Boolean Tensor Factorizations

Pauli Miettinen
Max Planck Institute for Informatics
Saarbrücken, Germany
pauli.miettinen@mpi-inf.mpg.de

***Abstract*—Tensors are multi-way generalizations of matrices, and similarly to matrices, they can also be factorized, that is, represented (approximately) as a product of factors. These factors are typicaly either all matrices or a mixture of matrices and tensors. With the widespread adoption of matrix factorization techniques in data mining, also tensor factroziations have started to gain attention.**

In this paper we study the Boolean tensor factorizations. We assume that the data is binary multi-way data, and we want to factorize it to binary factors using Boolean arithmetic (i.e. defining that $1+1=1$). Boolean tensor factorizations are, therefore, natural generalization of the Boolean matrix factorizations. We will study the theory of Boolean tensor factorizations and show that at least some of the benefits Boolean matrix factorizations have over normal matrix factorizations carry over to the tensor data. We will also present algorithms for Boolean variations of CP and Tucker decompositions, the two most-common types of tensor factorizations. With experimentation done with synthetic and real-world data, we show that Boolean tensor factorizations are a viable alternative when the data is naturally binary.

***Keywords*-Tensor factorization; CP factorization; Tucker factorization; Boolean tensor factorization; Boolean matrix factorization**

I. Introduction

With the matrix factorizations (or decompositions, the two terms are used interchangeably here) becoming widespread techniques in data mining, many researchers have started to focus on their multi-way generalizations – tensor factorizations. Tensor factorizations have proven to be powerful. Yet, they are also much more complex, both from the computational complexity's and interpretability's point of view. Independently, the Boolean matrix factorizations have attained growing research interest in data mining. They are shown to have such desirable properties as interpretability [1] and sparsity [2], although with cost of increased computational complexity. It is therefore natural to ask could Boolean tensor factorization, the generalization of Boolean matrix factorization in multi-way data, sustain the interpretability and sparsity of Boolean matrix factorizations and still be practical to compute. This papers aims to provide answers to that question.

When the data is binary, multiway data, Boolean tensor decompositions are intuitively appealing approach. An example of such data are subject–relation–object tuples, such as ('BarackObama' 'isA' 'PresidentOfTheUSA'). Expressing this data as a matrix would lose some level of dependencies: either that subject–object pairs can hold for different relations, that subject–relation pair can hold for different subjects, or that relation–object pair can hold for different subjects. Only three-way tensor can capture all these dependencies simultaneously. Using the Boolean decompositions, on the other hand, should help on interpreting the results (as they can be considered as sets of subjects, objects, and relations, instead of arbitrary vectors) and provide sparser results, among other things.

This paper will start by giving the definitions of Boolean tensor rank and CP and Tucker decomposition, after which we will we provide theoretical study of some aspects of Boolean tensor rank and CP decompositions. These results provide insights on the similarities and dissimilarities between, on one hand, Boolean and normal tensor factorizations, and on the other hand, Boolean tensor and Boolean matrix factorizations. For example, we generalize the sparsity result of Boolean matrix factorizations to Boolean tensor factorizations, showing that sparse binary tensors always have sparse Boolean factorizations.

We also propose two algorithms, one for Boolean CP decomposition and one for Boolean Tucker decomposition. We evaluate these algorithms with synthetic and real-world data, comparing them to state-of-the-art real-valued decomposition algorithms.

For the sake of clarity, most of this paper concentrates on 3-way tensors. The results are, however, straightforward to generalize to N-way tensors.

II. Background and Basic Definitions

We start by giving the basic notation used with tensors. With that notation we can define the normal tensor rank and normal CP and Tucker decompositions and summarize some of the important properties of them. We then define the Boolean tensor rank and CP and Tucker decomposition. Proving any properties of them is postponed to the next section. The section ends with a short review on how Boolean tensor factorizations relate to other data mining techniques.

A. Notation

Throughout this paper vectors are indicated as bold-face lower-case letters ($\boldsymbol{v}$), matices as bold-face upper-case letters ($\boldsymbol{M}$), and tensors as bold-face upper-case calligraphic letters ($\boldsymbol{\mathcal{T}}$). We present the notation for 3-way tensors, but it can

be extended to N-way tensors in a straight forward way. Element (i,j,k) of a 3-way tensor $\boldsymbol{\mathcal{X}}$ is denoted either as x_{ijk} or as $(\boldsymbol{\mathcal{X}})_{ijk}$. A colon in a subscript denotes taking that mode entirely; for example, if $\boldsymbol{X}$ is a matrix, $\boldsymbol{x}_{i:}$ denotes the ith row of $\boldsymbol{X}$ (for a shorthand, we use $\boldsymbol{x}_j$ to denote the jth column of $\boldsymbol{X}$). For a 3-way tensor $\boldsymbol{\mathcal{X}}$, $\boldsymbol{x}_{:jk}$ is the (j,k) *mode-1 (column) fiber*, $\boldsymbol{x}_{i:k}$ the (i,k) *mode-2 (row) fiber*, and $\boldsymbol{x}_{ij:}$ the (i,j) *mode-3 (tube) fiber*. Furthermore, $\boldsymbol{X}_{::k}$ is the kth *frontal slice* of $\boldsymbol{\mathcal{X}}$. We use $\boldsymbol{X}_k$ as a shorthand for the kth frontal slice.

A tensor can be *unfold* into a matrix by arranging its fibers as columns of a matrix. If mode-n fibers are used as the columns, the process is called *mode-n matricization*. The mode-n matricization of a tensor $\boldsymbol{\mathcal{X}}$ is denoted as $\boldsymbol{X}_{(n)}$.

For a tensor $\boldsymbol{\mathcal{X}}$, the number of non-zero elements in it is denoted by $|\boldsymbol{\mathcal{X}}|$. This and other tensor notation presented below is extended to matrices and vectors in a natural way. The Frobenius norm of a 3-way tensor $\boldsymbol{\mathcal{X}}$, $\|\boldsymbol{\mathcal{X}}\|$, is defined as $\sqrt{\sum_{i,j,k} x_{ijk}^2}$. If $\boldsymbol{\mathcal{X}}$ is binary, i.e. takes values only from $\{0,1\}$, $|\boldsymbol{\mathcal{X}}| = \|\boldsymbol{\mathcal{X}}\|^2$.

We use $\boxtimes$ to denote vector outer product in N modes. That is, if $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are vectors of length n, m, and l, respectively, $\boldsymbol{\mathcal{X}} = \boldsymbol{a} \boxtimes \boldsymbol{b} \boxtimes \boldsymbol{c}$ is a n-by-m-by-l tensor with $x_{ijk} = a_i b_j c_k$.

The *tensor sum* of two n-by-m-by-l tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ is just the element-wise sum, $(\boldsymbol{\mathcal{X}} + \boldsymbol{\mathcal{Y}})_{ijk} = x_{ijk} + y_{ijk}$. The *Boolean tensor sum* of binary tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ is defined as $(\boldsymbol{\mathcal{X}} \vee \boldsymbol{\mathcal{Y}})_{ijk} = x_{ijk} \vee y_{ijk}$.

If matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ are binary and $\boldsymbol{X}$ has r columns and $\boldsymbol{Y}$ has r rows (i.e. r is their inner dimension), their *Boolean matrix product*, $\boldsymbol{X} \circ \boldsymbol{Y}$, is defined as $(\boldsymbol{X} \circ \boldsymbol{Y})_{ij} = \bigvee_{k=1}^{r} x_{ik} y_{kj}$. The *Boolean matrix rank* of a binary matrix $\boldsymbol{A}$ is the least r such that there exists a pair of binary matrices $(\boldsymbol{X}, \boldsymbol{Y})$ of inner dimension r with $\boldsymbol{A} = \boldsymbol{X} \circ \boldsymbol{Y}$.

Let $\boldsymbol{X}$ be n_1-by-m_1 and $\boldsymbol{Y}$ be n_2-by-m_2 matrix (binary or otherwise). Their *Kronecker (matrix) product*, $\boldsymbol{X} \otimes \boldsymbol{Y}$, is the $n_1 n_2$-by-$m_1 m_2$ matrix defined by

$$\boldsymbol{X} \otimes \boldsymbol{Y} = \begin{bmatrix} x_{11}\boldsymbol{Y} & x_{12}\boldsymbol{Y} & \cdots & x_{1m_1}\boldsymbol{Y} \\ x_{21}\boldsymbol{Y} & x_{22}\boldsymbol{Y} & \cdots & x_{2m_1}\boldsymbol{Y} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n_1 1}\boldsymbol{Y} & x_{n_1 2}\boldsymbol{Y} & \cdots & x_{n_1 m_1}\boldsymbol{Y} \end{bmatrix}.$$

The *Khatri–Rao (matrix) product* of $\boldsymbol{X}$ and $\boldsymbol{Y}$ is defined as 'column-wise Kronecker'. That is, $\boldsymbol{X}$ and $\boldsymbol{Y}$ must have same number of columns ($m_1 = m_2 = m$), and their Khatri–Rao product $\boldsymbol{X} \odot \boldsymbol{Y}$ is the $n_1 n_2$-by-m matrix defined as

$$\boldsymbol{X} \odot \boldsymbol{Y} = \begin{bmatrix} x_{11}\boldsymbol{y}_1 & \cdots & x_{1m}\boldsymbol{y}_m \\ x_{21}\boldsymbol{y}_1 & \cdots & x_{2m}\boldsymbol{y}_m \\ \vdots & \ddots & \vdots \\ x_{n_1 1}\boldsymbol{y}_1 & \cdots & x_{n_1 m}\boldsymbol{y}_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{Y}\delta_1(\boldsymbol{X}) \\ \boldsymbol{Y}\delta_2(\boldsymbol{X}) \\ \vdots \\ \boldsymbol{Y}\delta_{n_1}(\boldsymbol{X}) \end{bmatrix},$$

where $\delta_i(\boldsymbol{X})$ is a diagonal matrix with the ith row of $\boldsymbol{X}$ in its diagonal. Notice that if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are binary, so are $\boldsymbol{X} \otimes \boldsymbol{Y}$ and $\boldsymbol{X} \odot \boldsymbol{Y}$.

B. Ranks and Factorizations

Tensor Rank and CP Decomposition: Just as we can define the matrix rank as the least number of rank-1 matrices needed to be summed together to obtain the original matrix, we can define the tensor rank as the least number of rank-1 tensors whose sum equals the original tensor. A *rank*-1 tensor is, still analogously to the matrix case, a tensor that is an outer product of vectors (N-way tensor requires N vectors). Thus we have

Definition 1 (Tensor rank)**.** The *rank* of a 3-way tensor $\boldsymbol{\mathcal{X}}$, $\operatorname{rank}(\boldsymbol{\mathcal{X}})$, is the least integer r such that there exist r rank-1 tensors whose sum is $\boldsymbol{\mathcal{X}}$, i.e.

$$\boldsymbol{\mathcal{X}} = \sum_{i=1}^{r} \boldsymbol{a}_i \boxtimes \boldsymbol{b}_i \boxtimes \boldsymbol{c}_i \,. \tag{1}$$

The definition of tensor rank gives rise to the first tensor factorization, the CP factorization[1] for rank-r approximation:

Definition 2 (The CP tensor decomposition)**.** Given an n-by-m-by-l tensor $\boldsymbol{\mathcal{X}}$ and an integer r, find matrices $\boldsymbol{A}$ (n-by-r), $\boldsymbol{B}$ (m-by-r), and $\boldsymbol{C}$ (l-by-r) such that they minimize

$$\left\| \boldsymbol{\mathcal{X}} - \sum_{i=1}^{r} \boldsymbol{a}_i \boxtimes \boldsymbol{b}_i \boxtimes \boldsymbol{c}_i \right\|. \tag{2}$$

Importantly, we can write the CP decomposition in the terms of matrices using unfolding (see e.g. [5]):

$$\begin{aligned} \boldsymbol{X}_{(1)} &= \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T \\ \boldsymbol{X}_{(2)} &= \boldsymbol{B}(\boldsymbol{C} \odot \boldsymbol{A})^T \\ \boldsymbol{X}_{(3)} &= \boldsymbol{C}(\boldsymbol{B} \odot \boldsymbol{A})^T \,. \end{aligned} \tag{3}$$

Tensor rank has some properties that differ from the matrix rank (for more detailed discussion, see e.g. [5]). For example, a real-valued tensor can have different rank over the real numbers and over the complex numbers – this can never happen with matrices. More importantly to the data mining applications, the maximum tensor rank of n-by-m-by-l tensor $\boldsymbol{\mathcal{X}}$ can be much more than $\min\{n,m,l\}$, which would be analogous to the matrix case, and there exists tensors with rank greater than $\max\{n,m,l\}$ (see [5]). Indeed, the best known upper bound is [5]

$$\operatorname{rank}(\boldsymbol{\mathcal{X}}) \leq \min\{nm, nl, ml\}. \tag{4}$$

Tensors can also be *degenerate* meaning that they can be approximated arbitrarily well by tensors of lower rank. Consequently, the factors of the best rank-$(k-1)$ approximation are not necessarily the factors of the best rank-k approximation. This is again in contrast with matrices, where the Eckart–Young theorem shows both cases impossible.

Finally, computing the tensor rank is NP-hard [6].

[1]The name is short for two names given to the same decomposition: CANDECOMP [3] and PARAFAC [4].

The Tucker Decomposition: The CP decomposition is a natural generalization of the matrix decompositions, but by no means the only one. Another common tensor decomposition is the *Tucker decomposition* [7] that generalizes the CP decomposition by allowing each mode have a different rank and using a core tensor to combine the factor matrices into one tensor. The Tucker decomposition is defined as follows.

Definition 3 (The Tucker tensor decomposition)**.** Given an n-by-m-by-l tensor $\boldsymbol{\mathcal{X}}$ and three integers r_1, r_2, and r_3, find r_1-by-r_2-by-r_3 core tensor $\boldsymbol{\mathcal{G}}$ and factor matrices $\boldsymbol{A}$ (n-by-r_1), $\boldsymbol{B}$ (m-by-r_2), and $\boldsymbol{C}$ (l-by-r_3) such that they minimize

$$\left\| \boldsymbol{\mathcal{X}} - \sum_{\alpha=1}^{r_1} \sum_{\beta=1}^{r_2} \sum_{\gamma=1}^{r_3} g_{\alpha\beta\gamma}\, \boldsymbol{a}_\alpha \boxtimes \boldsymbol{b}_\beta \boxtimes \boldsymbol{c}_\gamma \right\| . \tag{5}$$

Notice that we obtain the CP decomposition by requiring that $r_1 = r_2 = r_3 = r$ and that $\boldsymbol{\mathcal{G}}$ has 1s at its hyper-diagonal and 0s elsewhere. As with the CP decomposition, we can express Tucker decomposition with matrices using mode-n matricization [5]:

$$\begin{aligned} \boldsymbol{X}_{(1)} &= \boldsymbol{A}\boldsymbol{G}_{(1)}(\boldsymbol{C} \otimes \boldsymbol{B})^T \\ \boldsymbol{X}_{(2)} &= \boldsymbol{B}\boldsymbol{G}_{(2)}(\boldsymbol{C} \otimes \boldsymbol{A})^T \\ \boldsymbol{X}_{(3)} &= \boldsymbol{C}\boldsymbol{G}_{(3)}(\boldsymbol{B} \otimes \boldsymbol{A})^T . \end{aligned} \tag{6}$$

The Boolean Tensor Rank and Decompositions: We can now turn to the actual topic of this paper: the Boolean tensor decompositions. The Boolean versions of tensor rank and CP and Tucker decompositions are rather straight forward to define given their normal counterparts. One only needs to change the summation to $1 + 1 = 1$. Notice that this does not change the definition of a rank-1 tensor (or vector outer product). Thence, 3-way Boolean rank-1 tensor is a tensor that is an outer product of three binary vectors.

Definition 4 (Boolean tensor rank)**.** The *Boolean rank* of a 3-way binary tensor $\boldsymbol{\mathcal{X}}$, $\operatorname{rank}_B(\boldsymbol{\mathcal{X}})$, is the least integer r such that there exist r rank-1 binary tensors with

$$\boldsymbol{\mathcal{X}} = \bigvee_{i=1}^{r} \boldsymbol{a}_i \boxtimes \boldsymbol{b}_i \boxtimes \boldsymbol{c}_i . \tag{7}$$

The Boolean CP decomposition follows analogously, but notice the change of error measure from the Frobenius to the sum of differences.

Definition 5 (The Boolean CP tensor decomposition)**.** Given an n-by-m-by-l binary tensor $\boldsymbol{\mathcal{X}}$ and an integer r, find binary matrices $\boldsymbol{A}$ (n-by-r), $\boldsymbol{B}$ (m-by-r), and $\boldsymbol{C}$ (l-by-r) such that they minimize

$$\left| \boldsymbol{\mathcal{X}} - \bigvee_{i=1}^{r} \boldsymbol{a}_i \boxtimes \boldsymbol{b}_i \boxtimes \boldsymbol{c}_i \right| . \tag{8}$$

The unfolded versions also look similar to (3), though the matrix product have to be changed to the Boolean matrix product (recall that the Khatri–Rao product is closed under the Boolean algebra and thus does not need to be changed):

$$\begin{aligned} \boldsymbol{X}_{(1)} &= \boldsymbol{A} \circ (\boldsymbol{C} \odot \boldsymbol{B})^T \\ \boldsymbol{X}_{(2)} &= \boldsymbol{B} \circ (\boldsymbol{C} \odot \boldsymbol{A})^T \\ \boldsymbol{X}_{(3)} &= \boldsymbol{C} \circ (\boldsymbol{B} \odot \boldsymbol{A})^T . \end{aligned} \tag{9}$$

And finally, the Boolean Tucker decomposition is defined as follows.

Definition 6 (The Boolean Tucker tensor decomposition)**.** Given an n-by-m-by-l binary tensor $\boldsymbol{\mathcal{X}}$ and three integers r_1, r_2, and r_3, find binary r_1-by-r_2-by-r_3 core tensor $\boldsymbol{\mathcal{G}}$ and binary factor matrices $\boldsymbol{A}$ (n-by-r_1), $\boldsymbol{B}$ (m-by-r_2), and $\boldsymbol{C}$ (l-by-r_3) such that they minimize

$$\left| \boldsymbol{\mathcal{X}} - \bigvee_{\alpha=1}^{r_1} \bigvee_{\beta=1}^{r_2} \bigvee_{\gamma=1}^{r_3} g_{\alpha\beta\gamma}\, \boldsymbol{a}_\alpha \boxtimes \boldsymbol{b}_\beta \boxtimes \boldsymbol{c}_\gamma \right| . \tag{10}$$

The unfolded versions are:

$$\begin{aligned} \boldsymbol{X}_{(1)} &= \boldsymbol{A} \circ \boldsymbol{G}_{(1)} \circ (\boldsymbol{C} \otimes \boldsymbol{B})^T \\ \boldsymbol{X}_{(2)} &= \boldsymbol{B} \circ \boldsymbol{G}_{(2)} \circ (\boldsymbol{C} \otimes \boldsymbol{A})^T \\ \boldsymbol{X}_{(3)} &= \boldsymbol{C} \circ \boldsymbol{G}_{(3)} \circ (\boldsymbol{B} \otimes \boldsymbol{A})^T . \end{aligned} \tag{11}$$

C. Relations to Other Data Mining Methods

The approach taken is this paper is to consider the Boolean tensor factorizations as a variation of normal tensor factorizations. While this approach has its obvious benefits, it should be noted that it is by no means the only possible approach. *Tiling* a database [8] refers to the task of covering all 1s of a binary matrix using few[2] frequent itemsets. The Boolean matrix factorization can be seen as a generalization of this task, each rank-1 binary matrix defining a 'tile'. The difference is that tiling does not allow any 0s to be presented as 1s, whereas the Boolean matrix factorization allows this type of errors.

Analogously, we can consider Boolean CP tensor decomposition as an N-way lossy tiling, generalizing the N-way tiling [9]. Again, finding the least number of tiles needed to express the data set exactly is equivalent to finding the Boolean tensor rank, and minimizing the number of uncovered 1s in N-way tiling is related to minimizing the error in Boolean CP tensor decomposition (with the latter allowing the covering of 0s with 1s).

III. Theory

The definitions of Boolean tensor decompositions and rank are akin to their normal counterparts. But do they behave similarly? Or do they behave similarly to their Boolean matrix counterparts? In this section we will study some of the more important properties of Boolean tensor rank and CP and

[2]When the goal is to cover all 1s and minimize the number of tiles, it is equivalent to computing the Boolean rank [2]; when the number of tiles is given and the goal is to minimize the number of uncovered 1s, the problem is more akin to standard Boolean matrix factorization.

Tucker decompositions and try to answer those questions from the theory's point of view.

A. Boolean Tensor Rank

Recall from the previous section that the (normal) tensor rank behaved very differently to the matrix rank. One major difference is that the rank can be much bigger than the dimensions of the tensor. Here, the Boolean tensor rank behaves analogously. In fact, we can prove bounds to the Boolean tensor rank analogous to those known for the normal tensor rank. First is the lower-bound for maximum rank.

Proposition 1. *There exists binary n-by-m-by-l tensors that have Boolean rank higher than* $\max\{n, m, l\}$.

Proof: Let $n = m = l$. There are 2^{n^3} different binary n-by-n-by-n tensors, but only 2^{3n^2} triples of binary n-by-n factor matrices. Therefore, there has to be a tensor that does not have a rank-n factorization. ∎

Proving the upper bound, however, is much more complex. It will follow the proof of the similar claim for real-valued tensors, and therefore we present only a sketch of it.

Theorem 2. *For any n-by-m-by-l binary tensor $\boldsymbol{\mathcal{X}}$ we have*

$$\operatorname{rank}_B(\boldsymbol{\mathcal{X}}) \leq \min\{nm, nl, ml\}\,. \tag{12}$$

Proof sketch: Let $ml = \min\{nm, nl, ml\}$ (other cases are analogous). We show how to make a rank-ml exact Boolean CP factorization of $\boldsymbol{\mathcal{X}}$ thereby proving the claim. Specifically, we construct n-by-ml binary matrix $\boldsymbol{A}$, m-by-ml binary matrix $\boldsymbol{B}$, and l-by-ml binary matrix $\boldsymbol{C}$ such that the three unfolded equations of (9) hold.

The factor matrices are

$$\boldsymbol{A} = \boldsymbol{X}_{(1)}$$
$$\boldsymbol{B} = \underbrace{[\boldsymbol{I}_m\, \boldsymbol{I}_m \cdots \boldsymbol{I}_m]}_{l \text{ times}}$$
$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{J}_{1\times m} & 0 & \cdots & 0 \\ 0 & \boldsymbol{J}_{1\times m} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{J}_{1\times m} \end{bmatrix} \; l \text{ rows},$$

where $\boldsymbol{I}_m$ is the m-by-m identity matrix and $\boldsymbol{J}_{1\times m}$ is m-dimensional row vector full of 1s. To show that the first equation of (9) holds, we need to show that $\boldsymbol{C} \odot \boldsymbol{B} = \boldsymbol{I}_{ml}$. This is easy to see by remembering that

$$\boldsymbol{C} \odot \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}\delta_1(\boldsymbol{C}) \\ \boldsymbol{B}\delta_2(\boldsymbol{C}) \\ \vdots \\ \boldsymbol{B}\delta_l(\boldsymbol{C}) \end{bmatrix}.$$

That the other two equations of (9) also hold can be checked similarly. While the proofs are conceptually rather straight forward, they require very clumsy notation, and are therefore omitted. ∎

The real-valued tensor can be degenerate, but this is not an issue with Boolean tensor rank: arbitrarily close approximation is impossible with discrete errors. On the other hand, similarly to normal CP decomposition, there is no (known) reason why the factors of the least-error Boolean rank-$(r-1)$ decomposition should be the factors of the least-error rank-r decomposition.

One of the highlight features of Boolean matrix rank is that it can be considerably smaller than the normal matrix rank (in fact, a logarithm of the normal rank [10]). Establishing such results (or proving that they do not hold) with the tensor ranks is an important topic for future research.

B. Sparsity of Decompositions

In many applications of matrix decompositions, sparsity of the factor matrices is of importance. Recently, Miettinen [2] proved that Boolean matrix factorizations behave very well in this respect: any Boolean rank-k binary matrix has rank-k decomposition where the factor matrices have at most twice the number of 1s compared to the 1s in the original matrix. No such result is known for non-Boolean matrix or tensor factorizations, but for Boolean CP decomposition we can prove a generalized version of the result in [2].

Theorem 3. *Every N-way binary tensor $\boldsymbol{\mathcal{X}}$ with $\operatorname{rank}_B(\boldsymbol{\mathcal{X}}) = r$ has a rank-r Boolean CP decomposition with factor matrices $\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \ldots, \boldsymbol{A}^{(N)}$ such that*

$$\sum_{i=1}^{N} \left|\boldsymbol{A}^{(i)}\right| \leq N\,|\boldsymbol{\mathcal{X}}|\,. \tag{13}$$

Proof: The proof, which follows that of [2], is by induction on the tensor rank of $\boldsymbol{\mathcal{X}}$. In the base case, $\operatorname{rank}_B(\boldsymbol{\mathcal{X}}) = 1$. In this case $\boldsymbol{\mathcal{X}} = \boldsymbol{a}^{(1)} \boxtimes \boldsymbol{a}^{(2)} \boxtimes \cdots \boxtimes \boldsymbol{a}^{(N)}$, and hence $|\boldsymbol{\mathcal{X}}| = \prod_{i=1}^{N} \left|\boldsymbol{a}^{(i)}\right| \geq N^{-1} \sum_{i=1}^{N} \left|\boldsymbol{a}^{(i)}\right|$ (no $\left|\boldsymbol{a}^{(i)}\right| = 0$ or else $\boldsymbol{\mathcal{X}}$ is empty).

Assume then that the claim holds for tensors of rank $r-1$ and let $\operatorname{rank}_B(\boldsymbol{\mathcal{X}}) = r$. Let

$$\boldsymbol{\mathcal{Y}}^{(j)} = \boldsymbol{a}_j^{(1)} \boxtimes \boldsymbol{a}_j^{(2)} \boxtimes \cdots \boxtimes \boldsymbol{a}_j^{(N)}$$

for all $j = 1, \ldots, r$ and let $\boldsymbol{\mathcal{Y}}^{(\setminus r)} = \bigvee_{j=1}^{r-1} \boldsymbol{\mathcal{Y}}^{(j)}$ such that $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{Y}}^{(\setminus r)} \vee \boldsymbol{\mathcal{Y}}^{(r)}$. As $\boldsymbol{\mathcal{Y}}^{(\setminus r)}$ is of rank-$(r-1)$, by induction assumption we have that

$$N\left|\boldsymbol{\mathcal{Y}}^{(\setminus r)}\right| \geq \sum_{i=1}^{N} \sum_{j=1}^{r-1} \left|\boldsymbol{a}_j^{(i)}\right|\,. \tag{14}$$

Now, consider a vector $\boldsymbol{a}_r^{(1)}$. Let k be such that $a_{kr}^{(1)} = 1$. If for every $i_2, i_3, \ldots, i_N$ such that $y^{(r)}_{ki_1i_2i_3\ldots i_N} = 1$ it also holds that $y^{(\setminus r)}_{ki_1i_2i_3\ldots i_N} = 1$, we can set $a^{(1)}_{kr}$ (and consequently $y^{(r)}_{ki_1i_2i_3\ldots i_N}$) to 0 without changing $\boldsymbol{\mathcal{Y}}^{(\setminus r)} \vee \boldsymbol{\mathcal{Y}}^{(r)}$. The same holds for $\boldsymbol{a}_r^{(2)}, \ldots, \boldsymbol{a}_r^{(N)}$. It follows that each 1 in any of $\boldsymbol{a}_r^{(i)}$ must contribute at least one new 1 to $\boldsymbol{\mathcal{X}}$ compared to

$\boldsymbol{\mathcal{Y}}^{(\setminus r)}$ (though different vectors $\boldsymbol{a}_r^{(i)}$ can contribute to the same 1). Therefore we have

$$\left|\boldsymbol{\mathcal{Y}}^{(\setminus r)}\right| + \max_{i=1,\ldots,N}\left\{\left|\boldsymbol{a}_r^{(i)}\right|\right\} \leq |\boldsymbol{\mathcal{X}}|\,. \tag{15}$$

Combining the results above, we get

$$\begin{aligned}\sum_{i=1}^{N}\left|\boldsymbol{A}^{(i)}\right| &= \sum_{i=1}^{N}\sum_{j=1}^{r}\left|\boldsymbol{a}_j^{(i)}\right| \\ &\leq N\left|\boldsymbol{\mathcal{Y}}^{(\setminus r)}\right| + \left|\boldsymbol{\mathcal{Y}}^{(r)}\right| \\ &= N\left|\boldsymbol{\mathcal{Y}}^{(\setminus r)}\right| + \prod_{i=1}^{N}\left|\boldsymbol{a}_r^{(i)}\right| \\ &\leq N\left(\left|\boldsymbol{\mathcal{Y}}^{(\setminus r)}\right| + \max_{i=1,\ldots,N}\left\{\left|\boldsymbol{a}_r^{(i)}\right|\right\}\right) \\ &\leq N\,|\boldsymbol{\mathcal{X}}|\,,\end{aligned} \tag{16}$$

where the first inequality follows from (14) and the last from (15). ∎

This result is tight: consider an N-way tensor $\boldsymbol{\mathcal{X}}$ with $|\boldsymbol{\mathcal{X}}| = 1$. None of the factor matrices $\boldsymbol{A}^{(i)}$ can be empty, or else their outer product is empty. Therefore any exact factorization of $\boldsymbol{\mathcal{X}}$ requires at least N 1s in the factor matrices. The result is stated for *exact* decompositions, but there is an easy corollary to bound the density of the factor matrices in terms of the original matrix and the induced error.

Corollary 4. *Let $\boldsymbol{\mathcal{X}}$ be a binary N-way tensor with $\operatorname{rank}_B(\boldsymbol{\mathcal{X}}) = r$ and let $(\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \ldots, \boldsymbol{A}^{(N)})$ be a rank-p approximate CP decomposition of $\boldsymbol{\mathcal{X}}$. Let $\oplus$ be the element-wise exclusive or and $\boldsymbol{\mathcal{E}} = \boldsymbol{\mathcal{X}} \oplus \left(\bigvee_{i=1}^{p} \boldsymbol{a}_i^{(1)} \boxtimes \boldsymbol{a}_i^{(2)} \boxtimes \cdots \boxtimes \boldsymbol{a}_i^{(N)}\right)$ the error tensor. Then*

$$\sum_{i=1}^{N}\left|\boldsymbol{A}^{(i)}\right| \leq N(|\boldsymbol{\mathcal{X}}| + |\boldsymbol{\mathcal{E}}|)\,. \tag{17}$$

Proof: By definition $\bigvee_{i=1}^{p} \boldsymbol{a}_i^{(1)} \boxtimes \boldsymbol{a}_i^{(2)} \boxtimes \cdots \boxtimes \boldsymbol{a}_i^{(N)} = \boldsymbol{\mathcal{X}} \oplus \boldsymbol{\mathcal{E}}$, which then must be rank-p tensor, and therefore $\sum_{i=1}^{N}\left|\boldsymbol{A}^{(i)}\right| \leq N\,|\boldsymbol{\mathcal{X}} \oplus \boldsymbol{\mathcal{E}}| \leq N(|\boldsymbol{\mathcal{X}}| + |\boldsymbol{\mathcal{E}}|)$. ∎

Notice that these are strictly existential results: they do not tell us how to find such sparse decompositions, just that they exist.

C. Computational Complexity

As mentioned in Section II-B, finding the tensor rank is NP-hard. Similar results hold for Boolean tensor rank, as well as computing the optimal Boolean CP and Tucker decompositions for given ranks. Unlike with normal decompositions, however, here the results follow trivially from the fact that the corresponding matrix decompositions are NP-hard.

Proposition 5. *Given a binary tensor $\boldsymbol{\mathcal{X}}$, (1) finding the least-error Boolean CP decomposition of the given rank r is* NP*-hard, (2) deciding the* $\operatorname{rank}_B(\boldsymbol{\mathcal{X}})$ *is* NP*-hard, and (3) finding the least-error Boolean Tucker decomposition for given parameters $(r_1, r_2, \ldots, r_N)$ is* NP*-hard.*

Proof: For (1) and (2), we notice that finding the least-error Boolean matrix factorization and deciding the Boolean rank of a binary matrices are specializations of their tensor counterparts. As these matrix problems are already NP-hard, their tensor generalizations are too. For the Tucker decomposition, we have the following result. Let $\boldsymbol{\mathcal{X}}$ be 2-way, i.e. matrix $\boldsymbol{X}$, and let $r_1 = r_2 = r$. Now the Tucker decomposition returns three binary matrices, $\boldsymbol{G}$, $\boldsymbol{A}$, and $\boldsymbol{B}$. If this is the least-error Boolean Tucker decomposition of $\boldsymbol{X}$, then $(\boldsymbol{A} \circ \boldsymbol{G}, \boldsymbol{B})$ is the least-error rank-r Boolean matrix factorization of $\boldsymbol{X}$. ∎

These reductions also show that the inapproximability results for Boolean matrix decompositions [1] carry over to the tensor cases at least to some extent.

IV. Algorithms

The definitions of Boolean tensor decompositions are of little use in data mining unless we have algorithms for finding those decompositions. The results from the previous section show that we cannot hope to find polynomial-time optimal algorithms, and even algorithms with provable approximation guarantees are unlikely. We therefore utilize heuristic methods.

A. The `BCP_ALS` Algorithm

The alternating least-squares projection heuristic is the workhorse of algorithms for normal CP decomposition [5]. We utilize similar technique with the Boolean CP decomposition, although with few changes.

Our first 'workhorse' is the `Asso` algorithm for Boolean matrix decompositions by Miettinen et al. [11] (notice, though, that the `Asso` algorithm is not required by the algorithm – any other method for solving the Boolean matrix decomposition would do). It is used to initialize the three factor matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ (again, we give the algorithms for 3-way tensors, but they can be easily generalized to N-way tensors). This is done instead of the more common random initialization as the Boolean iterative update (described below) typically converges very quickly to local optimum, and is thus unable to escape bad initial solutions. Naturally, as this initialization is deterministic, if it yields bad initial solution that the iterative update is unable to escape, the algorithm will return bad solution and cannot use randomness to avoid it.

To use the `Asso` algorithm, we give the mode-n matricizations of $\boldsymbol{\mathcal{X}}$ as an input to it. The three left-hand-side factor matrices (from the three different matricizations) constitute the initial solution. We then move to the iterative update phase. For this we use the unfolded format (9), fixing two factor matrices while updating the third. There is a problem, however: given binary matrices $\boldsymbol{X}_{(1)}$ and $\boldsymbol{Y} = (\boldsymbol{C} \odot \boldsymbol{B})^T$, finding the binary matrix $\boldsymbol{A}$ such that $\left|\boldsymbol{X}_{(1)} - \boldsymbol{A} \circ \boldsymbol{Y}\right|$ is

Algorithm 1 An algorithm for the Boolean CP decomposition

Input: A 3-way binary tensor $\mathcal{X}$, rank r.
Output: Binary factor matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$.

function BCP_ALS($\mathcal{X}, r$)
  $\boldsymbol{A} \leftarrow \texttt{Asso}(\boldsymbol{X}_{(1)}, r)$
  $\boldsymbol{B} \leftarrow \texttt{Asso}(\boldsymbol{X}_{(2)}, r)$
  $\boldsymbol{C} \leftarrow \texttt{Asso}(\boldsymbol{X}_{(3)}, r)$
  repeat
    $\boldsymbol{A} \leftarrow \texttt{UpdateFactor}(\boldsymbol{X}_{(1)}, \boldsymbol{A}, (\boldsymbol{C} \odot \boldsymbol{B})^T)$
    $\boldsymbol{B} \leftarrow \texttt{UpdateFactor}(\boldsymbol{X}_{(2)}, \boldsymbol{B}, (\boldsymbol{C} \odot \boldsymbol{A})^T)$
    $\boldsymbol{C} \leftarrow \texttt{UpdateFactor}(\boldsymbol{X}_{(3)}, \boldsymbol{C}, (\boldsymbol{B} \odot \boldsymbol{A})^T)$
  until converged
  return $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$
end function

minimized is known as the Basis Usage problem, and is NP-hard to even approximate well [1]. We therefore have to update the factors using a greedy heuristic. We apply the following technique from [11]: consider each row of $\boldsymbol{X}_{(1)}$ separately, and let c be a function such that $c(i) = 0$ if $(\boldsymbol{a} \circ \boldsymbol{Y})_i = 1$, where $\boldsymbol{a}$ is the corresponding row of $\boldsymbol{A}$ before updates. Now define function cover as

$$\mathsf{cover}(\boldsymbol{x}, \boldsymbol{z}, c) = \sum_i \big(c(i)[x_i = 1] - c(i)[x_i = 0]\big)[z_i = 1]\,. \tag{18}$$

We can now update the values of $\boldsymbol{a}$ such that $\mathsf{cover}(\boldsymbol{x}, \boldsymbol{a} \circ \boldsymbol{Y}, c)$ is maximized in polynomial time. Doing this for every row of $\boldsymbol{A}$ we obtain the updated factor matrix (with $\boldsymbol{B}$ and $\boldsymbol{C}$ fixed, we can update each row of $\boldsymbol{A}$ independently). This procedure is called UpdateFactor. The complete algorithm, called BCP_ALS, is presented as Algorithm 1.

With n-by-m-by-l input tensor, the BCP_ALS algorithm converges to local optimum in at most nml steps, as each step is guaranteed to reduce error by at least 1.

B. The BTucker_ALS Algorithm

Solving the Boolean Tucker decomposition is more involved because of the core tensor $\mathcal{G}$. Every element of $\mathcal{G}$ can potentially effect every element of the approximate representation, as the (i, j, k) element of the representation is

$$\bigvee_{\alpha=1}^{r_1} \bigvee_{\beta=1}^{r_2} \bigvee_{\gamma=1}^{r_3} g_{\alpha\beta\gamma} a_{i\alpha} b_{j\beta} c_{k\gamma}\,.$$

Therefore, a change in a single element of $\mathcal{G}$ can change the product completely. This, however, is a very hypothetical situation. First, if $a_{i\alpha} b_{j\beta} c_{k\gamma} = 0$, the value of $g_{\alpha\beta\gamma}$ is irrelevant – the product will be zero in any case. Second, if there is (α, β, γ) for which $g_{\alpha\beta\gamma} a_{i\alpha} b_{j\beta} c_{k\gamma} = 1$, the other values of $\mathcal{G}$ do not have any effect – the element (i, j, k) will be 1. These two observations help us to compute the gain we can obtain by flipping an element of $\mathcal{G}$. The whole procedure is given in Algorithm 2, where $[x] = \{1, 2, \ldots, x\}$.

The UpdateG algorithm is used to initialize $\mathcal{G}$ (after factor matrices are initialized, and setting initial $\mathcal{G}$ to all-zero) and later to update it in the iterative update process. The full BTucker_ALS algorithm is presented in Algorithm 3.

Also BTucker_ALS is guaranteed to converge in nml steps for n-by-ml input tensor as each step will reduce the error at least by 1.

Algorithm 2 An algorithm for updating the core tensor

Input: A 3-way n-by-m-by-l binary tensor $\mathcal{X}$, (r_1, r_2, r_3) Boolean Tucker decomposition $(\mathcal{G}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ of $\mathcal{X}$.
Output: Updated binary core tensor $\mathcal{G}$

function UpdateG($\mathcal{X}, \mathcal{G}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$))
  $\widetilde{\mathcal{X}} \leftarrow \bigvee_{\alpha=1}^{r_1} \bigvee_{\beta=1}^{r_2} \bigvee_{\gamma=1}^{r_3} g_{\alpha\beta\gamma}\, \boldsymbol{a}_\alpha \boxtimes \boldsymbol{b}_\beta \boxtimes \boldsymbol{c}_\gamma$
  for $(\alpha, \beta, \gamma) \in [r_1] \times [r_2] \times [r_3]$ **do**
    $gain \leftarrow 0$
    for all (i, j, k) such that $a_{i\alpha} b_{j\beta} c_{k\gamma} = 1$ **do**
      if $g_{\alpha\beta\gamma} = 0$ and $\tilde{x}_{ijk} = 0$ **then**
        $gain \leftarrow gain + x_{ijk}$
      else if $g_{\alpha\beta\gamma} = 1$ and $\tilde{x}_{ijk} = 1$ **then**
        if not exists $(\alpha', \beta', \gamma')$ such that $g_{\alpha'\beta'\gamma'} a_{i\alpha'} b_{j\beta'} c_{k\gamma'} = 1$ **then**
          $gain \leftarrow gain + (1 - x_{ijk})$
        end if
      end if
    end for
    if $gain > 0$ **then**
      $g_{\alpha\beta\gamma} \leftarrow 1 - g_{\alpha\beta\gamma}$
    end if
  end for
  return $\mathcal{G}$
end function

Algorithm 3 An algorithm for the Boolean Tucker decomposition

Input: A 3-way binary tensor $\mathcal{X}$, ranks (r_1, r_2, r_3).
Output: Boolean Tucker decomposition $(\mathcal{G}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$

function BTucker_ALS($\mathcal{X}, r_1, r_2, r_3$)
  $\boldsymbol{A} \leftarrow \texttt{Asso}(\boldsymbol{X}_{(1)}, r)$
  $\boldsymbol{B} \leftarrow \texttt{Asso}(\boldsymbol{X}_{(2)}, r)$
  $\boldsymbol{C} \leftarrow \texttt{Asso}(\boldsymbol{X}_{(3)}, r)$
  $\mathcal{G} \leftarrow \texttt{UpdateG}(\mathcal{X}, \boldsymbol{0}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
  repeat
    $\boldsymbol{A} \leftarrow \texttt{UpdateFactor}(\boldsymbol{X}_{(1)}, \boldsymbol{A}, \boldsymbol{G}_{(1)} \circ (\boldsymbol{C} \otimes \boldsymbol{B})^T)$
    $\boldsymbol{B} \leftarrow \texttt{UpdateFactor}(\boldsymbol{X}_{(2)}, \boldsymbol{B}, \boldsymbol{G}_{(1)} \circ (\boldsymbol{C} \otimes \boldsymbol{A})^T)$
    $\boldsymbol{C} \leftarrow \texttt{UpdateFactor}(\boldsymbol{X}_{(3)}, \boldsymbol{C}, \boldsymbol{G}_{(1)} \circ (\boldsymbol{B} \otimes \boldsymbol{A})^T)$
    $\mathcal{G} \leftarrow \texttt{UpdateG}(\mathcal{X}, \mathcal{G}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
  until converged
  return $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$
end function

V. Experimental Evaluation

As the algorithms presented in the previous section are based on heuristics, it is important to study their behaviour with extensive experimentation. For this, we use both synthetic and real-world data focus being in the former, as synthetic data offers better control over the data characteristics and thus allows more systematic study of the algorithms' properties.

We compare the proposed algorithms to three algorithms

for normal CP and Tucker decompositions. Two of them, the CP_ALS and Tucker_ALS algorithms are standard alternating least-squares-based algorithms (see [5]), while the third, CP_OPT by Acar et al. [12], is based on optimization approach.

Comparing real-valued and Boolean methods is not straight forward. Finding the optimum Boolean CP decomposition is not the same thing as finding the optimum normal CP decomposition. Care must be taken to not make too far-reaching assumptions from these results, as there are two types of results: the results the algorithms report, and the optimum results for the algorithms' tasks. The former we know, the latter we do not. Therefore, if one algorithm performs badly compared to others with particular data, it might not necessarily mean that the algorithm itself has problems – it may well be that the algorithm returned the optimum for the task it tried to optimize.

Nevertheless, comparing real-valued and Boolean algorithms can provide some insights to the algorithms' behaviour. When synthetic data is made so that it is decomposable via Boolean methods, we should assume the Boolean methods to be comparable (or even better, as the case might be) to the real-valued ones. To facilitate the comparisons, we report two kinds of errors from the real-valued methods. First is the squared Frobenius. This is the error measure the algorithms aim to minimize, but it is somewhat unfair to the Boolean methods as squaring the element-wise error shrinks the cost of small errors, typically yielding dense real-valued decompositions that make little error in each element. The other error we report is the sum of absolute differences (or L_1 error). This error penalizes more equally for all kinds of errors, but as the real-valued methods do not try to optimize it, it is unfair to them. The subscripted F in algorithm's name denotes squared Frobenius while subscripted B denotes L_1 error.

The algorithms were implemented in Matlab and C using Matlab Tensor Toolbox [13]. The three algorithms for the normal CP and Tucker decompositions were from the Tensor Toolbox.

A. Synthetic Data

The purpose of the synthetic data experiments was to study algorithms' properties when used with data that has different characteristics. The three data characteristics studied were (1) the rank of the tensor (in case of Tucker, the size of the core tensor); (2) the density of the factor matrices (and core tensor); and (3) the noise level. For all experiments, we created 10 random data sets with identical properties and the reported results are averages over those data sets. In all figures, the width of the error bars is twice the standard deviation.

1) CP Decomposition: The synthetic CP data was made by first making random binary factor matrices of predefined density and size. These factor matrices were multiplied together to obtain binary tensors, after which noise was applied. All resulting tensors were of size 50-by-70-by-100. The results for the CP decomposition experiments are in Figure 1.

Tensor Rank: The rank of the tensor varied between 4 and 64 with factor matrix density being 0.5 and noise level being 0.1 (of 1s in the data). The results are seen in Figure 1. The first noticeable result is that with smaller values of r, BCP_ALS is the best of all methods, being better than even the real-valued methods with squared Frobenius error. With $r = 32, 64$, the real-valued methods become slightly better. The error of BCP_ALS mostly increases with the rank. This is probably due to the increased complexity of the data, making it harder for the iterative updates to find the optimal results.

Factor Matrix Density: The factor matrix density varied between 0.3 and 0.7 with rank being 16 and noise level 0.1. Results are in Figure 1(b). Here, $\texttt{CP\_ALS}_F$ seems to be the best method, though with density 0.5 BCP_ALS is the best (corresponding to Figure 1(a)). Why real-valued methods peak at density 0.5 is unclear (recall that this is not the density of the resulting data, but the factor matrix density). BCP_ALS, however, seems to behave as expected: denser data again has more complexity, making it harder for the algorithm.

Noise Level: The noise level varied between 0.05 and 0.4 (meaning that between $0.05\,|\boldsymbol{\mathcal{X}}|$ and $0.4\,|\boldsymbol{\mathcal{X}}|$ of elements of $\boldsymbol{\mathcal{X}}$ were flipped). Rank was 16 and density 0.5. The results are in Figure 1(c). With smaller values of noise BCP_ALS is again the best method, but as the noise level increased, its error increased faster than that of $\texttt{CP\_ALS}_F$ or $\texttt{CP\_OPT}_F$. Here the ability of making many small errors seems to benefit the latter two algorithms.

2) Tucker Decomposition: The synthetic Tucker data was made similarly to the synthetic CP data. The size of the resulting tensors were again 50-by-70-by-100, and unless otherwise mentioned, the core tensor was of size 8-by-8-by-8, density was 0.2, and noise level was 0.1. The results are in Figure 2.

In all experiments, BTucker_ALS is worse than $\texttt{Tucker\_ALS}_F$, and often also worse than $\texttt{Tucker\_ALS}_B$. This seems to suggest that either the created data has lower-error real-valued Tucker decomposition or that the complexity and the 'everything affects everything' nature of the Tucker decomposition makes it very hard for the iterative algorithm to find good solutions. Either way, notice that the BTucker_ALS algorithm has a peak at the error exactly at the default core tensor size (8-by-8-by-8) and default density (0.2). It is possible that different default values had yield better results for BTucker_ALS.

B. Real-World Data

For the real-world data we used entity–relation–entity tuples from the TextRunner open information extrac-

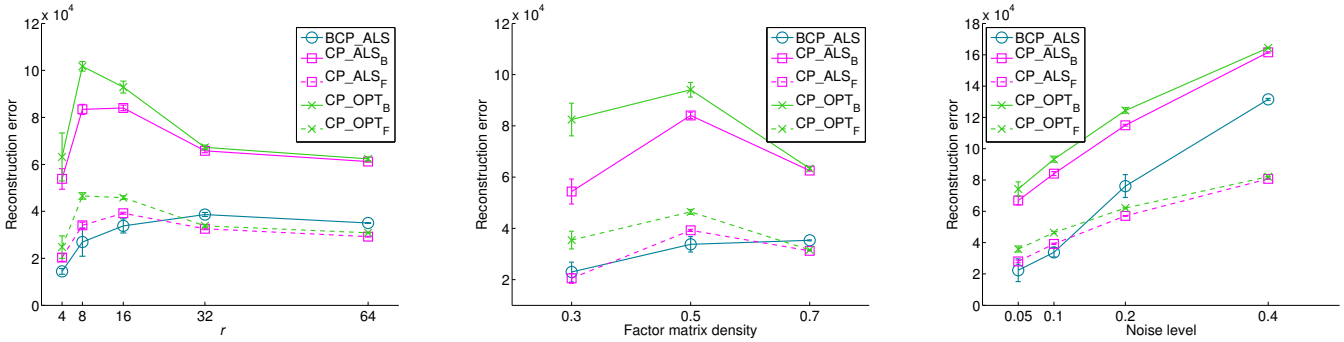

Figure 1. Results for synthetic CP decomposition data of different parameters: (a) Tensor rank. (b) Factor matrix density. (c) Noise level.

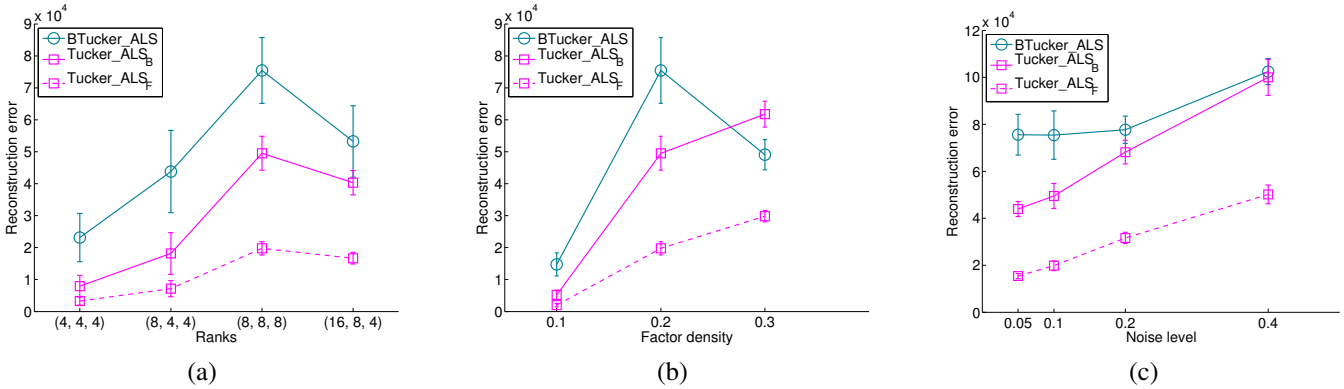

Figure 2. Results for synthetic Tucker decomposition data of different parameters: (a) Core tensor size. (b) Factor matrix density. (c) Noise level.

tion algorithm[3] [14]. The data itself is huge, so we made three subsets of it, called ResolverS, ResolverM, and ResolverL. ResolverS is 132-by-107-by-20 (entity-by-entity-by-relation), ResolverM is 151-by-191-by-70, and ResolverL is 343-by-360-by-200. All three data sets are extremely sparse.

1) The CP Decomposition: The reconstruction errors for the CP decomposition with different ranks are in Figure 3. For some reason, with ResolverS, `BCP_ALS` is unable to find good decomposition. We assume that this is due to the fact that the fibers of ResolverS are very sparse, typically having just one or two 1s. This means that there is almost no (Boolean) structure which `BCP_ALS` could find.

With the two larger data sets, however, the `BCP_ALS` algorithm performs relatively well, being constantly better than $\texttt{CP\_ALS}_B$ and $\texttt{CP\_OPT}_B$, and in case of ResolverL, better than $\texttt{CP\_OPT}_F$. While the real-valued methods returned dense factor matrices (as was expected), `BCP_ALS` returned very sparse ones (results omitted).

2) The Tucker Decomposition: The reconstruction errors for the Tucker decomposition with different core tensor sizes are in Figure 4. These results somewhat mirror those with CP decompositions: with ResolverS, `BTucker_ALS` is the worst, but with the two other data sets, `BTucker_ALS` resides between F and B variation of `Tucker_ALS`. Notice, though, that `BTucker_ALS` is almost constantly very close to $\texttt{Tucker\_ALS}_F$, the only exception being the bump in Figure 4(b).

[3]http://www.cis.temple.edu/~yates/papers/jair-resolver.html

C. Discussion

The experiments, both with synthetic and real-world data, show that the `BCP_ALS` algorithm performs very well, being comparable to (or better than) real-valued methods with squared Frobenius error. It achieves this while delivering much sparser factors. The results were also interpretable in the sense that the factor matrices defined sets of entities and relations that naturally belong together, such as geographic locations ({'Germany', 'India', 'Paris', 'Soviet Union'} was an example of one factor with ResolverL). The nature of the data, being very noisy and only a random subset of the full data made further analysis on the interpretability of the result hard.

The `BTucker_ALS` algorithm's performance was more mixed. With synthetic data it did not perform as well as

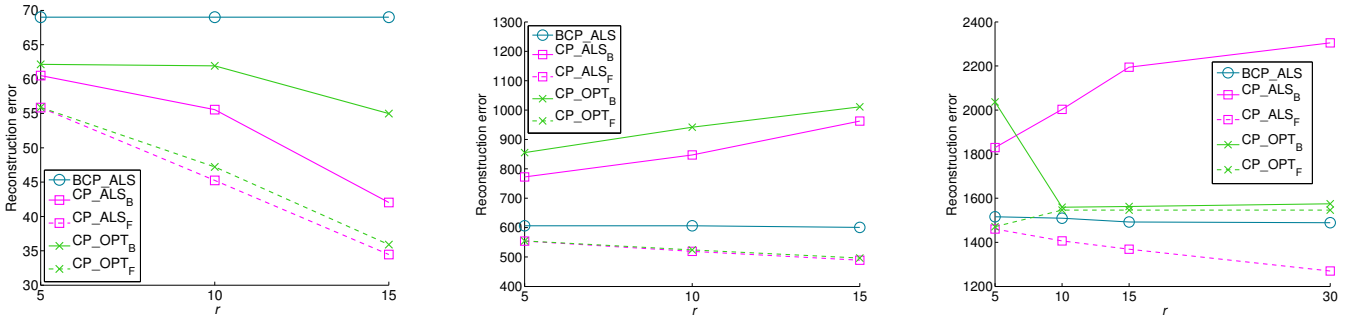

Figure 3. Results for CP decompositions at real-world data: (a) ResolverS. (b) ResolverM. (c) ResolverL.

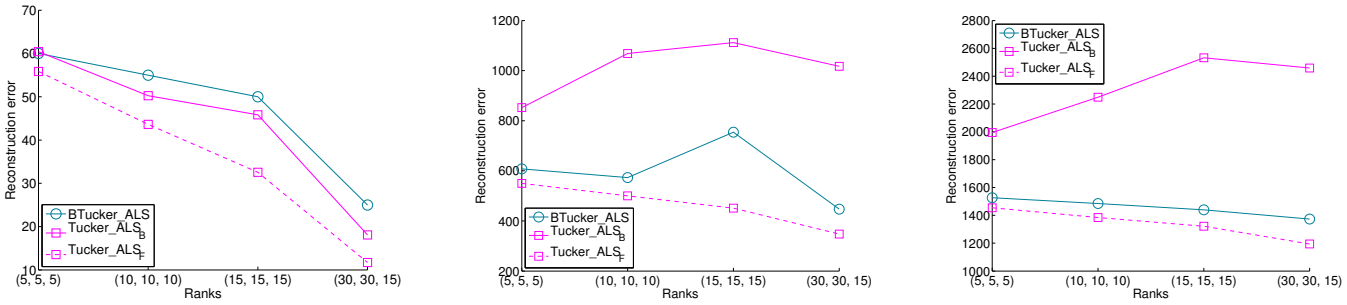

Figure 4. Results for Tucker decompositions at real-world data: (a) ResolverS. (b) ResolverM. (c) ResolverL.

expected, but with real-world data its performance was good. Which one of these reflects the typical scenario is an open question.

VI. Related Work

Boolean matrix factorizations have gained interest in data mining community during the past few years. The use of Boolean matrix factorizations in data mining was proposed in [11], although related concepts, such as tiles and formal concepts, were studied much earlier. Before that, Boolean tensor factorizations were mostly studied by combinatorics; see [10] and references therein. For some applications and variations of Boolean matrix factorizations, see [1].

Tensor factorizations are also well-studied, dating back to late Twenties. The two popular decomposition methods, Tucker and CP, were proposed in Sixties [7] and Seventies [3], [4], respectively. The topic has nevertheless attained growing interest in recent years, both in numerical linear algebra and computer science communities. For a comprehensive study of recent work, see [5].

One field of computer science that has adopted tensor decompositions is computer vision and machine learning. The interest to non-negative tensor factorizations stems from these fields [15], [16].

Many algorithms have been proposed to finding closed itemsets in N-way data [9], [17]–[22]. The output of these algorithms can then be used to find an N-way tiling of the binary tensor [9]. The concept of 3-way itemsets was generalized into dense triclusters that, unlike itemsets, can have 1s where the original data has 0s, by Ignatov et al. [23]. The use of triclusters in lossy 3-way tiling has not been studied, however.

VII. Conclusions

We have presented the Boolean CP and Tucker tensor decompositions. The theoretical analysis of these topics shows that Boolean tensor decompositions have some useful features (such as the sparsity), and that moving from matrices to tensors have 'leveled the playing field' with real-valued decompositions: problems that were hard with Boolean matrices but easy with real-valued matrices (such as the matrix rank) are now hard with both and the idiosyncrasies tensors bring to the Boolean case (such as high maximum rank) are also found in real-valued case. If fact, some of the problems with the real-valued tensor rank (such as degenerate tensors) do not manifest themselves with Boolean tensor rank. Nevertheless, there are still many open problems in the relation between Boolean and real tensor rank, most notably, what are the extremal differences between these two ranks

on a binary tensor (i.e. what is the lower bound of Boolean rank in terms of real rank and vice versa).

The algorithms we proposed were based on rather straight forward alternating optimization heuristics. Despite (or because of) this, they worked generally very well, being often almost as good, or even better, as real-valued ones. This happens very rarely with matrices: usually, SVD is the best, even if Boolean methods would have the theoretical possibilities to be better. Nevertheless, finding better algorithms, both in terms of accuracy and in terms of memory and time efficiency, is important. Of particular interest are algorithms that can handle large but extremely sparse tensors, perhaps in distributed manner for better scalability.

We have shown that Boolean tensor factorizations are viable data mining method, and expect them provide many interesting research questions in the upcoming years.

REFERENCES

[1] P. Miettinen, "Matrix decomposition methods for data mining: Computational complexity and algorithms," Ph.D. dissertation, University of Helsinki, 2009.

[2] ——, "Sparse Boolean Matrix Factorizations," in *ICDM '10*, 2010, pp. 935–940.

[3] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[4] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multimodal factor analysis," UCLA Working Papers in Phonetics, Tech. Rep., 1970.

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

[6] J. Håstad, "Tensor rank is NP-complete," *J. Algorithms*, vol. 11, no. 4, pp. 644–654, Dec. 1990.

[7] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[8] F. Geerts, B. Goethals, and T. Mielikäinen, "Tiling databases," in *DS '04*, 2004, pp. 77–122.

[9] L. Cerf, J. Besson, C. Robardet, and J.-F. Boulicaut, "Closed patterns meet n-ary relations," *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 1, 2009.

[10] S. D. Monson, N. J. Pullman, and R. Rees, "A Survey of Clique and Biclique Coverings and Factorizations of $(0,1)$-Matrices," *Bull. ICA*, vol. 14, pp. 17–86, 1995.

[11] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila, "The Discrete Basis Problem," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 10, pp. 1348–1362, Oct. 2008.

[12] E. Acar, T. G. Kolda, and D. M. Dunlavy, "An optimization approach for fitting canonical tensor decompositions," Sandia National Laboratories, Tech. Rep. SAND2009-0857, 2009.

[13] B. W. Bader and T. G. Kolda, "MATLAB Tensor Toolbox version 2.4," 2010, http://csmr.ca.sandia.gov/tgkolda/TensorToolbox/.

[14] A. Yates and O. Etzioni, "Unsupervised methods for determining object and relation synonyms on the web," *J. Artif. Intell. Res.*, vol. 34, pp. 255–296, 2009.

[15] A. Shashua and T. Hazan, "Non-negative tensor factorization with applications to statistics and computer vision," in *ICML '05*, 2005.

[16] Y.-D. Kim and S. Choi, "Nonnegative Tucker Decomposition," in *CVPR '07*, 2007, pp. 1–8.

[17] R. V. Nataraj and S. Selvan, "Closed Pattern Mining from n-ary Relations," *Int. J. Comput. Appl.*, vol. 1, no. 9, pp. 9–13, 2010.

[18] R. Missaoui and L. Kwuida, "Mining Triadic Association Rules from Ternary Relations," in *ICFCA '11*, 2011, pp. 204–218.

[19] S. Selvan and R. V. Nataraj, "Efficient Mining of Large Maximal Bicliques from 3D Symmetric Adjacency Matrix," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 12, pp. 1797–1802, 2010.

[20] R. Jaschke, A. Hotho, C. Schmitz, B. Ganter, and G. Stumme, "TRIAS — An Algorithm for Mining Iceberg Tri-Lattices," in *ICDM '06*, 2006, pp. 907–911.

[21] L. Ji, K.-L. Tan, and A. K. H. Tung, "Mining frequent closed cubes in 3D datasets," in *VLDB '06*. VLDB Endowment, 2006, pp. 811–822.

[22] R. Bělohlávek and V. Vychodil, "Factorizing Three-Way Binary Data with Triadic Formal Concepts," in *KES '10*, 2010, pp. 471–480.

[23] D. I. Ignatov, S. O. Kuznetsov, R. A. Magizov, and L. E. Zhukov, "From Triconcepts to Triclusters," in *RSFDGrC '11*, 2011, pp. 257–264.