each tuple (u,i,t,s,r) indicates that user u gave item i score s, r times, at timestamp t. In the SWM dataset (Akoglu et al., 2013), the timestamps are in dates, and the items are entertaining software from a popular online software marketplace. In the Yelp dataset, the timestamps are in dates, and the items are businesses listed on Yelp, a review site. In the Android dataset (McAuley et al., 2015), the timestamps are hours, and the items are Android apps on Amazon, an online store. In

the Netflix dataset (Bennett and Lanning, 2007), the timestamps are in dates, and the items are movies listed on Netflix, a movie rental and streaming service. In the YahooM. dataset (Dror et al., 2012), 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 (user, page, timestamp, #revisions). Each tuple (u,p,t,r) indicates that user u revised page p, r times, at timestamp t (in hour) in Wikipedia, a crowd-sourcing online encyclopedia. In the KoWiki dataset, the pages are from Korean Wikipedia. In the EnWiki dataset, the pages are from English Wikipedia.

Temporal social networks. Temporal social networks are relations with schema (source, destination, timestamp, #interactions). Each tuple (s,d,t,i) indicates that user s interacts with user d, i times, at timestamp t. In the Youtube dataset (Mislove et al., 2007), the timestamps are in hours, and the interactions are becoming friends on Youtube, a video-sharing website. In the SMS dataset, the timestamps are in hours, and the interactions are sending text messages.

TCP Dumps. The DARPA dataset (Lippmann et al., 2000), collected by the Cyber Systems and Technology Group in 1998, is a relation with schema (source IP, destination IP, timestamp, #connections). Each tuple (s,d,t,c) indicates that c connections were made from IP s to IP d at timestamp t (in minutes). The AirForce dataset, used for KDD Cup. 1999, is a relation with schema (protocol, service, src bytes, dst bytes, flag, host count, srv count, #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.

TABLE 4 | D-CUBE spots network attacks and synchronized behavior fastest and most accurately from TCP dumps and rating datasets, respectively.

Datasets	AirForce		DARPA		Android		Yelp	
	Elapsed	AUROC	Elapsed	AUROC	Elapsed	Recall @	Elapsed	Recall @
	Time (s)		Time (s)		Time (s)	Top-10	Time (s)	Top-10
CPD	413.2	0.854	105.0	0.926	59.9	0.54	47.5	0.52
MAF	486.6	0.912	102.4	0.514	95.0	0.54	49.4	0.52
CROSSSPOT	575.5	0.924	132.2	0.923	71.3	0.54	56.7	0.52
M-ZOOM	27.7	0.975	22.7	0.923	28.4	0.70	17.7	0.30
M-BIZ	29.8	0.977	22.7	0.923	30.6	0.70	19.5	0.30
D-CUBE	15.6	0.987	9.1	0.930	7.0	0.90	4.9	0.60

TABLE 5 | D-Cube successfully detects spam reviews in the SWM dataset.

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

TABLE 6 | Summary of the dense subtensors that D-CUBE detects in the SWM, KoWiki, and EnWiki 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	9.98 M	1.71 M	7,931	Bot activities
	2	541 K	343 K	4,211	Bot activities
	3	23.5 M	973 K	3,395	Bot activities

TABLE 7 | D-Cube successfully spots bot activities in the 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

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

4.1.3 Implementations

We implemented the following dense-subtensor detection methods for 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 and M-BIZ (Shin et al., 2018): We used the open-source Java implementations of M-ZOOM and M-BIZ³. As suggested in Shin et al. (2018), we used the outputs of M-ZOOM as the initial states in M-BIZ.
- CROSSSPOT (Jiang et al., 2015): We used a Java implementation of the open-source implementation of CROSSSPOT⁴. 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 Shin et al. (2018).
- CPD (CP Decomposition): Let $\{A^{(n)}\}_{n=1}^N$ be the factor matrices obtained by CP Decomposition (Kolda and Bader (2009)). 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 the Tensor Toolbox⁵ for CP Decomposition.
- MAF (Maruhashi et al., 2011): We used the Tensor Toolbox for CP Decomposition, which MAF is largely based on.

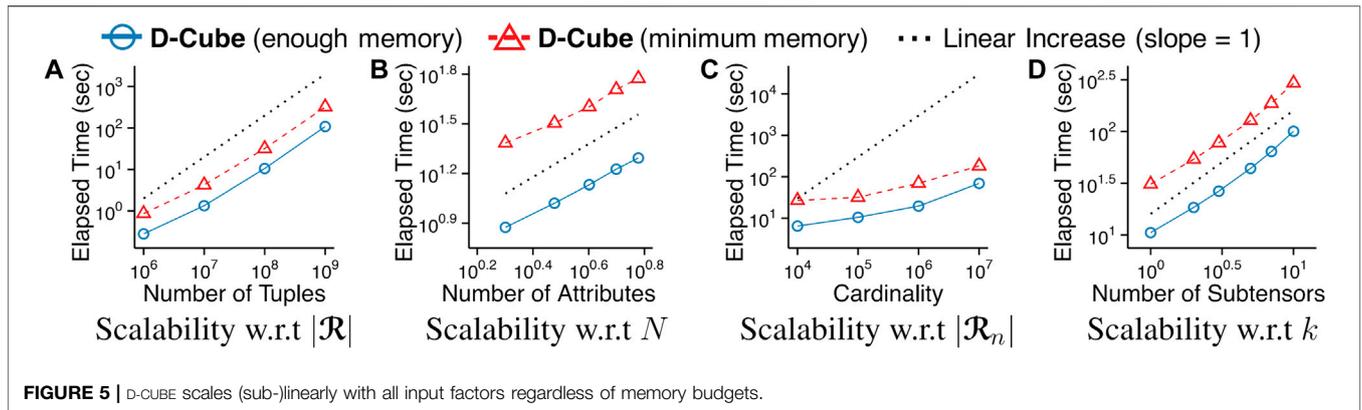
4.2 Q1. Memory Efficiency

We compare the amount of memory required by different methods for handling the real-world datasets. As seen in **Figure 3**, D-CUBE, which does not require tuples to be stored in memory, needed up to **1,561x less memory** than the second most memory-efficient method, which stores tuples in memory.

³<https://github.com/kijungs/mzoom>

⁴<https://github.com/mjiang89/CrossSpot>

⁵<https://www.sandia.gov/tgkolda/TensorToolbox/>



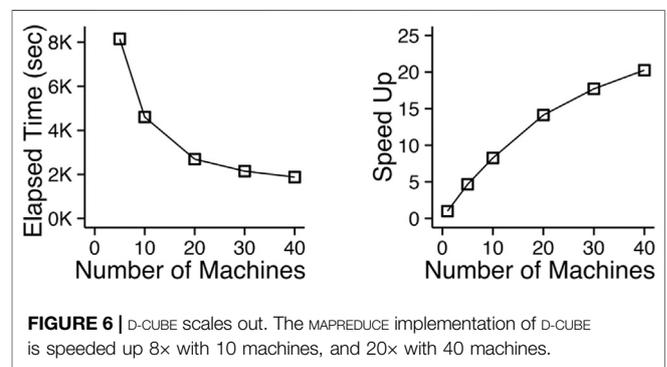
Due to its memory efficiency, D-CUBE successfully handled **1,000× larger data** than its competitors within a memory budget. We ran methods on 3-way synthetic tensors with different numbers of tuples (i.e., $|\mathcal{R}|$), with a memory budget of 16GB per machine. In every tensor, the cardinality 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 1A** 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 in Dense-Subtensor Detection

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 4 shows the results averaged over all considered datasets.⁶ The results in each data set can be found in the supplementary material. D-CUBE provided the best trade-off between speed and accuracy. 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 spotted high-density subtensors consistently regardless of target density measures**. Specifically, on average, D-CUBE with

⁶In each dataset, we measured the relative running time of each method (compared to the running time of D-Cube with the maximum density policy) and the relative density of detected dense subtensors (compared to the density of subtensors detected by D-Cube with the maximum density policy). Then, we averaged them over all considered datasets.



the maximum density policy was most accurate in dense-subtensor detection when ρ_{geo} and $\rho_{es(10)}$ were used; and it was second most accurate when ρ_{susp} and $\rho_{es(1)}$ were used. When ρ_{ari} was used, M-ZOOM, M-BIZ, and D-CUBE with the maximum cardinality policy were on average more accurate than D-CUBE with the maximum density policy. Although MAF does not appear in **Figure 4**, 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

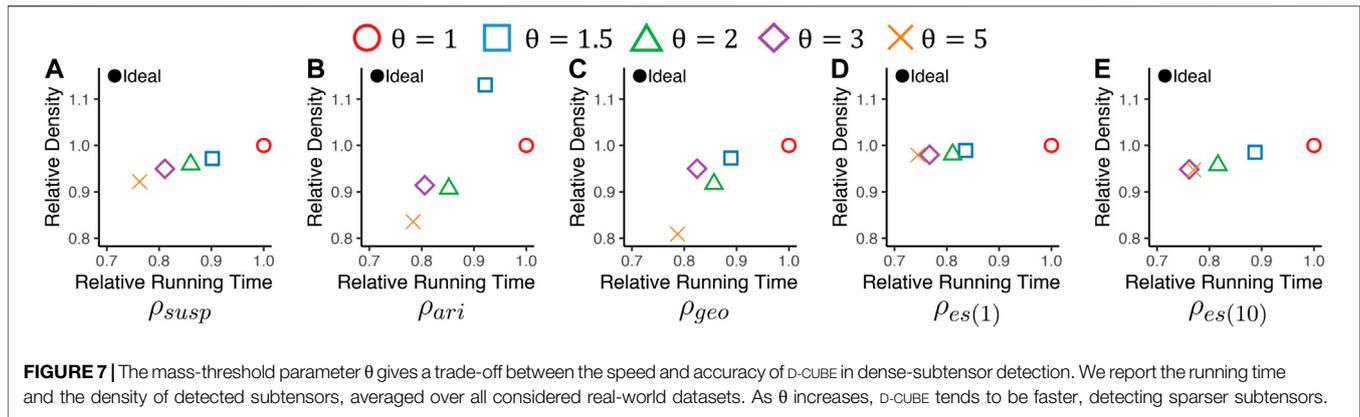


Figure 5) to barely meet the requirements although D-CUBE was up to $3\times$ faster in the former case.

We also evaluate the machine scalability of the MAPREDUCE 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.

4.5 Q4. Effectiveness in Anomaly Detection

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

4.5.1 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. The 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. The 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 1C 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 area under the ROC curve (AUROC) as seen in Table 4, where we report the AUROC when each method was used with the density measure giving the highest AUROC. In both datasets, using D-CUBE resulted in the highest AUROC.

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

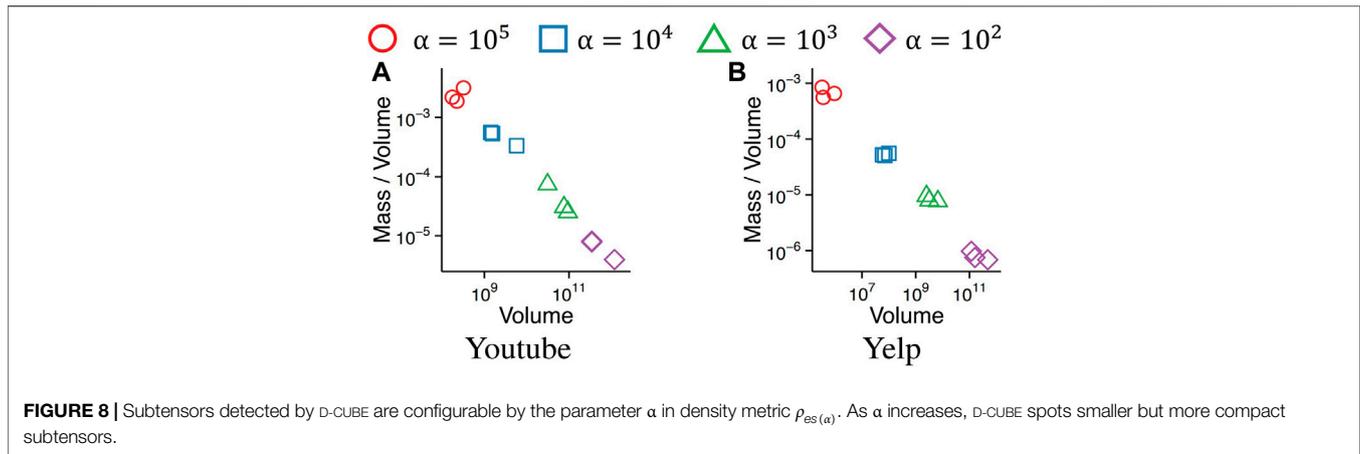
We injected 10 such random dense subtensors whose volumes varied from $15 \times 15 \times 1 \times 1$ to $60 \times 60 \times 1 \times 1$ in the Yelp and Android datasets. We compared the ratio of the injected subtensors detected by each dense-subtensor detection method. We considered each injected 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. That is, we measured the recall at top 10. We repeated this experiment 10 times, and the averaged results are summarized in Table 4. For each method, we report the results with the density measure giving the highest recall. In both datasets, D-CUBE detected a largest number of the injected subtensors. Especially, in the Android dataset, D-CUBE detected 9 out of the 10 injected subtensors, while the second best method detected only 7 injected subtensors on average.

4.5.3 Spam-Review Detection in Rating Data

D-CUBE successfully spotted spam reviews in the SWM dataset, which contains reviews from an online software marketplace. We modeled the 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 6 shows the statistics of the top-3 dense subtensors. Although ground-truth labels were not available, as the examples in Table 5 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.

4.5.4 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 6** gives the statistics of the top-3 dense subensors detected by D-CUBE (with $\rho = \rho_{ari}$ and the maximum cardinality policy) in the KoWiki dataset and by D-CUBE (with $\rho = \rho_{geo}$ and the maximum density policy) in the EnWiki dataset. All three subensors detected in the KoWiki dataset indicated edit wars. For example, the second subensor corresponded to an edit war where 4 users changed 4 pages, 1,011 times, within 5 h. On the other hand, all three subensors detected in the Enwiki dataset indicated bot activities. For example, the third subensor corresponded to 3 bots which edited 1,067 pages 973,747 times. The users composing the top-5 dense subensors in the EnWiki dataset are listed in **Table 7**. Notice that all of them are bots.

4.6 Q5. Effects of Parameter θ on Speed and Accuracy in Dense-Subensor Detection

We investigate the effects of the mass-threshold parameter θ on the speed and accuracy of D-CUBE in dense-subensor detection. We used the serial version of D-CUBE with a memory budget of 16GB, and we measured the relative density of detected subensors and its running time, as in **Section 4.3**. **Figure 7** shows the results averaged over all considered datasets. Different θ values provided a trade-off between speed and accuracy in dense-subensor detection. Specifically, increasing θ tended to make D-CUBE faster but also make it detect sparser subensors. This tendency is consistent with our theoretical analyses (Theorems 1–3 in **Section 3.2**). The sensitivity of the dense-subensor detection 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 Subensors Detected by D-CUBE

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

5 CONCLUSION

In this work, we propose D-CUBE, a disk-based dense-subensor 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 subensors it finds. Moreover, we propose a 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 1,561× 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 guaranteeing the best approximation ratio (Theorem 3) in dense-subensor detection and spotting the densest subensors 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://dmlab.kaist.ac.kr/dcub>

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: <http://dmlab.kaist.ac.kr/dcub>.

AUTHOR CONTRIBUTIONS

KS, BH, and CF contributed to conception and design of the study. KS performed the experiments. JK performed the mathematical analysis. KS wrote the first draft of the manuscript. KS and BH wrote sections of the manuscript. All authors contributed to manuscript revision, read, and approved the submitted version.

FUNDING

This research was supported by National Research Foundation of Korea (NRF) Grant funded by the Korea government (MSIT) (No. NRF-2020R1C1C1008296) and Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)). This research was also supported by the National Science Foundation under Grant Nos. CNS-1314632 and IIS-1408924. This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

- Akoglu, L., Chandy, R., and Faloutsos, C. (2013). Opinion fraud detection in online reviews by network effects. *ICWSM*.
- Akoglu, L., McGlohon, M., and Faloutsos, C. (2010). Oddball: spotting anomalies in weighted graphs. *PAKDD*.
- Akoglu, L., Tong, H., and Koutra, D. (2015). Graph based anomaly detection and description: a survey. *Data Mining Knowl. Discov.* 29, 626–688. doi:10.1201/b15352-15
- Andersen, R., and Chellapilla, K. (2009). Finding dense subgraphs with size bounds. *WAW*.
- Bahmani, B., Goel, A., and Munagala, K. (2014). Efficient primal-dual graph algorithms for mapreduce. *WAW*.
- Bahmani, B., Kumar, R., and Vassilvitskii, S. (2012). Densest subgraph in streaming and mapreduce. *PVLDB* 5, 454–465. doi:10.14778/2140436.2140442
- Balalau, O. D., Bonchi, F., Chan, T., Gullo, F., and Sozio, M. (2015). Finding subgraphs with maximum total density and limited overlap. *WSDM*.
- Bennett, J., and Lanning, S. (2007). The netflix prize. *KDD Cup*.
- Beutel, A., Xu, W., Guruswami, V., Palow, C., and Faloutsos, C. (2013). Copycatch: stopping group attacks by spotting lockstep behavior in social networks. *WWW*.
- Charikar, M. (2000). Greedy approximation algorithms for finding dense components in a graph. *APPROX*.
- Dean, J., and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM* 51, 107–113. doi:10.21276/ijre.2018.5.5.4
- Dror, G., Koenigstein, N., Koren, Y., and Weimer, M. (2012). The yahoo! music dataset and kdd-cup'11. *KDD Cup*.
- Epasto, A., Lattanzi, S., and Sozio, M. (2015). Efficient densest subgraph computation in evolving graphs. *WWW*.
- Galbrun, E., Gionis, A., and Tatti, N. (2016). Top-k overlapping densest subgraphs. *Data Mining Knowl. Discov.* 30, 1134–1165. doi:10.1007/s10618-016-0464-z
- Goldberg, A. V. (1984). Finding a maximum density subgraph. Technical Report.

ACKNOWLEDGMENTS

The content of the manuscript has been presented in part at the 10th ACM International Conference on Web Search and Data Mining (Shin et al., 2017b). In this extended version, we refined D-CUBE with a new parameter θ , and we proved that the time complexity of D-CUBE is significantly improved with the refinement (Lemma 1 and Theorem 1). We also proved that, for N-way tensors, D-CUBE gives an θN -approximation guarantee for Problem 1 (Theorem 3). Additionally, we considered an extra density measure (Definition 3) and an extra competitor (i.e., M-BIZ); and we applied D-CUBE to three more real-world datasets (i.e., KoWiki, EnWiki, and SWM) and successfully detected edit wars, bot activities, and spam reviews (Tables 5–7). Lastly, we conducted experiments showing the effects of parameters θ and α on the speed and accuracy of D-CUBE in dense-subtensor detection (Figures 7 and 8). Most of this work was also included in the PhD thesis of the first author (KS).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fdata.2020.594302/full#supplementary-material>

- Hooi, B., Shin, K., Song, H. A., Beutel, A., Shah, N., and Faloutsos, C. (2017). Graph-based fraud detection in the face of camouflage. *ACM Trans. Knowl. Discov. Data* 11, 44. doi:10.1145/3056563
- Jeon, I., Papalexakis, E. E., Kang, U., and Faloutsos, C. (2015). Hatn2: billion-scale tensor decompositions. *ICDE*, 1047–1058.
- Jiang, M., Beutel, A., Cui, P., Hooi, B., Yang, S., and Faloutsos, C. (2015). A general suspiciousness metric for dense blocks in multimodal data. *ICDM*.
- Jiang, M., Cui, P., Beutel, A., Faloutsos, C., and Yang, S. (2014). Catchsync: catching synchronized behavior in large directed graphs. *KDD*.
- Kang, U., Papalexakis, E., Harpale, A., and Faloutsos, C. (2012). Gigatensor: scaling tensor analysis up by 100 times—algorithms and discoveries. *KDD*.
- Kannan, R., and Vinay, V. (1999). Analyzing the structure of large graphs. Technical Report.
- Khuller, S., and Saha, B. (2009). On finding dense subgraphs. *ICALP*, 597–608.
- Kolda, T. G., and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Rev.* 51, 455–500. doi:10.2172/755101
- Lee, V. E., Ruan, N., Jin, R., and Aggarwal, C. (2010). A survey of algorithms for dense subgraph discovery. *Managing and Mining Graph Data*, 303–336.
- Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., et al. (2000). Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. *DISSEX*.
- Maruhashi, K., Guo, F., and Faloutsos, C. (2011). Multiaspectforensics: pattern mining on large-scale heterogeneous networks with tensor analysis. *ASONAM*.
- McAuley, J., Pandey, R., and Leskovec, J. (2015). Inferring networks of substitutable and complementary products. *KDD*.
- Mislove, A., Marcon, M., Gummadi, K. P., Druschel, P., and Bhattacharjee, B. (2007). Measurement and analysis of online social networks. *IMC*.
- Oh, J., Shin, K., Papalexakis, E. E., Faloutsos, C., Yu, H., and S-hot (2017). Scalable high-order tucker decomposition. *WSDM*.
- Papalexakis, E. E., Faloutsos, C., and Sidiropoulos, N. D. (2012). Parcube: sparse parallelizable tensor decompositions. *PKDD*.
- Rossi, R. A., Gallagher, B., Neville, J., and Henderson, K. (2013). Modeling dynamic behavior in large evolving graphs. *WSDM*.

- Ruhl, J. M. (2003). Efficient algorithms for new computational models. Ph.D. thesis, Massachusetts Institute of Technology.
- Saha, B., Hoch, A., Khuller, S., Raschid, L., and Zhang, X. N. (2010). Dense subgraphs with restrictions and applications to gene annotation graphs. RECOMB.
- Shah, N., Beutel, A., Gallagher, B., and Faloutsos, C. (2014). Spotting suspicious link behavior with fbox: an adversarial perspective. ICDM.
- Shin, K., Eliassi-Rad, T., and Faloutsos, C. (2016). Corescope: graph mining using k-core analysis—patterns, anomalies and algorithms. ICDM.
- Shin, K., Hooi, B., and Faloutsos, C. (2018). Fast, accurate, and flexible algorithms for dense subtensor mining. *ACM Trans. Knowledge Discov. Data* 12, 28. doi:10.1145/3154414.1-2830
- Shin, K., Hooi, B., Kim, J., and Faloutsos, C. (2017b). D-cube: dense-block detection in terabyte-scale tensors. WSDM.
- Shin, K., Hooi, B., Kim, J., and Faloutsos, C. (2017a). Densealert: incremental dense-subtensor detection in tensor streams. KDD.
- Shin, K., and Kang, U. (2014). Distributed methods for high-dimensional and large-scale tensor factorization. ICDM.
- Tsourakakis, C., Bonchi, F., Gionis, A., Gullo, F., and Tsiarli, M. (2013). Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. KDD.
- Wang, Y., Tung, H. Y., Smola, A. J., and Anandkumar, A. (2015). Fast and guaranteed tensor decomposition via sketching. NIPS.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Copyright © 2021 Shin, Hooi, Kim and Faloutsos. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.