Data Min Knowl Disc (2016) 30:1–46
DOI 10.1007/s10618-015-0401-6

Decomposition-by-normalization (DBN): leveraging approximate functional dependencies for efficient CP and tucker decompositions

Mijung Kim · K. Selçuk Candan

Received: 10 October 2013 / Accepted: 4 January 2015 / Published online: 28 January 2015

Abstract For many multi-dimensional data applications, tensor operations as well as relational operations both need to be supported throughout the data lifecycle. Tensor based representations (including two widely used tensor decompositions, CP and Tucker decompositions) are proven to be effective in multi-aspect data analysis and tensor decomposition is an important tool for capturing high-order structures in multi-dimensional data. Although tensor decomposition is shown to be effective for multi-dimensional data analysis, the cost of tensor decomposition is often very high. Since the number of modes of the tensor data is one of the main factors contributing to the costs of the tensor operations, in this paper, we focus on reducing the modality of the input tensors to tackle the computational cost of the tensor decomposition process. We propose a novel `decomposition-by-normalization` scheme that first normalizes the given relation into smaller tensors based on the functional dependencies of the relation, decomposes these smaller tensors, and then recombines the sub-results to obtain the overall decomposition. The decomposition and recombination steps of the `decomposition-by-normalization` scheme fit naturally in settings with multiple cores. This leads to a highly efficient, effective, and parallelized `decomposition-by-normalization` algorithm for both dense and sparse tensors for CP and Tucker decompositions. Experimental results confirm the efficiency and effectiveness of the proposed `decomposition-by-normalization`

Responsible editors: Chih-Jen Lin.

M. Kim (✉) · K. S. Candan
Arizona State University, Tempe, AZ, USA
e-mail: mijung.kim.1@asu.edu

K. S. Candan
e-mail: candan@asu.edu

scheme compared to the conventional nonnegative CP decomposition and Tucker decomposition approaches.

Keywords Tensor decomposition · CP decomposition · Tucker decomposition · Parallel tensor decomposition · Parallel CP decomposition · Parallel Tucker decomposition

1 Introduction

Relational data have various representations. Let $A_1, \ldots, A_n$ be a set of attributes in a relation and $D_1, \ldots, D_n$ be the attribute domains. The tensor model maps each attribute to a mode in an n-modal array where each possible tuple is a cell. The existence (absence) of a particular tuple in the database instance is denoted by inserting a 1 (0) in the cell; the model can also represent fuzzy or probabilistic tuples by filling the cell with a value between 0 and 1.

Tensor based representations, including two widely used decompositions, CP (Carroll and Chang 1970; Harshman 1970) and Tucker (Tucker 1966) decompositions, are proven to be effective in multi-aspect data analysis. Consequently, tensor decomposition is an important tool for capturing high-order structures in multi-dimensional data in many application domains including scientific data management (Andersson and Bro 2000; Harshman 1970; Phan and Cichocki 2011; Tucker 1966; Zhang et al. 2009), sensor data management (Sun et al. 2008; Tsourakakis 2010), and social network data analysis (Kolda et al. 2005; Kolda and Sun 2008; Mahoney et al. 2006).

Tensor decomposition gives benefits over matrix-based tools, which allow more powerful data analysis such as cross-mode clustering (Sun et al. 2009) and time evolving streaming data analysis (Sun et al. 2008). In particular, we focus on using tensor decomposition on clustering problems. Tensor decomposition is widely used for clustering (Kolda and Sun 2008; Sun et al. 2009, 2008).

Unfortunately, tensor decomposition operation can be prohibitively costly when the tensor data have a large number of modes:

– One obvious problem is the space needed to hold the input tensors. When the tensor is *dense* (i.e., has a large number of nonzero entries) or when a dense tensor representation is used for algorithmic reasons, the space required to hold the data increases exponentially with the number of modes.
– The Tucker decomposition may be infeasible for large data sets (even if the original tensor is sparse) since the tensors needed to represent intermediate results are often dense.

Recent attempts to overcome these problems using parellel tensor decomposition (Antikainen et al. 2011; Phan and Cichocki 2011; Zhang et al. 2009) techniques also face difficulties, including synchronization and data exchange overheads.

1.1 Contributions of this paper: decomposition-by-normalization (DBN)

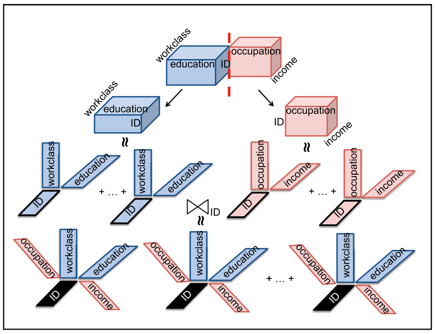
Our goal in this paper is to tackle the high computational cost of the tensor decomposition process. Since, as described above, the number of modes of the tensor data is

workclass	education	ID	occupation	income
Self-emp-not-inc	Masters	1	Exec-managerial	>50K
Never-worked	Bachelors	2	-	<=50K
Private	Some-college	3	Sales	>50K
…	…	…	…	…

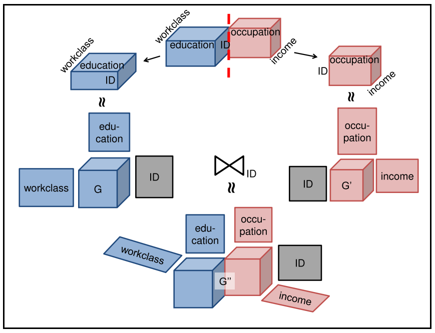
workclass	education	ID
Self-emp-not-inc	Masters	1
Never-worked	Bachelors	2
Private	Some-college	3
…	…	…

ID	occupation	income
1	Exec-managerial	>50K
2	-	<=50K
3	Sales	>50K
…	…	…

(a) Normalization (vertical data partitioning)



(b) DBN process for CP decomposition



(c) DBN process for Tucker decomposition

Fig. 1 **a** Normalization of a relation $\mathcal{R}$(`workclass, education, ID, occupation, income`) into two relations $\mathcal{R}_1$(`workclass, education, ID`) and $\mathcal{R}_2$(`ID, occupation, income`) based on the key (`ID`); `decomposition-by-normalization` (DBN): normalization of $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$, **b** rank-r_1 CP decomposition of $\mathcal{R}_1$ and rank-r_2 CP decomposition of $\mathcal{R}_2$ that are combined on the `ID` mode into rank-($r_1 \times r_2$) CP decomposition of $\mathcal{R}$, and **c** rank-$(\ldots, r_1, \ldots)$ Tucker decomposition of $\mathcal{R}_1$ and rank-$(\ldots, r_2, \ldots)$ Tucker decomposition of $\mathcal{R}_2$ that are combined on the `ID` mode into rank-$(\ldots, r_1 \times r_2, \ldots)$ Tucker decomposition of $\mathcal{R}$

one of the main factors contributing to the cost of the tensor operations, we argue that if

– a tensor with large number of modes can be normalized (i.e., vertically partitioned) into tensors with smaller number of modes and
– each sub-tensor is decomposed independently,

then the resulting partial decompositions can be efficiently combined to obtain the decomposition of the original tensor. We refer to this as the `decomposition-by-normalization` (DBN) scheme.

Example 1 Consider the 5-attribute relation, $\mathcal{R}$(`workclass, education, ID, occupation, income`) in Fig. 1a and assume that we want to decompose the corresponding tensor for multi-dimensional analysis.

Figure 1a illustrates an example normalization which divides this 5-attribute relation into two smaller relations with 3 attributes, $\mathcal{R}_1$`(workclass, education, ID)` and $\mathcal{R}_2$`(ID, occupation, income)`, respectively.

Figure 1b, c, then, illustrate the proposed DBN scheme for CP and Tucker decompositions, respectively: In both cases, once the two partitions are decomposed, we combine the resulting core tensors and factor matrices to obtain the decomposition of the original tensor corresponding to the relation $\mathcal{R}$. □

Benefits of DBN for CP Decompositions In the CP decomposition example above (Fig. 1b),

– if the input relation $\mathcal{R}$ is *dense*, we argue that decompositions of partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ will be much faster than that of the original relation $\mathcal{R}$ and the gain will more than compensate for the normalization and recombination costs of DBN.
– If the input relation $\mathcal{R}$ is *sparse*, on the other hand, the decomposition cost is not only determined by the number of modes, but also the number of nonzero entries in the tensor. Consequently, unless the partitioning provides smaller numbers of tuples in both partitions, we cannot theoretically expect DBN to provide large gains. However, as we experimentally verify in Sect. 8, DBN scheme fits naturally in multi-core implementations, thus in practice provides significant advantages even for sparse input tensors.

Benefits of DBN for Tucker Decompositions Since the scale of the intermediate blowup problem (Kolda and Sun 2008) depends largely on the modality of the input tensor, we argue that dividing the tensor into sub-tensors with smaller number of modes will help eliminate this notorious bottleneck. Moreover, similarly to the case in CP decompositions, each individual sub-tensor decomposition can run on an available core without having to communicate with other sub-tensor decompositions running on different cores, leading to effective parallelizations of Tucker decompositions. *Challenges and Contributions* Note that in general, a given tensor can be partitioned into two in multiple ways. The key challenges we address in this paper are (a) *how best to partition* a given tensor into smaller tensors and (b) *how to recombine* the sub-result to obtain the decomposition of the original tensor. In particular, achieving the projected advantages of the DBN strategy requires us to address the following key challenges:

– *Challenge 1* First of all, we need to ensure that the join attribute is selected in such a way that the normalization (i.e., the vertical partitioning) process does not lead to spurious tuples. Secondly, the join attribute needs to partition the data in such a way that the later steps in which decompositions of the individual partitions are combined into an overall decomposition do not introduce errors. One way to prevent the normalization process from introducing spurious data is to select an attribute which *functionally determines* the attributes that will be moved to the second partition. *This requires an efficient method to determine functional dependencies in the data.*
– *Challenge 2* A second difficulty is that many data sets may not have perfect functional dependencies to leverage for normalization. *In that case, we need to be able to identify and rely on approximate functional dependencies in the data.*

- *Challenge 3* Once the approximate functional dependencies are identified, *we need a mechanism to partition the data into two partitions in such a way that will lead to least amount of errors during later stages.* In this paper, we argue that partitioning the attributes in a way that minimizes *inter-partition* functional dependencies and maximizes *intra-partition* dependencies will lead to least amount of errors in the recombination step.
- *Challenge 4* Moreover, after data is vertically partitioned and individual partitions are decomposed, the individual decompositions need to be *recombined to obtain the decomposition* of the original relation. This process needs to be done in a way that is efficient and parallelizable.

The paper is organized as follows: We first provide the relevant background and discuss the related works in Sect. 2. We provide an overview of the proposed DBN scheme in Sect. 3. In Sect. 4, we extend the CP-based `join-by-decomposition` (JBD-CP) (Kim and Candan 2011) approach to Tucker decompositions. We then focus on selecting the best partitions for the normalization step of DBN (Sect. 5). In Sect. 6, we present rank-pruning strategies to further reduce the cost of DBN. We experimentally evaluate DBN in Sect. 8 in both stand-alone and parallel configurations. We focus on the accuracy and the running time of the alternative algorithms. Note that although the proposed algorithms are developed for nonnegative CP and Tucker decompositions, experiments show that in practice they work well also for general tensor decompositions involving negative values. Experimental results provide evidence that in addition to being significantly faster than conventional decompositions, DBN can approximate well the accuracy of the conventional tensor decomposition techniques. We conclude the paper in Sect. 9.

2 Background and related work

2.1 Tensors and tensor decomposition

Tensors Tensors are generalizations of matrices: while a matrix is essentially a two dimensional array, a tensor is an array of arbitrary dimensions. Thus, a vector can be thought of as a tensor of 1st order and an object-feature matrix is a tensor of 2nd order, while a multi-sensor data stream (i.e., sensors, features of sensed data, and time) can be represented as a tensor of 3rd order. As in the case of matrices, the dimensions of the tensor array are referred to as its *modes*.

Tensor Decomposition Tensor decomposition has been used in a large number of domains, including signal processing, computer vision, and data mining. Tensor-based data representation and tensor analysis are also increasingly popular in emerging fields, such as social network analysis (Kolda et al. 2005). The two most popular tensor decompositions are the Tucker (Tucker 1966) and the CANDECOMP/PARAFAC (Harshman 1970; Carroll and Chang 1970) decompositions, which are considered to be higher-order generalization of matrix singular value matrix decomposition (SVD). CANDECOMP (Carroll and Chang 1970) and PARAFAC (Harshman 1970) decompositions (together known as the CP decomposition) take a different approach and decompose the input tensor into a sum of component rank-one tensors. More specifically, the rank-r CP Decomposition of the tensor $\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}$ can be defined as

$$CP(\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}) = \tilde{\mathcal{P}}_{I_1 \times I_2 \times \cdots \times I_N} = \langle \lambda, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \rangle,$$

such that

$$\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N} \approx \sum_{k=1}^{r} \lambda_k \circ U_{:k}^{(1)} \circ U_{:k}^{(2)} \circ \cdots \circ U_{:k}^{(N)}, \tag{1}$$

where λ_i is the i-th element of vector λ of size r and $U_{:i}^{(n)}$ is the i-th column vector of the matrix $\mathbf{U}^{(n)}$ of size $I_n \times r$, for $n = 1, \cdots, N$.

Note that the CP decomposition operation is an approximate operation and $\mathcal{P}$ may not be exactly reconstructed from $\tilde{\mathcal{P}}$. In other words, the following weighted sum, $\hat{\mathcal{P}}$, of the rank-one tensors may be different from $\mathcal{P}$:

$$\hat{\mathcal{P}}_{I_1 \times I_2 \times \cdots \times I_N} = \sum_{k=1}^{r} \lambda_k \circ U_{:k}^{(1)} \circ U_{:k}^{(2)} \circ \cdots \circ U_{:k}^{(N)}. \tag{2}$$

Therefore, the norm of $\mathcal{P}$ denoted by $\|\mathcal{P}\|$ may also be different from $\|\hat{\mathcal{P}}\|$. Note that $\|\hat{\mathcal{P}}\|$ can be computed directly from the decomposition $\tilde{\mathcal{P}}$ without having to reconstruct the tensor $\hat{\mathcal{P}}$, since $\|\hat{\mathcal{P}}\| = \|\tilde{\mathcal{P}}\|$, which is computed in Bader and Kolda (2006) as

$$\|\tilde{\mathcal{P}}\| = \lambda^T (\mathbf{U}^{(N)T} \mathbf{U}^{(N)} * \cdots * \mathbf{U}^{(1)T} \mathbf{U}^{(1)}) \lambda. \tag{3}$$

Tucker decomposition (Tucker 1966) is a higher-order generalization of principal component analysis (PCA) (Kolda and Bader 2009). The rank-$(r_1, r_2, \ldots, r_N)$ Tucker Decomposition of the tensor $\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}$ can be defined as

$$Tucker(\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}) = \tilde{\mathcal{P}}_{I_1 \times I_2 \times \cdots \times I_N} = \langle \mathcal{G}, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \rangle,$$

such that

$$\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N} \approx \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_N \mathbf{U}^{(N)}, \tag{4}$$

where $\mathcal{G}$ is a core tensor of size $r_1 \times r_2 \times \cdots \times r_N$ and $\mathbf{U}^{(n)}$ is the n^{th} factor matrix of size $I_n \times r_n$, for $n = 1, \ldots, N$. Again, the norm of Tucker decomposition of $\mathcal{P}$, $\|\hat{\mathcal{P}}\|$ is computed directly from the decomposition $\tilde{\mathcal{P}}$ (see Bader and Kolda (2006) for the computation of $\|\tilde{\mathcal{P}}\|$).

Many of the algorithms for decomposing tensors are based on an iterative process that approximates the best solution until a convergence condition is reached. The *alternating least squares* (ALS) method is relatively old and has been successfully applied to the problem of tensor decomposition (Carroll and Chang 1970; Harshman 1970). Non-iterative approaches to tensor decomposition include closed form solutions, such as generalized rank annihilation method (GRAM) (Sanchez and Kowalski 1986) and direct trilinear decomposition (DTLD) (Sanchez and Kowalski 1990), which fit the model by solving a generalized eigenvalue problem.

Related approaches to tensor decomposition include probabilistic methods (Chu and Ghahramani 2009; Hoff 2011; Xu et al. 2012), which aim to capture correlations between different tensor modes to handle missing and imprecise entries. In this paper, we also interpret tensor decomposition probabilistically by imposing additional constraints (nonnegativity and summation to 1) on the decomposition. In the case of the CP decomposition, for example, each nonzero element in the core can be thought of denoting a cluster strength and the values of the entries of the factor matrices can be interpreted as the conditional probabilities of the entries given clusters. Our probabilistic treatment of the tensor decomposition process, however, is motivated not by handling imprecise data, but to support lossless (or minimal loss) partitioning and recombination of the tensors.

Scalable and Parallel Tensor Decompositions Randomized sampling has been used to approximate tensor decomposition for space and time savings (Mahoney et al. 2006; Tsourakakis 2010). Kolda et al. (2005) developed a greedy PARAFAC algorithm for large-scale, sparse tensors in MATLAB. A memory-efficient Tucker (MET) proposed in Kolda and Sun (2008) addressed the intermediate blowup problem in Tucker decomposition. According to the ALS method for solving Tucker Decomposition, the bottleneck computation is the input tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \cdots \times I_N$ times factor matrices $\mathbf{A}^{(n)}$ of size $I_n \times r_n$ for $n = 1, \ldots, N$,

$$\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)} \cdots \times_{(n-1)} \mathbf{A}^{(n-1)} \times_{(n+1)} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)}.$$

Since the intermediate result of a sparse tensor multiplied by factor matrices can be dense, intermediate results may be too big to fit in the available memory, even when the final result $\mathcal{Y}$, whose size is $\max_n (I_n \prod_{m \neq n} r_m)$ may easily fit. MET addresses this problem by calculating $\mathcal{Y}$ in an element-wise manner for reducing the size of intermediate memory. Instead of updating the whole $\mathcal{Y}$, MET updates a subset of the modes (e.g., each slice $\mathbf{Y}_{:j_2:}$ or fiber $\mathbf{y}_{:j_2 j_3}$). As a result, the size of intermediate result of MET is $\prod_{m \notin \varepsilon} I_m$, where ε is a subset of modes computed element-wise. We show later in Sect. 8, MET still suffers from the intermediate blowup problem in high mode (more than 4-mode) input tensors whereas DBN performs well.

Phan and Cichocki (2011) proposed a modified ALS PARAFAC algorithm called grid PARAFAC for large scale tensor data. The grid PARAFAC divides a large tensor into sub-tensors that can be factorized using any available PARAFAC algorithm in a parallel manner and iteratively combines into the final decomposition. The grid PARAFAC can be converted to grid NTF by enforcing nonnegativity.

The grid PARAFAC reduces the dimensionality of the input tensor, while our approach reduces the number of modes of the input tensor, in order to address the high cost of tensor decomposition.

Zhang et al. (2009) parallelized NTF by dividing a given original 3-mode tensor into three semi-non negative matrix factorization problems. These matrices are distributed to independent processors to facilitate parallelization. A parallelized NTF algorithm that is specialized for Compute Uniform Device Architecture (CUDA) parallel computing framework was presented in Antikainen et al. (2011).

Note that since block-based parallel algorithms are based on ALS where one variable can be optimized given that the other variables are fixed, the communication cost among the blocks is not avoidable. In the proposed parallelized DBN strategy, on the other hand, each block is completely separable and run independently.

Zhou et al. (2009) provide an efficient way of performing N-way CP decomposition where they leverage the fact that one factor matrix is correctly estimated, the other factor matrices can be computed and estimate two factors first then perform the full CP decomposition by the Khatri-Rao product.

3 Decomposition-by-normalization (DBN)

Our goal in this paper is to tackle the high computational cost of decomposition process through what we refer to as the `decomposition-by-normalization` (DBN). In this section, we first introduce the relevant notations, provide background on key concepts, and then present an overview of the DBN process.

3.1 Key concepts

Without loss of generality, we assume that relations are represented in the form of *occurrence tensors* Kim and Candan (2011).

3.1.1 Tensor representation of relational data

Let $A_1, \ldots, A_n$ be a set of attributes in the schema of a relation, $\mathcal{R}$, and $D_1, \ldots, D_n$ be the attribute domains. Let the relation instance $\mathcal{R}$ be a finite multi-set of tuples, where each tuple $t \in D_1 \times \ldots \times D_n$.

Definition 1 *(Occurrence Tensor)* An *occurrence tensor* $\mathfrak{R}_o$ corresponding to the relation instance $\mathcal{R}$ is an n-mode tensor, where each attribute $A_1, \ldots, A_n$ is represented by a mode. For the i-th mode, which corresponds to A_i, let $D'_i \subseteq D_i$ be the (finite) subset of the elements such that

$$\forall v \in D'_i \ \exists t \in \mathcal{R} \ \ s.t. \ \ t.A_i = v,$$

where $t.A_i$ denotes the i-th attribute of the tuple t and let $idx(v)$ denote the rank of v among the values (or a category in case that the attribute is categorical) in D'_i relative to an (arbitrary) total order, $<_i$, defined over the elements of the domain, D_i. The cells of the *occurrence tensor* $\mathfrak{R}_o$ are such that

$$\mathfrak{R}_o[u_1, \ldots, u_n] = 1 \leftrightarrow \exists t \in \mathcal{R} \ \ s.t. \ \ \forall_{1 \le j \le n} \ idx(t.A_j) = u_j$$

and 0 otherwise.

Intuitively, each cell indicates whether the corresponding tuple exists in the multi-set corresponding to the relation or not. ∘

3.1.2 Tensor join

Since in this paper we represent relations as occurrence tensors, we rely on the definition of *tensor join* operation introduced in Kim and Candan (2011) when we need to combine data sets represented as tensors: Let $\mathfrak{P}$ and $\mathfrak{Q}$ be two tensors, representing relation instances $\mathcal{P}$ and $\mathcal{Q}$, with attribute sets, $\mathbb{A}^P = \{A_1^P, \ldots, A_n^P\}$ and $\mathbb{A}^Q = \{A_1^Q, \ldots, A_m^Q\}$, respectively. In the rest of this section, we denote the index of each cell of $\mathfrak{P}$ as $(i_1, i_2, \ldots, i_n)$; similarly, the index of each cell of $\mathfrak{Q}$ is denoted as $(j_1, j_2, \ldots, j_m)$. The cell indexed as $(i_1, \ldots, i_n)$ of $\mathfrak{P}$ is denoted by $\mathfrak{P}[i_1, \ldots, i_n]$ and the cell indexed as $(j_1, \ldots, j_m)$ of $\mathfrak{Q}$ is denoted by $\mathfrak{Q}[j_1, \ldots, j_m]$.

Definition 2 *(Tensor Join ($\bowtie$) and Tensor Equi-Join)* Given two relational tensors $\mathfrak{P}$ and $\mathfrak{Q}$, and a condition φ, we define their join as

$$\mathfrak{P} \bowtie_\varphi \mathfrak{Q} \overset{\text{def}}{=} \sigma_\varphi(\mathfrak{P} \times \mathfrak{Q}).$$

Given two relations $\mathcal{P}$ and $\mathcal{Q}$, with attribute sets, $\mathbb{A}^P = \{A_1^P, \ldots, A_n^P\}$ and $\mathbb{A}^Q = \{A_1^Q, \ldots, A_m^Q\}$, and a set of attributes $\mathbb{A} \subseteq \mathbb{A}^P$ and $\mathbb{A} \subseteq \mathbb{A}^Q$, the *equi-join* operation, $\bowtie_{=,\mathbb{A}}$, is defined as the join operation, with the condition that matching attributes in the two relations will have the same values, followed by a projection operation that eliminates one instance of $\mathbb{A}$ from the resulting relation. □

3.1.3 Functional dependencies

A functional dependency (FD) between two sets of attributes, $\mathbb{X}$ and $\mathbb{Y}$, is defined as follows (Elmasri and Navathe 1994).

Definition 3 *(Functional Dependency)* A functional dependency (FD), denoted by $\mathbb{X} \rightarrow \mathbb{Y}$, holds for relation instance $\mathcal{R}$, if and only if for any two tuples t_1 and t_2 in $\mathcal{R}$ that have $t_1[\mathbb{X}] = t_2[\mathbb{X}]$, $t_1[\mathbb{Y}] = t_2[\mathbb{Y}]$ also holds.

We refer to a functional dependency as a *pairwise functional dependency* if the sets $\mathbb{X}$ and $\mathbb{Y}$ are both singleton. ∘

Intuitively, a functional dependency is a constraint between two sets of attributes $\mathbb{X}$ and $\mathbb{Y}$ in a relation denoted by $\mathbb{X} \rightarrow \mathbb{Y}$, which specifies that the values of the $\mathbb{X}$ component of a tuple uniquely determine the values of the $\mathbb{Y}$ component. Note that if $\mathbb{A} = \{A_1, \ldots, A_n\}$ is a set of attributes in the schema of a relation, R, and $\mathbb{X}, \mathbb{Y} \subseteq \mathbb{A}$ are two subsets of attributes such that $\mathbb{X} \rightarrow \mathbb{Y}$, then the relation instance $\mathcal{R}$ can be vertically partitioned into two relation instances $\mathcal{R}_1$, with attributes $\mathbb{A} \setminus \mathbb{Y}$, and $\mathcal{R}_2$, with attributes $\mathbb{X} \cup \mathbb{Y}$, such that $\mathcal{R} = \mathcal{R}_1 \bowtie \mathcal{R}_2$; in other words the set of attributes $\mathbb{X}$ serves as a foreign key and joining vertical partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ on $\mathbb{X}$ gives back the relation instance $\mathcal{R}$ without any missing or spurious tuples.

Note that, discovery of FDs in a given data set is a challenging problem since the complexity increases exponentially in the number of attributes (Mannila and Räihä 1992). Moreover, in many data sets, attributes may not have perfect FDs due to exceptions and outliers in the data. In such cases, we may only be able to locate approximate FDs (Huhtala et al. 1999) instead of exact FDs:

```
DBN algorithm (input: a relation $\mathcal{R}$, a decomposition algorithm (CP or Tucker))
 1: Identify paFD (pairwise approximate FDs between the pairs of attributes of $\mathcal{R}$)
 2: Select the attribute $A_k$ with the highest $\sum_{k \neq j} \sigma_{k,j}$ such that $\sigma_{k,j} \geq \tau_{support}$ as the vertical partitioning (and
    join) attribute $X$ (Desiderata 1 and 2)
 3: if $\mathcal{R}$ is a sparse tensor then
 4:     if $X$ (approximately) determines all attributes of $\mathcal{R}$ then
 5:         findInterFDPartition(paFD,false) (see Figure 3)
 6:     else
 7:         Move $X$ and all attributes determined by $X$ to $\mathcal{R}_1$; move $X$ and remaining attributes to $\mathcal{R}_2$ (Desideratum
            4 and 5).
 8:     end if
 9: else {i.e., $\mathcal{R}$ is a dense tensor}
10:     if $X$ (approximately) determines all attributes of $\mathcal{R}$ then
11:         findInterFDPartition(paFD,true)
12:     else
13:         Move $X$ and attributes determined by $X$ to $\mathcal{R}_1$; move $X$ and remaining attributes to $\mathcal{R}_2$ – these moves
            are constrained such that the number of attributes of $\mathcal{R}_1$ and $\mathcal{R}_2$ are similar (Desideratum 3 and 5)
14:     end if
15: end if
16: Partition $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$.
17: If the selected $X$ does not perfectly determine the attributes of $\mathcal{R}_1$ then remove sufficient number of outlier
    tuples from $\mathcal{R}$ to enforce the FDs between $X$ and the attributes of $\mathcal{R}_1$
18: Create occurrence tensors of $\mathcal{R}_1$ and $\mathcal{R}_2$
19: Run JBD-CP (Figure 5) or JBD-Tucker (Figure 6) algorithm according to the input decomposition algorithm
    with the tensors corresponding to $\mathcal{R}_1$ and $\mathcal{R}_2$
```

Fig. 2 Pseudo-code of DBN

Definition 4 *(Approximate Functional Dependency)*

An approximate functional dependency (aFD), denoted by $\mathbb{X} \overset{\sigma}{\rightarrow} \mathbb{Y}$ holds for relation instance $\mathcal{R}$, if and only if

– there is a subset $\mathcal{R}' \subseteq \mathcal{R}$, such that $|\mathcal{R}'| = \sigma \times |\mathcal{R}|$ and, for any two tuples t_1 and t_2 in $\mathcal{R}'$ that have $t_1[\mathbb{X}] = t_2[\mathbb{X}]$, $t_1[\mathbb{Y}] = t_2[\mathbb{Y}]$ also holds; and
– there is no subset $\mathcal{R}'' \subseteq \mathcal{R}$, such that $|\mathcal{R}''| > \sigma \times |\mathcal{R}|$ where the condition holds.

We refer to the value of σ as the support of the aFD, $\mathbb{X} \overset{\sigma}{\rightarrow} \mathbb{Y}$. □

Many algorithms for FD and approximate FD discovery exist, including TANE (Huhtala et al. 1999), Dep-Miner (Lopes et al. 2000), FastFD (Wyss et al. 2001), and CORDS (Ilyas et al. 2004).

3.2 Overview of the decomposition-by-normalization (DBN) process

The overall structure of the `decomposition-by-normalization` (DBN) process, visualized in Fig. 1, is similar for both CP and Tucker decompositions. In this subsection, we present and discuss the pseudo code of DBN. In the following sections, we will study the key steps of the process in greater detail.

The pseudo code of DBN algorithm is presented in Fig. 2. In its first step (Line 1), DBN evaluates the pairwise (approximate) FDs among the attributes of the input relation. For this purpose, we employ and extend TANE (Huhtala et al. 1999), an efficient algorithm for discovering FDs. Our modification of the TANE algorithm returns a set of (approximate) FDs between attribute pairs and, for each candidate dependency, $A_i \rightarrow A_j$, it provides a corresponding support value, $\sigma_{i,j}$.

```
findInterFDPartition ( input: paFD, balanced)
1: Create a complete pairwise approximate FD graph with weighted nodes, $G$, where each node is an attribute with
   the weight, which is the size of the corresponding attribute and edge weights are the support values of paFD.
2: if balanced == false then
3:     Run minimum average cut on $G$ to find a maximally independent partitioning (Desideratum 5)
4: else {i.e., balanced == true}
5:     Run balanced cut on $G$ to find a balanced cut first and in case that there are alternative balanced cuts, maxi-
       mally independent partitioning (Desideratum 3 and 5)
6: end if
```

Fig. 3 Pseudo-code of interFD-based partition algorithm; this is detailed in Sect. 5.2

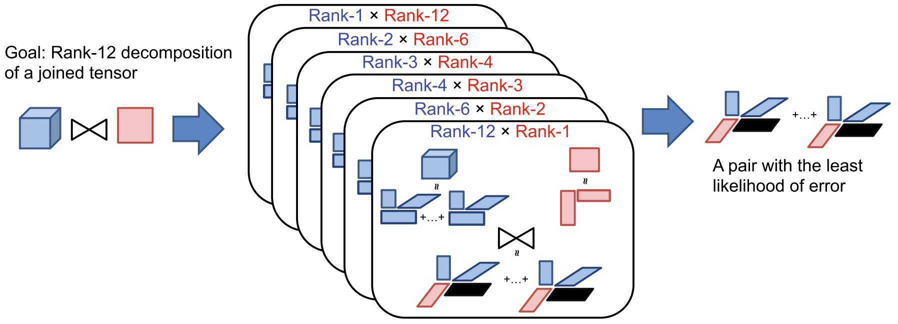


Fig. 4 Rank-12 decomposition of a joined tensor by `join-by-decomposition` where there are 6 pairs of decompositions and a pair with the least likelihood of error is chosen as the final decomposition

The next steps of the algorithm involve selecting the attribute, X, that will serve as the foreign key (Line 2) and partitioning the input relation $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$ around X (Lines 3 through 16). If the selected join attribute X does not perfectly determine the attributes of $\mathcal{R}_1$, then to prevent introduction of spurious tuples, we need to remove (outlier) tuples from $\mathcal{R}$ to restore the discovered FDs between the attribute, X, and the attributes that are selected to be moved to partition $\mathcal{R}_1$ (Line 17). Note that a major part of the DBN algorithm involves deciding how to partition the input data into two in the most effective manner. In Sect. 5, we will discuss the partitioning process in detail (Fig. 3).

Finally, once $\mathcal{R}_1$ and $\mathcal{R}_2$ are obtained, we create the occurrence tensors for the two partitions (Line 18) and execute the JBD-CP and JBD-Tucker modules (Fig. 4), to obtain the final decomposition through a `join-by-decomposition` process (Line 19). We discuss the `join-by-decomposition` step next.

4 Join-by-decomposition

The join-by-decomposition (JBD-CP) algorithm for CP decompositions was proposed in Kim and Candan (2011) to efficiently obtain non-negative CP decompositions of joined tensors. Instead of first performing a join and then decomposing the result using CP, JBD-CP first performs CP decompositions of the input data sets, and then combines the results. In this section, we first provide an overview of this JBD-CP scheme, and then extend it to Tucker decompositions of joined tensors. We refer to the JBD algorithm for Tucker decompositions as JBD-Tucker. The pseudo-codes of

```
JBD-CP algorithm (input: two tensors $\mathcal{P}$, $\mathcal{Q}$, rank $r$, and the modes of the join factor matrix of $\mathcal{P}$ and $\mathcal{Q}$)
1: for each pair of factors, $r_p$ and $r_q$ such that $r_p \times r_q = r$ do
2:     Run any available CP algorithm to get $\tilde{\mathcal{P}}_{r_p}$ and $\tilde{\mathcal{Q}}_{r_q}$ such that $\tilde{\mathcal{P}}_{r_p}$ = rank-$r_p$ CP of $\mathcal{P}$, $\tilde{\mathcal{Q}}_{r_q}$ = rank-$r_q$ CP of $\mathcal{Q}$
3:     Combine $\tilde{\mathcal{P}}_{r_p}$ and $\tilde{\mathcal{Q}}_{r_q}$ on their join factor matrices into $\tilde{\mathcal{X}}_{r_p,r_q}$ using Equation (5)
4:     Compute and record the pair selection measure, $psm(r_p, r_q)$, for $r_p$ and $r_q$
5: end for
6: Return $\tilde{\mathcal{X}}_{r_p,r_q}$ corresponding to $(r_p, r_q)$ with the best $psm(r_p, r_q)$ value
```

Fig. 5 Pseudo-code of JBD-CP algorithm

```
JBD-Tucker algorithm (input: two tensors $\mathcal{P}$, $\mathcal{Q}$, rank $r_{x,1}, ..., r_{x,k}, ..., r_{x,N_x}$, and the modes $i$ and $j$ of the join factor matrices of $\mathcal{P}$ and $\mathcal{Q}$ respectively)
1: for each pair of factors, $r_{p,i}$ and $r_{q,j}$ such that $r_{p,i} \times r_{q,j} = r_{x,k}$ do
2:     Run any available Tucker algorithm to get $\tilde{\mathcal{P}}_{r_{p,1},...,r_{p,N_p}}$ and $\tilde{\mathcal{Q}}_{r_{q,1},...,r_{q,N_q}}$ such that $\tilde{\mathcal{P}}_{r_{p,1},...,r_{p,N_p}}$ = rank-$r_{p,1}, ..., r_{p,N_p}$ Tucker decomposition of $\mathcal{P}$, $\tilde{\mathcal{Q}}_{r_{q,1},...,r_{q,N_q}}$ = rank-$r_{q,1}, ..., r_{q,N_q}$ Tucker decomposition of $\mathcal{Q}$
3:     Combine $\tilde{\mathcal{P}}_{r_{p,1},...,r_{p,N_p}}$ and $\tilde{\mathcal{Q}}_{r_{q,1},...,r_{q,N_q}}$ on their core tensors and their join factor matrices using Equation (6) into $\tilde{\mathcal{X}}_{r_{x,1},...,r_{x,k},...,r_{x,N_x}}$
4:     Compute and record the pair selection measure, $psm(r_{p,i}, r_{q,j})$, for $r_{p,i}$ and $r_{q,j}$
5: end for
6: Return $\tilde{\mathcal{X}}_{r_{x,1},...,r_{x,k},...,r_{x,N_x}}$ where $r_{p,i} \times r_{q,j} = r_{x,k}$ corresponding to $(r_{p,i}, r_{q,j})$ with the best $psm(r_{p,i}, r_{q,j})$ value
```

Fig. 6 Pseudo-code of JBD-Tucker algorithm

the JBD algorithm for CP (JBD-CP) and Tucker decompositions (JBD-Tucker) are shown in Figs. 5 and 6 respectively. Table 1 shows notations used in the paper.

4.1 JBD for CP decomposition (JBD-CP)

Given two tensors, $\mathcal{P}$ and $\mathcal{Q}$, JBD-CP constructs a rank-r CP decomposition of the joined tensor $\mathcal{X} = \mathcal{P} \bowtie \mathcal{Q}$, by selecting two integers, r_1 and r_2, such that $r_1 \times r_2 = r$, obtaining rank-r_1 and rank-r_2 decompositions of the two input tensors, and then combining these two decompositions along the given factor matrix corresponding to the join attribute.

Consider two 3-mode relational tensors, $\mathcal{P}$ and $\mathcal{Q}$, with $u \times l \times m$ and $u \times d \times s$ dimensions, respectively, and an equi-join operation on the first mode of these tensors (note that for simplicity, we assume that both modes have u slices along the join attribute, representing the common values for the two relations along the equi-join attribute). The rank-r_p and rank-r_q CP decompositions of $\mathcal{P}$ and $\mathcal{Q}$ are as follows:

$$\mathcal{P}_{u\times l\times m} \approx \sum_{a=1}^{r_p} \lambda_a \circ U_{:a} \circ L_{:a} \circ M_{:a}, \qquad \mathcal{Q}_{u\times d\times s} \approx \sum_{b=1}^{r_q} \lambda'_b \circ U'_{:b} \circ D_{:b} \circ S_{:b}.$$

When decompositions are nonnegative and the tensors are properly normalized, the equation for $\mathcal{P}$ can be interpreted probabilistically as

Table 1 Notation

Notation	Description
$\mathcal{P}$	A tensor
$\tilde{\mathcal{P}}$	Tensor decomposition of $\mathcal{P}$
$\tilde{\mathcal{P}}_r$	Rank-r CP decomposition of $\mathcal{P}$
$\tilde{\mathcal{P}}_{r_1,r_2,\ldots r_n}$	Rank-$r_1, r_2, \ldots, r_n$ Tucker decomposition of $\mathcal{P}$
$\hat{\mathcal{P}}$	Reconstructed tensor from $\tilde{\mathcal{P}}$
$\\|\mathcal{P}\\|$	The Fronenius norm of $\mathcal{P}$
$V_{i:}$	i-th row vector
$V_{:j}$	j-th column vector
λ	Core of CP decomposition
$\mathcal{G}$	Core of Tucker decomposition
$\mathcal{R}$	A relation
X	Join attribute
A_k	k-th attribute
$\sigma_{i,j}$	Support value of approximate FD between attribute pair (A_i, A_j) (see Definition 4)
$\tau_{support}$	A support lower-bound
FD	Functional dependency
aFD	Approximate FD
paFD	Pairwise approximate FDs
G	A graph

$$\mathcal{P}_{u\times l\times m} \approx \sum_{a=1}^{r_p} P(C_a^p) \sum_{i=1}^{u} P(U_{i:}|C_a^p) \times \sum_{j=1}^{l} P(L_{j:}|C_a^p) \sum_{k=1}^{m} P(M_{k:}|C_a^p).$$

Here C_*^p are the clusters of $\mathcal{P}$; analogously, the equation for $\mathcal{Q}$ can also be interpreted probabilistically. Let us denote the equi-join tensor $\mathcal{P} \bowtie_{=,\mathbf{U}} \mathcal{Q}$ as $\mathcal{X}$. Similarly to the input tensors $\mathcal{P}$ and $\mathcal{Q}$, we can also probabilistically interpret the rank-r decomposition of $\mathcal{X}$:

$$\mathcal{X}_{u\times l\times m\times d\times s} \approx \sum_{c=1}^{r} P(C_c^x) \sum_{i=1}^{u} P(U_{i:}|C_c^x) \times \sum_{j=1}^{l} P(L_{j:}|C_c^x) \sum_{k=1}^{m} P(M_{k:}|C_c^x) \times \sum_{f=1}^{d} P(D_{f:}|C_c^x) \sum_{g=1}^{s} P(S_{g:}|C_c^x),$$

where C^x_* are the clusters of the joined tensor. Note that if the r_p and r_q clusters of the input tensors are independent from each other and $r_p \times r_q = r$, we can rewrite this in terms of the clusters and membership probabilities of the input tensors as

$$\begin{aligned}\mathcal{X}_{u\times l\times m\times d\times s} \approx \hat{\mathcal{X}}_{u\times l\times m\times d\times s} = &\sum_{a=1}^{r_p}\sum_{b=1}^{r_q} P(C^p_a)P(C^q_b)\sum_{i=1}^{u} P(U_{i:}|C^p_a)P(U_{i:}|C^q_b)\\ &\times \sum_{j=1}^{l} P(L_{j:}|C^p_a)\sum_{k=1}^{m} P(M_{k:}|C^p_a)\\ &\times \sum_{f=1}^{d} P(D_{f:}|C^q_b)\sum_{g=1}^{s} P(S_{g:}|C^q_b). \end{aligned} \tag{5}$$

This gives us a way to reconstruct the CP decomposition of the join tensor directly from the CP decompositions of the input tensors, which are much cheaper to obtain. However, this reconstruction makes sense only if the clusters of the input tensors are independent from each other.

$$\begin{aligned} P(C^x_{a,b}) &= P(C^p_a \wedge C^q_b) = P(C^p_a)P(C^q_b),\\ P(U_*|C^x_{a,b}) &= P(U_*|C^p_a \wedge C^q_b) = P(U_{:a}|C^p_a)P(U_{:b}|C^q_b),\\ P(L_*|C^x_{a,b}) &= P(L_*|C^p_a \wedge C^q_b) = P(L_*|C^p_a),\\ P(M_*|C^x_{a,b}) &= P(M_*|C^p_a \wedge C^q_b) = P(M_*|C^p_a),\\ P(D_*|C^x_{a,b}) &= P(D_*|C^p_a \wedge C^q_b) = P(D_*|C^q_b),\\ P(S_*|C^x_{a,b}) &= P(S_*|C^p_a \wedge C^q_b) = P(S_*|C^q_b). \end{aligned}$$

Otherwise, there will be a nonzero difference between $\mathcal{X}$ and $\hat{\mathcal{X}}$. We consider the norm-based pair selection measure defined as

$$psm_{norm}(r_p, r_q) = |\|\mathcal{X}\| - \|\hat{\mathcal{X}}_{r_p,r_q}\||^{-1},$$

that leads to the minimum approximation error, $\|\mathcal{X} - \hat{\mathcal{X}}\|$. Note that this term will be efficiently computed since $\|\hat{\mathcal{X}}\|$ can be computed directly from the decomposition $\tilde{\mathcal{X}}$ as discussed in Sect. 2.1.

The intuition behind the psm_{norm} is as follows: For $\mathcal{W}, \hat{\mathcal{W}} \geq 0$, $\|\mathcal{W} - \hat{\mathcal{W}}\| \geq |\|\mathcal{W}\| - \|\hat{\mathcal{W}}\||$ holds, i.e., while the term $|\|\mathcal{W}\| - \|\hat{\mathcal{W}}\||$ is only a lower bound on $\|\mathcal{W} - \hat{\mathcal{W}}\|$, it may still provide an indication of the size of the term and thus we may be able to minimize the term $\|\mathcal{W} - \hat{\mathcal{W}}\|$ by minimizing $|\|\mathcal{W}\| - \|\hat{\mathcal{W}}\||$.

Note that in general it is possible to find two tensors $\mathcal{X}$ and $\mathcal{Y}$, such that $\|\mathcal{X}\|$ is close to $\|\mathcal{Y}\|$, but $\|\mathcal{X} - \mathcal{Y}\|$ is large. However, in this case, the two tensors are not independent: $\hat{\mathcal{W}}$ is created through $\mathcal{W}$ through a decomposition process. Therefore, it is not likely for the process to return some unrelated $\hat{\mathcal{W}}$, such that $\|\mathcal{W}\|$ is close to $\|\hat{\mathcal{W}}\|$, but $\|\mathcal{W} - \hat{\mathcal{W}}\|$ is large. In other words, by construction $\hat{\mathcal{W}}$ is constrained to be related to $\mathcal{W}$ and the constraint $\|\mathcal{W}\| - \|\hat{\mathcal{W}}\|$ is sufficient, in practice, to help identify good solutions. This is confirmed in experiments reported in Sect. 8.

4.2 JBD for tucker decomposition (JBD-Tucker)

In this subsection, we extend JBD to Tucker decompositions (JBD-Tucker). Similarly to the formulation of JBD-CP, we formulate JBD-Tucker as follows. Consider two 3-mode relational tensors, $\mathcal{P}$ and $\mathcal{Q}$, with $u \times a \times b$ and $u \times d \times e$ dimensions, respectively. The rank-(R_p, S, T) and rank-(R_q, V, W) Tucker decompositions of $\mathcal{P}$ and $\mathcal{Q}$ are as follows:

$$\mathcal{P} \approx \mathcal{G}_p \times_1 \mathbf{U} \times_2 \mathbf{A} \times_3 \mathbf{B} = \sum_{r_p=1}^{R_p} \sum_{s=1}^{S} \sum_{t=1}^{T} \mathcal{G}_p[r_p, s, t]\, U_{:r_p} \circ A_{:s} \circ B_{:t}.$$

$$\mathcal{Q} \approx \mathcal{G}_q \times_1 \mathbf{U}^{'} \times_2 \mathbf{D} \times_3 \mathbf{E} = \sum_{r_q=1}^{R_q} \sum_{v=1}^{V} \sum_{w=1}^{W} \mathcal{G}_q[r_q, v, w]\, U^{'}_{:r_q} \circ D_{:v} \circ E_{:w}.$$

Here, each core tensor of $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{G}_p$ and $\mathcal{G}_q$ respectively, expresses the weight (or strength) of the interaction between the different components. Similarly to CP decomposition in Sect. 4.1, if decompositions are nonnegative and normalized, the Tucker decomposition for $\mathcal{P}$ can be interpreted probabilistically with respect to rank R_p as

$$\mathcal{P} \approx \sum_{r_p=1}^{R_p} P(C^p_{r_p}) \sum_{i=1}^{u} P(U_{i:}|C^p_{r_p}) \sum_{j=1}^{a} A_{j:} \sum_{k=1}^{b} B_{k:},$$

where C^p_* are the clusters of the values of the join attribute for $\mathcal{P}$ and $P(C^p_{r_p}) = \sum_{s=1}^{S} \sum_{t=1}^{T} P(C^p_{r_p} \wedge C^p_s \wedge C^p_t)$. Analogously, the Tucker decomposition for $\mathcal{Q}$ can also be interpreted probabilistically with respect to rank R_q as

$$\mathcal{Q} \approx \sum_{r_q=1}^{R_q} P(C^q_{r_q}) \sum_{i=1}^{u} P(U_{i:}|C^q_{r_q}) \sum_{l=1}^{d} D_{l:} \sum_{m=1}^{e} E_{m:},$$

where C^q_* are the clusters of the values of the join attribute for $\mathcal{Q}$ and $P(C^q_{r_q}) = \sum_{v=1}^{V} \sum_{w=1}^{W} P(C^q_{r_q} \wedge C^q_v \wedge C^q_w)$.

Let us denote the equi-join tensor $\mathcal{P} \bowtie_{=,\mathbf{U}} \mathcal{Q}$ as $\mathcal{X}$. Similarly to the input tensors $\mathcal{P}$ and $\mathcal{Q}$, we can also interpret the rank-(R, S, T, V, W) Tucker decomposition of $\mathcal{X}$ probabilistically with respect to R:

$$\mathcal{X} \approx \hat{\mathcal{X}} = \sum_{r=1}^{R} P(C^x_r) \sum_{i=1}^{u} P(U_{i:}|C^x_r) \sum_{j=1}^{a} A_{j:} \sum_{k=1}^{b} B_{k:} \sum_{l=1}^{d} D_{l:} \sum_{m=1}^{e} E_{m:},$$

where C_*^x are the clusters of the values of the join attribute for the joined tensor $\mathcal{X}$ and $P(C_r^x) = \sum_{s=1}^{S} \sum_{t=1}^{T} \sum_{v=1}^{V} \sum_{w=1}^{W} P(C_r^x \wedge C_s^x \wedge C_t^x \wedge C_v^x \wedge C_w^x)$.

Note that if the R_p and R_q clusters of the input tensors are independent from each other and $R_p \times R_q = R$, we can rewrite this in terms of the clusters and membership probabilities of the input tensors as

$$\mathcal{X} \approx \sum_{r_p=1}^{R_p} \sum_{r_q=1}^{R_q} P(C_{r_p}^p) P(C_{r_q}^q) P(U_{i:}|C_{r_p}^p) P(U_{i:}|C_{r_q}^q) \sum_{j=1}^{a} A_{j:} \sum_{k=1}^{b} B_{k:} \sum_{l=1}^{d} D_{l:} \sum_{m=1}^{e} E_{m:}. \tag{6}$$

Once again, this gives us a way to reconstruct the Tucker decomposition of the join tensor directly from the Tucker decompositions of the input tensors, which are much cheaper to obtain. However, this reconstruction makes sense only if the clusters of the input tensors are independent from each other:

$$P(C_{r_p,r_q}^x) = P(C_{r_p}^p \wedge C_{r_q}^q) = P(C_{r_p}^p) P(C_{r_q}^q),$$
$$P(U_*|C_{r_p,r_q}^x) = P(U_*|C_{r_p}^p \wedge C_{r_q}^q) = P(U_*|C_{r_q}^p) P(U_*|C_{r_q}^q).$$

Otherwise, there will be a nonzero difference between $\mathcal{X}$ and $\hat{\mathcal{X}}$. As in JBD-CP, we employ the norm-based pair selection (psm_{norm}) method for selecting the rank-$(\ldots, R_p, \ldots)$ and rank-$(\ldots, R_q, \ldots)$ Tucker decompositions of $\mathcal{P}$ and $\mathcal{Q}$. Again, $\|\hat{\mathcal{X}}\|$ can be computed directly from the decomposition $\tilde{\mathcal{X}}$, thus psm_{norm} is computed much more efficiently than $\|\mathcal{X} - \hat{\mathcal{X}}\|$. Also psm_{norm} approximates the fit error in JBD-Tucker.

Although the JBD algorithm is developed for nonnegative CP and Tucker decompositions, it works well also for general tensor decompositions involving negative values. For this purpose, we relax this requirement and experiment with general tensor decompositions for JBD and DBN, which we will show in Sect. 8.

5 Vertical data partitioning

As discussed in Sect. 3.2, a significant challenge that DBN has to address is to partition the input data into two in such a way that they can be recombined effectively through the JBD process introduced in the previous section. In this section, we discuss vertical partitioning strategies for CP and Tucker decompositions. Below we first list the key desiderata that govern how the DBN algorithm makes the partitioning decision.

- *Desideratum 1* As we discussed above, when we need to use approximate FDs when partitioning the input data, this may result in the removal of outlier tuples to preserve the semantics of the FDs. Therefore, to prevent over-thinning of the relation $\mathcal{R}$, the considered approximate FDs need to have few outliers and high support; i.e., $\sigma_{i,j} \geq \tau_{support}$, for a sufficiently large support lower-bound, $\tau_{support}$.

Secondly, when we vertically partition the relation $\mathcal{R}$ with attributes $\mathbb{A} = \{A_1, \ldots, A_n\}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$, one of the attributes (X) of $\mathcal{R}_2$ should serve as a foreign key into

$\mathcal{R}_1$ to ensure that joining of the vertical partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ (on X) gives back $\mathcal{R}$ without missing any tuples or introducing any spurious ones.

– *Desideratum 2* If $\mathbb{A}_1$ is the set of attributes of vertical partition $\mathcal{R}_1$ and $\mathbb{A}_2$ is the set of attributes of vertical partition $\mathcal{R}_2$, then there must be an attribute $X \in \mathbb{A}_2$, such that for each attribute $Y \in \mathbb{A}_1$, $X \overset{\sigma}{\rightarrow} Y$, for $\sigma \geq \tau_{support}$.

Since the overall size (in terms of modes and their dimensionalities) of the input tensor is a major cost factor for dense (for CP and Tucker decompositions) or Tucker decomposing sparse tensors, we prefer that the partitions are balanced in terms of their dimensionalities.

– *Desideratum 3* For dense (CP and Tucker decompositions) and sparse (Tucker decomposition), vertical partitioning should be such that the sizes of $\mathcal{R}_1$ and $\mathcal{R}_2$ are similar.

When CP decomposing sparse tensors, the major contributor to the decomposition cost is the number of nonzero entries in the tensor.

– *Desideratum 4* For CP decomposition of sparse tensors, vertical partitioning should be such that the total number of tuples of $\mathcal{R}_1$ and $\mathcal{R}_2$ are minimized.

Any information encoded by the FDs crossing the two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ is potentially lost when $\mathcal{R}_1$ and $\mathcal{R}_2$ are individually decomposed. This leads to our final desideratum:

– *Desideratum 5* The vertical partitioning should be such that the support for the inter-partition FDs (except for the FDs involving the join attribute X) are minimized.

5.1 Overview of the partitioning strategies

We use different strategies to satisfy the above desiderata depending on whether we work on sparse or dense tensors and whether we seek CP or tucker decompositions:

- **Case 1: CP Decomposition on Sparse Tensors** This case has two subcases:
  - *Case 1.1: Exact Functional Dependencies* When the join attribute X determines all attributes of $\mathcal{R}$, we apply the interFD-based vertical partitioning strategy detailed in Sect. 5.2.
  - *Case 1.2: Approximate Functional Dependencies* When the join attribute X approximately determines a subset of the attributes of $\mathcal{R}$, we create a partition $\mathcal{R}_1$ with all the attributes determined with a support higher than the threshold ($\tau_{support}$) by the join attribute. This helps us satisfy Desiderata 1 and 2. The second partition, $\mathcal{R}_2$, consists of the join attribute X and all the remaining attributes.
    Note that, since we can include any attribute in $\mathcal{R}_1$ as long as it is determined by X, there may be still multiple ways to partition the data. Therefore, we apply the interFD-based partitioning strategy discussed in Sect. 5.2 to choose the two partitions. Note also that, the size of $\mathcal{R}_2$ is, by construction, equal to the number of tuples in $\mathcal{R}$ independent of which attributes are included in it.

On the other hand, the size of $\mathcal{R}_1$ can be reduced down to the number of unique values of X by eliminating duplicate tuples (to satisfy Desideratum 4).

– **Case 2: CP Decomposition on Dense Tensors or Tucker Decomposition** When we are operating on dense tensors or when we seek Tucker decompositions of sparse or dense tensors, we consider Desideratum 3, which prefers balanced partitions as discussed in Sect. 5.2. When there are alternative balanced partitioning cases, we apply the interFD-based vertical partitioning strategy to break ties, which is also discussed in Sect. 5.2.

5.2 InterFD criterion and vertical partitioning algorithms

5.2.1 Minimizing the likelihood of decomposition errors

As discussed in Sect. 4, given a partitioning, $\mathcal{R} = \mathcal{R}_1 \bowtie_A \mathcal{R}_2$, the accuracy of the decomposition is likely to be high if the non-join attributes of the two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ are independent from each other. Building on this observation (which we also validate in Sect. 8), DBN tries to partition the input relational tensor $\mathcal{R}$ in such a way that the resulting partitions, $\mathcal{R}_1$ and $\mathcal{R}_2$, are as independent from each other as possible. We refer to this as the InterFD criterion.

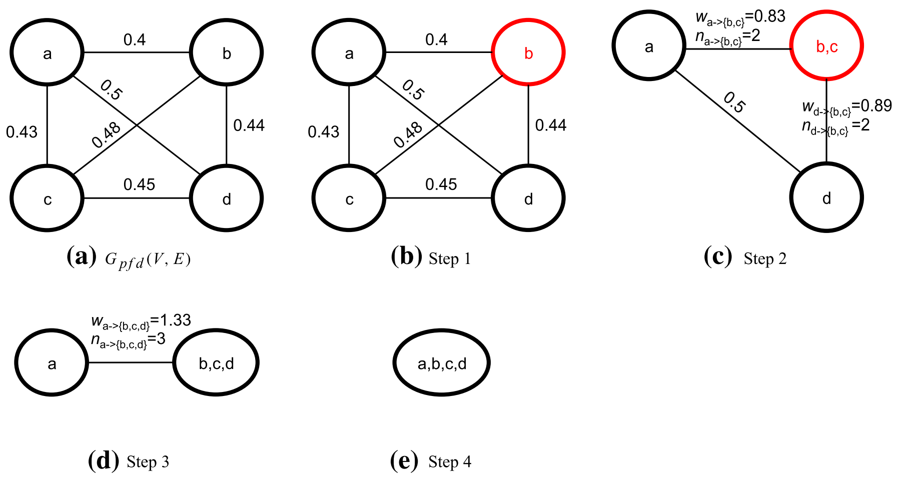
Remember that the support of an approximate FD is defined as the percentage of tuples in the data set for which the FD holds. Thus, in order to quantify the dependence of pairwise attributes, we rely on the supports of pairwise FDs. Since we have two possible FDs ($X \rightarrow Y$ and $Y \rightarrow X$) for each pair of attributes, we use the average of the two as the overall support of the pair of attributes X and Y. Given these pairwise supports, we approximate the overall dependency between two partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ using the average support of the pairwise FDs (excluding the pairwise FDs involving the join attribute) crossing the two partitions.

Let the pairwise FD graph, $G_{pfd}(V, E)$, be a complete, weighted, and undirected graph, where:

– each vertex $v \in V$ represents an attribute (mode),
– the size of the domain (dimensionality) of the mode corresponding to vertex, v, is represented as a weight of the vertex, w_v, and
– the weight, w_e, of the edge e between nodes v_i and v_j is the average support of the approximate FDs $v_i \rightarrow v_j$ and $v_j \rightarrow v_i$.

We argue that the interFD-based vertical data partitioning problem can be formulated in terms of locating a cut on G_{pfd} with the minimum average weight. To solve the problem efficiently, we extend the minimum *total* weighted cut algorithm presented in Stoer and Wagner (1997) to identify the minimum *average* weight. The overall process is similar to that presented in Stoer and Wagner (1997) and has the same time complexity of complexity, $O(|V||E| + |V|^2 log|V|)$:

Given an undirected graph $G_{pfd}(V, E)$, the algorithm copies V into V', where each edge $e \in E$ is annotated with a counter n_e initially set to 1. The algorithm then first picks a vertex v with the cut with the minimum average weight. We compute the average edge weight of a cut between a set of vertices S and $V \backslash S$, denoted by $\bar{w}_S$, such that



0.4
0.5
0.43
0.48
0.44
0.45
a
b
c
d
(a) $G_{pfd}(V, E)$
(b) Step 1
$w_{a\text{->}\{b,c\}}$=0.83
$n_{a\text{->}\{b,c\}}$=2
b,c
$w_{d\text{->}\{b,c\}}$=0.89
$n_{d\text{->}\{b,c\}}$=2
(c) Step 2
$w_{a\text{->}\{b,c,d\}}$=1.33
$n_{a\text{->}\{b,c,d\}}$=3
b,c,d
a,b,c,d
(d) Step 3
(e) Step 4

Fig. 7 An example of the minimum average cut algorithm for $G_{pfd}(V, E)$ (see Example 2)

$$\bar{w}_S = \sum w_e / \sum n_e, \text{ for } e \in \{(v_1, v_2) \in E | v_1 \in S, v_2 \in V \backslash S\}. \tag{7}$$

Then, the algorithm selects a neighbor v' of v such that $\{v, v'\}$ has a cut from $V' \backslash \{v, v'\}$ with the smallest average weight. The algorithm shrinks V' by merging v and v' into a new vertex, v''. Any pair of edges $e = a \rightarrow v$ and $e' = a \rightarrow v'$ originating from the same vertex a is replaced by a new edge $e'' = a \rightarrow v''$, where $w''_e = w_e + w'_e$ and $n''_e = n_e + n'_e$. Any other edge to v or v' is simply re-routed to v''. The process is stopped when $|V'| = 1$. The minimum of the minimum average cuts at each step of the algorithm is returned as the minimum average cut. The following example shows how the minimum average cut algorithm runs on a graph step by step.

Example 2 Consider the graph $G_{pfd}(V, E)$ in Fig. 7a. Initially, as shown in Fig. 7a, the weight of each edge is assigned with the average support of pairwise approx. FDs.

- Step 1: Among the vertices a, b, c, and d ($\bar{w}_{\{a\}} = (0.4 + 0.5 + 0.43)/3 = 0.443$, $\bar{w}_{\{b\}} = (0.4 + 0.48 + 0.44)/3 = 0.44$, $\bar{w}_{\{c\}} = (0.43 + 0.48 + 0.45)/3 = 0.45$, and $\bar{w}_{\{d\}} = (0.44 + 0.5 + 0.45)/3 = 0.46$), the minimum average cut is the cut between $\{b\}$ and $\{a,c,d\}$ with the average weight ($\bar{w}_{\{b\}} = 0.44$) (see Fig. 7b).
- Step 2: Vertex b is merged with vertex c into $\{b, c\}$ with the smallest average weight among vertices a ($\bar{w}_{\{a,b\}} = (0.43 + 0.48 + 0.5 + 0.44)/4 = 0.46$), c ($\bar{w}_{\{b,c\}} = (0.43+0.4+0.45+0.44)/4 = 0.43$), and d ($\bar{w}_{\{b,d\}} = (0.45+0.48+0.5+0.4)/4 = 0.46$). Edges $a \rightarrow b$ and $a \rightarrow c$ are replaced by the edge $a \rightarrow \{b, c\}$ with weight $w_{a\rightarrow\{b,c\}} = 0.4+0.43 = 0.83$ and counter $n_{a\rightarrow\{b,c\}} = 2$. Edges $d \rightarrow b$ and $d \rightarrow c$ are replaced by the edge $d \rightarrow \{b, c\}$ with weight $w_{d\rightarrow\{b,c\}} = 0.44 + 0.45 = 0.89$ and counter $n_{d\rightarrow\{b,c\}} = 2$. The minimum average cut is the cut between $\{b,c\}$ and $\{a,d\}$ with the average weight ($\bar{w}_{\{b,c\}} = 0.43$) (see Fig. 7c).
- Step 3: $\{b, c\}$ is merged with vertex d into $\{b, c, d\}$ with the smallest average weight among vertices a ($\bar{w}_{\{a,b,c\}} = (0.89 + 0.5)/3 = 0.46$) and d ($\bar{w}_{\{b,c,d\}} = (0.83+0.5)/3 = 0.44$). Edges $a \rightarrow \{b, c\}$ and $a \rightarrow d$ are replaced by

the edge $a \rightarrow \{b, c, d\}$ with weight $w_{a\rightarrow\{b,c,d\}} = 0.83 + 0.5 = 1.33$ and counter $n_{a\rightarrow\{b,c,d\}} = 3$. The cut between $\{a\}$ and $\{b, c, d\}$ is the last cut with weight $\bar{w}_{\{b,c,d\}} = 0.44$ (see Fig. 7d).

– Step 4: The process ends since $|V'| = 1$ (see Fig. 7e). The minimum of the minimum average cuts at each step is $\{b, c\}$ and $\{a, d\}$ in Step 2.

5.2.2 *Balanced partitioning*

When targeting Desideratum 3, we seek a balanced partitioning of the attributes. Unfortunately, the general problem of obtaining balanced partitions is an NP-complete problem even for simple sets of values (Garey and Johnson 1979). While there are various approximation and heuristic algorithms including (Karmarker and Karp 1983), applying these directly would only optimize balance, but ignore other criteria. We therefore choose the average cut based partitioning scheme discussed above in a way that also considers balance of attributes. In particular, we associate a *balance score* to each vertex, v:

$$balance\_score(v) = \frac{max\{size(V_v), size(V \backslash V_v)\}}{min\{size(V_v), size(V \backslash V_v)\}}, \tag{8}$$

where V_v is the set of original vertices merged into v (if v is a original vertex, then V_v is $\{v\}$) and $size(V_v)$ is $\prod_{v_{org} \in V_v} w_{v_{org}}$ ($w_{v_{org}}$ is the weight of v_{org}), and minimize the balance score as the vertices are merged in a similar manner that interFD-based vertex partitioning minimizes the average edge weight. Note that the balance score will be 1.0 for the most balanced cut and the higher the score is, the less the partitions are balanced. Instead of using average weights as in interFD based partitioning, we now select the next cut based on the resulting balance score.

Given an undirected graph $G_{pfd}(V, E)$, the algorithm makes a copy V' or V and first picks the vertex v with the *minimum balance score*, among all vertices in V'. If there are multiple alternatives, then the algorithm selects the one which has the cut with the minimum average weight among the alternatives. Then, the algorithm selects a neighbor v' of v such that $\{v, v'\}$ has the smallest balance score; again, if there are alternatives, then the algorithm selects the neighbor such that $\{v,v'\}$ has a cut from $V' \backslash \{v, v'\}$ with the smallest average weight. The algorithm then shrinks V' by merging v and v' into a new vertex, v'', with the vertex weight, $w_{v''} = w_v \times w_{v'}$ and the balance score of v'' is computed using Eq. 8. For any pair of edges $e = a \rightarrow v$ and $e' = a \rightarrow v'$ originating from the same vertex a, we create a new edge $e'' = a \rightarrow v''$, where the edge weight $w''_e = w_e + w'_e$ and counter $n''_e = n_e + n'_e$. Any other edge to v or v' is simply re-routed to v''. The process is stopped when the balance score is 1.0 or $|V'| = 1$. The most balanced cut among the most balanced cuts of each step is returned. The following is an example of the balanced cut algorithm.

Example 3 For the balanced cut algorithm, we consider the graph $G_{pfd}(V, E)$ with weighted edges and weighted vertices (see Fig. 8a).

– Step 1: The cut between $\{b\}$ and $\{a, c, d\}$ with the minimum average cut is chosen out of the two alternative cuts, (1) $\{b\}$ and $\{a, c, d\}$ ($\bar{w}_{\{b\}} = 0.44$) and (2) $\{c\}$

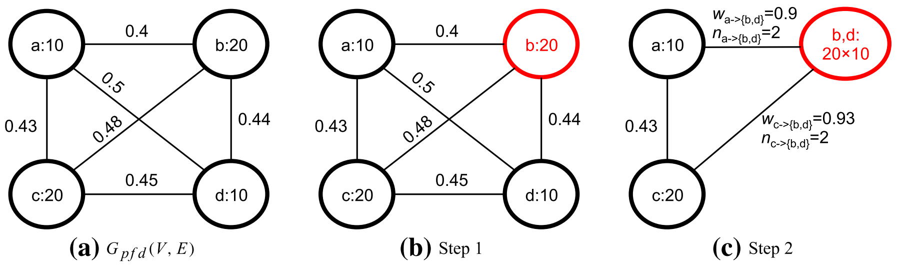


Fig. 8 An example of the balanced cut algorithm for $G_{pfd}(V, E)$ with weighted edges and weighted vertices (see Example 3)

and $\{a, b, d\}$ ($\bar{w}_{\{c\}} = 0.45$), with the equal balance score, (10 × 20 × 10) / 20 (see Fig. 8b).

– Step 2: The vertex d is chosen with the minimum average weight out of two neighbors of the vertex b, (1) vertex a ($\bar{w}_{\{a,b\}} = (0.43 + 0.48 + 0.5 + 0.44)/4 = 0.463$) and (2) vertex d ($\bar{w}_{\{b,d\}} = (0.45 + 0.48 + 0.5 + 0.4)/4 = 0.458$) with the equal smallest balance score 1.0 and merged with the vertex b into $\{b, d\}$ with the vertex weight (20 × 10). Edges $a \to b$ and $a \to d$ are replaced by the edge $a \to \{b, d\}$ with weight $w_{a\to\{b,d\}} = 0.4 + 0.5 = 0.9$ and counter $n_{a\to\{b,d\}} = 2$. Edges $c \to b$ and $c \to d$ are replaced by the edge $c \to \{b, d\}$ with weight $w_{c\to\{b,d\}} = 0.48 + 0.45 = 0.93$ and counter $n_{c\to\{b,d\}} = 2$. The cut between $\{a, c\}$ and $\{b, d\}$ is the most balanced cut (balance score: 1.0) with the minimum average weight ($\bar{w}_{\{b,d\}} = 0.458$); the process ends since the balance score is 1.0 (see Fig. 8c).

6 Further optimizations: rank pruning based on intra-partition dependencies

As discussed in the previous section, given a partitioning of $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$, to obtain a rank-r decomposition of $\mathcal{R}$ using JBD, we need to consider rank-r_1 and rank-r_2 decompositions of $\mathcal{R}_1$ and $\mathcal{R}_2$, such that $r = r_1 \times r_2$ and pick the (r_1, r_2) pair which is likely to minimize recombination errors. In this section, we argue that we can rely on the supports of the dependencies that make up the partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ to prune (r_1, r_2) pairs which are not likely to give good fits. In particular, we observe that the higher the overall dependency between the attributes that make up a partition, the more likely the data in the partition can be described with a smaller number of clusters. Since the number of clusters of a data set is related to the rank of the decomposition, this leads to the observation that the higher the overall dependency between the attributes in a partition, the smaller should be the decomposition rank of that partition.

Thus, given $\mathcal{R}_1$ and $\mathcal{R}_2$, we need to consider only those rank pairs (r_1, r_2), where if the average intra-partition FD support for $\mathcal{R}_1$ is larger than the support for $\mathcal{R}_2$, then $r_1 < r_2$ and vice versa. We refer to this as the *intraFD* criterion for rank pruning. Similarly to interFD, given the supports of FDs, we define intraFD as the average support of the pairwise FDs (excluding the pairwise FDs involving the join attribute) within each partition. In Sect. 8, we evaluate the effect of the interFD-based partitioning

Table 2 Notation used for cost analysis

Notation	Description
$\mathcal{X}$	The input tensor of size $K_1 \times K_2 \times \cdots \times J \times \cdots \times K_{N_x}$ let $\mathcal{X}$ be partitoned into $\mathcal{P}$ and $\mathcal{Q}$ on the join mode $\mathbf{J}$ of size J; i.e., $\mathcal{P} \bowtie_{=,\mathbf{J}} \mathcal{Q}$
J	The size of the join mode $\mathbf{J}$
$\mathcal{P}$	The 1st partition tensor of size $I_1 \times I_2 \times \cdots \times J \times \cdots \times I_{N_p}$
$\mathcal{Q}$	The 2nd partition tensor of size $I'_1 \times I'_2 \times \cdots \times J \times \cdots \times I'_{N_q}$
r	The rank of $\mathcal{X}$
$r_{p,i}$ and $r_{q,i}$	The ith ranks of $\mathcal{P}$ and $\mathcal{Q}$, resp.; i.e., $(r_{p,i}, r_{q,i}) \in \{(r_{p,i}, r_{q,i}) \mid r_{p,i} \times r_{q,i} = r\}$
N_p	# of modes of $\mathcal{P}$
N_q	# of modes of $\mathcal{Q}$
N_x	# of modes of $\mathcal{X}$
$\alpha_{r,*}$	# of ALS iterations needed for the rank-r CP decomposition of the tensor denoted by "*"
$\lvert\mathcal{P}\rvert$	# of nonzero entries of a tensor $\mathcal{P}$
$\lvert\mathcal{Q}\rvert$	# of nonzero entries of a tensor $\mathcal{Q}$
$\lvert\mathcal{X}\rvert$	# of nonzero entries of a tensor $\mathcal{X}$
n_{pair}	# of (r_p, r_q); i.e., $\lvert\{(r_{p,i}, r_{q,i}) \mid r_{p,i} \times r_{q,i} = r\}\rvert$
ϕ	Join selectivity
$\phi_{\perp}$	A lower bound of join selectivity

and intraFD-based rank pruning strategy of DBN for both dense and sparse tensor decomposition in terms of the efficiency and the accuracy.

7 Cost analysis

In this section, we provide cost analyses for decomposition-by-normalization strategies for CP and Tucker decompositions (DBN-CP and DBN-Tucker, respectively).

Unlike the conventional tensor decomposition process, DBN involves a data partitioning (normalization) step followed by a series of partial decompositions, joins, and candidate selection steps (see Sect. 3.2): For a target r decomposition, DBN performs as many partial decompositions as the number, n_{pair}, of rank pairs (r_p, r_q) where $r = r_p \times r_q$. Join and norm-based candidate selection steps of the DBN are also performed once for each pair (r_p, r_q). Since these costs are negligible compared to the tensor decomposition cost, in this section, we focus on tensor compositions cost.

7.1 Cost of DBN-CP

Table 3 presents an overview of the execution times for conventional CP (simply called CP in the rest of this section) and DBN-CP. Symbols used in this section are introduced in Table 2.

Table 3 Execution time cost for CP decomposition

Algorithm	Cost	
CP	Dense tensors	$O(\prod_{i=1}^{N_x} K_i)$[a]
	Sparse tensors	$O(\alpha_{r,\mathcal{X}}\ r\ \lvert\mathcal{X}\rvert\ N_x)$[b]
lDBN-CP	Dense tensors	$O\left(n_{pair}\left(\prod_{i=1}^{N_p} I_i + \prod_{i=1}^{N_q} I'_i\right)\right)$[a]
	Sparse tensors	$O\left(\sum_{i=1}^{n_{pair}}\left(\alpha_{r_{p,i},\mathcal{P}}\ r_{p,i}\ \lvert\mathcal{P}\rvert\ N_p + \alpha_{r_{q,i},\mathcal{Q}}\ r_{q,i}\ \lvert\mathcal{Q}\rvert\ N_q\right)\right)$[b]

[a] The execution time cost for dense tensors is based on Sun et al. (2008)
[b] The execution time cost for sparse tensors is based on the analysis of the code in Bader and Kolda (2007)

7.1.1 CP decomposition of dense tensors

As we see in Table 3, For dense tensors, the DBN strategy increases the number of decomposition operations from one to n_{pair} (the number of rank-pairs), but each decomposition involves smaller numbers of modes. Since, in the case of dense tensors, the cost of the CP decomposition is exponential in the numbers of modes and since DBN-CP reduces the number of modes that need to be considered, as we experimentally observe in Sect. 8, DBN-CP is more efficient than the conventional CP decomposition.

7.1.2 CP decomposition of sparse tensors

As we also see in Table 3, the execution time cost of the conventional CP operation for sparse tensors depends on the rank, the number of nonzero entries, the number of modes, as well as the number of alternating least squares (ALS) iterations (Bader and Kolda 2007).

When all things equal, the main contributor to the cost of the CP decomposition on sparse tensors is the number of nonzero entries. Therefore, in order to predict whether CP or DBN-CP will be more efficient, we need to consider the number of nonzero entries in the input tensors: In particular, if the ratio

$$\phi = \lvert\mathcal{X}\rvert/(\lvert\mathcal{P}\rvert\lvert\mathcal{Q}\rvert)$$

is high and we have more tuples (nonzero entries) in the input tensor than the partition tensors, then DBN-CP is likely to be more efficient than the CP; otherwise, CP may be competitive. In other words, DBN-CP is likely to outperform CP if the following holds:

$$r\lvert\mathcal{X}\rvert N_x > \sum_{i=1}^{n_{pair}}(r_{p,i}\lvert\mathcal{P}\rvert N_p + r_{q,i}\lvert\mathcal{Q}\rvert N_q),$$

or, equivalently,

$$\lvert\mathcal{X}\rvert > \sum_{i=1}^{n_{pair}}(r_{p,i}\lvert\mathcal{P}\rvert N_p + r_{q,i}\lvert\mathcal{Q}\rvert N_q)/(rN_x).$$

Table 4 Notation for Tucker decomposition

Notation	Description
$r_{x,1}, \dots, r_{x,N_x}$	Decomposition ranks for $\mathcal{X}$
$r_{p,1}, \dots, r_{p,N_p}$	Decomposition ranks for $\mathcal{P}$
$r_{q,1}, \dots, r_{q,N_q}$	Decomposition ranks for $\mathcal{Q}$
$r_{p,i,l}$, $r_{q,j,l}$, and $r_{x,k}$	The lth rank pair $(r_{p,i,l}, r_{q,j,l})$ of the join modes (ith and jth modes) of $\mathcal{P}$ and $\mathcal{Q}$ and the rank $(r_{x,k})$ of the join mode (kth mode) of $\mathcal{X}$; i.e., $(r_{p,i,l}, r_{q,j,l}) \in \{(r_{p,i,l}, r_{q,j,l}) \mid r_{p,i,l} \times r_{q,j,l} = r_{x,k}\}$
n_{pair}	# of $(r_{p,i,l}, r_{q,j,l})$; i.e., $\lvert\{(r_{p,i,l}, r_{q,j,l}) \mid r_{p,i,l} \times r_{q,j,l} = r_{x,k}\}\rvert$
$\beta_{(r_1,\dots r_N),*}$	# of ALS iterations needed for the rank-$(r_1, \dots, r_N)$ Tucker decomposition of the tensor denoted by "*"
ε_*	A subset of modes that are computed element-wise in MET for the tensor denoted by "*"
$C_{m,*}$	The eigen decomposition cost for the mth mode of the tensor denoted by "*" [a]

[a] For eigen decomposition, we assume that MATLAB's `eigs` function based on ARPACK uses an iterative power method to identify eigenvalues. Therefore, the overall eigen decomposition cost is a function of this iteration count

Since we have $|\mathcal{X}| = |\mathcal{P} \bowtie_{=,\mathbf{J}} \mathcal{Q}| = \phi|\mathcal{P}||\mathcal{Q}|$, we can rewrite the above inequality as

$$\phi(|\mathcal{P}||\mathcal{Q}|) > \sum_{i=1}^{n_{pair}} (r_{p,i}|\mathcal{P}|N_p + r_{q,i}|\mathcal{Q}|N_q)/(rN_x).$$

This gives us a lower bound, $\phi_\perp$, on the join selectivity:

$$\phi > \phi_\perp = \sum_{i=1}^{n_{pair}} (r_{p,i}|\mathcal{P}|N_p + r_{q,i}|\mathcal{Q}|N_q)/(|\mathcal{P}||\mathcal{Q}|rN_x).$$

This lower bound threshold provides a practical predictor to judge whether DBN-CP will be more advantageous, for sparse tensors, than CP.

7.2 Cost of DBN-tucker

Table 4 lists additional notations needed for the analysis of the Tucker decomposition costs.

The main cost of Tucker decomposition for dense tensors is the number of modes, which is similar to CP decomposition for dense tensors. Therefore, the costs analysis for DBN-Tucker on dense tensors also follows the cost analysis of DBN-CP on dense tensors presented in Table 3.

Table 5 Execution time cost for Tucker decomposition on sparse tensors

Algorithm	Cost	
MET	$TTM_x(m)$[a]	$O(\sum_{m' \neq m}(\lvert\mathcal{X}\rvert K_{m'} r_{x,m'}) \prod_{m' \neq \varepsilon_x} r_{x,m'})$
	$SVD_x(m)$[a]	$O(K_m^2 \times \prod_{m' \neq m} r_{x,m'} + C_{m,\mathcal{X}})$
	Total	$O(\beta_{(r_{x,1},\ldots,r_{x,N_x}),\mathcal{X}} \sum_{m=1}^{N_x}(TTM_x(m) + SVD_x(m))$
DBN-Tucker (using MET)	$TTM_p(m)$[a]	$O(\sum_{m' \neq m}(\lvert\mathcal{P}\rvert I_{m'} r_{p,m'}) \prod_{m' \neq \varepsilon_p} r_{p,m'})$
	$SVD_p(m)$[a]	$O(I_m^2 \times \prod_{m' \neq m} r_{p,m'} + C_{m,\mathcal{P}})$
	$TTM_q(m)$[a]	$O(\sum_{m' \neq m}(\lvert\mathcal{Q}\rvert I'_{m'} r_{q,m'}) \prod_{m' \neq \varepsilon_q} r_{q,m'})$
	$SVD_q(m)$[a]	$O(I'^2_m \times \prod_{m' \neq m} r_{q,m'} + C_{m,\mathcal{Q}})$
	Total	$O(\sum_{l=1}^{n_{pair}}(\beta_{(r_{p,1},\ldots,r_{p,i,l},\ldots,r_{p,N_p}),\mathcal{P}} \sum_{m=1}^{N_p}(TTM_p(m) + SVD_p(m))$ $+ \beta_{(r_{q,1},\ldots,r_{q,j,l},\ldots,r_{q,N_q}),\mathcal{Q}} \sum_{m=1}^{N_q}(TTM_q(m) + SVD_q(m))$

[a] MET algorithm we consider consists of two major steps applied to each mode m: (a) TTM computation and (b) SVD computation; see Kolda and Sun (2008) for details

Table 6 Bottleneck memory cost for Tucker decomposition

Algorithm	Cost
MET	$O\left(\max_{\varepsilon_x}\left(\prod_{m \notin \varepsilon_x} K_m\right)\right)$[a]
DBN-Tucker (using MET)	$O\left(\max\left(\max_{\varepsilon_p}\left(\prod_{m \notin \varepsilon_p} I_m\right), \max_{\varepsilon_q}\left(\prod_{m \notin \varepsilon_q} I'_m\right)\right)\right)$

[a] The costs are based on Kolda and Sun (2008)

The cost analysis for sparse tensors, on the other hand, is more complex. We focus on Tucker decomposition for sparse tensors.

In Table 5, we present the cost analysis of DBN-Tucker on sparse tensors assuming that it is build on MET (Bader and Kolda 2006). As before, DBN-Tucker, involves as many partial Tucker decompositions as the number, n_{pair}, of rank pairs (r_p, r_q) where $r = r_p \times r_q$, but each decomposition involves smaller number of modes. Since, as we see in Table 5, the cost of Tucker tensor decomposition is exponential in the number of modes, we expect that DBN-Tucker will be more efficient than conventional Tucker decompositions. Experiment results reported in Sect. 8 verify this.

Note that as reported in Table 6, one major benefit for the proposed DBN based Tucker decomposition scheme is that the size of the intermediate results is smaller than that for conventional Tucker decomposition: this is because DBN decomposes smaller sub-tensors. Since a major challenge in Tucker decompositions is the memory needed to store the intermediary results, for large data sets, and especially when the available memory is limited, DBN-Tucker is likely to be more advantageous. We experimentally verify this in Sect. 8.

7.3 Additional discussions

In this section, we discuss additional properties and extensions of the DBN strategy.

7.3.1 DBN with tight memory constraints

The main aim of DBN is to reduce the cost of tensor decomposition by reducing the number of modes of the input tensor. However, there can still be cases in which, the input tensor does not fit in memory thus a straight-forward application of DBN is infeasible. In this case, however, we can split the tensor on each mode into multiple blocks and run a block-based tensor decomposition scheme, such as Phan and Cichocki (2011), where tensor decomposition on each block leverages DBN.

7.3.2 DBN on multi-attribute dependencies

Now we consider cases where a single attribute is not enough to determine other attributes, but two attributes together can determine the rest. In Sect. 4.1, we formulated CP decomposition of a tensor $\mathcal{P}$ of size $u \times l \times m$ as following.

$$\mathcal{P}_{u\times l\times m} \approx \sum_{a=1}^{r_p} \lambda_a \circ U_{:a} \circ L_{:a} \circ M_{:a},$$

which can be interpreted probabilistically as

$$\mathcal{P}_{u\times l\times m} \approx \sum_{a=1}^{r_p} P(C_a^p) \sum_{i=1}^{u} P(U_{i:}|C_a^p) \times \sum_{j=1}^{l} P(L_{j:}|C_a^p) \sum_{k=1}^{m} P(M_{k:}|C_a^p).$$

Let us assume $\mathcal{P}$ has a compound key that consists of two attributes, U and L and represent $\mathcal{P}$ as another tensor $\mathcal{P}'$ of size $w \times m$ where $w = u \times l$, in which the compound key is represented as a single attribute, W. Then the CP decomposition of $\mathcal{P}'$ can be formulated as

$$\mathcal{P}'_{w\times m} \approx \sum_{a=1}^{r_p} \lambda_a \circ W_{:a} \circ M_{:a},$$

and probabilistically,

$$\mathcal{P}'_{w\times m} \approx \sum_{a=1}^{r_p} P(C_a^p) \sum_{i=1}^{w} P(W_{i:}|C_a^p) \times \sum_{k=1}^{m} P(M_{k:}|C_a^p).$$

We can formulate the other tensor, $\mathcal{Q}$, similarly.

Given these, we can approximately reconstruct the factor matrices of each original attribute of the compound key, U and L, from the factor matrix of W. Since $P(W_{k:}|C^p_*) = P(U_{i:}, M_{j:}|C^p_*)$ where $k = (i-1) \times u + j$ for $i = 1..u$ and $j = 1..l$, the factor matrices of the attributes U and L are obtained by marginal probabilities

$$P(U_{i:}|C^p_*) = \sum_{j=1}^{l} P(W_{\{(i-1)\times u+j\}:}|C^p_*)$$

and

$$P(L_{j:}|C^p_*) = \sum_{i=1}^{u} P(W_{\{(i-1)\times u+j\}:}|C^p_*),$$

respectively.

7.3.3 Uniqueness

In Sect. 4, we introduced a method to determine a rank pair with minimum approximation error for DBN. Now we discuss whether tensor decompositions with these chosen ranks on each of tensors are guaranteed in terms of uniqueness.

When one of the partitions in DBN is a matrix, then its decomposition is a bilinear decomposition, which may not be unique. However, in this case, it is possible to add additional constraints, such as orthogonality constraints, for the bilinear decomposition to make decomposition unique (Kolda and Bader 2009).

When one of the partitions in DBN is a tensor with 3 or more modes, we can use Kruskal's uniqueness condition (Kruskal 1977): in CP, the component matrices A, B and C are essentially unique if

$$k_A + k_B + k_C >= 2R + 2,$$

where k_A, k_B and k_C are the k-ranks of A,B and C, respectively, R is the target rank and $k_A, k_B, k_C \leq R$. Here the k-rank of a matrix is defined as the largest value of m such that every subset of m columns of the matrix is linearly independent. When one of the partial decomposition does not hold this condition, the final decomposition is not guaranteed to be unique. Therefore, we recommend avoiding rank pairs for which this condition does not hold to help guarantee uniqueness of the final decomposition.

8 Experimental evaluations

In this section, we present the result of the experiments we have carried out to assess the efficiency and effectiveness of the decomposition-by-normalization (DBN) strategy. We consider both CP and Tucker decompositions and use both sparse and dense tensors. We ran our experiments on a 6-core Intel(R) Xeon(R) CPU X5355 @ 2.66GHz machine with 24GB of RAM.

Table 7 Relational tensor data sets with the same number of nonzero entries for partitions for DBN

Data set		Size	# nonzero
D1	Adult	$118 \times 90 \times 20{,}263 \times 5 \times 2$	20,263
D2		$7 \times 20{,}263 \times 5 \times 6 \times 16$	20,263
D3		$72 \times 20{,}263 \times 90 \times 2 \times 2$	20,263
D4		$20{,}263 \times 14 \times 2 \times 6 \times 94$	20,263
D5		$20{,}263 \times 5 \times 2 \times 90 \times 72$	20,263
D6	Breast Cancer Wisconsin (Mangasarian and Wolberg 1990)	$645 \times 10 \times 11 \times 2 \times 10$	630
D7		$10 \times 645 \times 9 \times 10 \times 10$	630
D8		$10 \times 10 \times 11 \times 10 \times 645$	630
D9		$2 \times 10 \times 10 \times 10 \times 645$	630
D10		$10 \times 10 \times 645 \times 9 \times 10$	630
D11	IPUMS Census Database (Ruggles and Sobek 1997)	$3{,}890 \times 4 \times 13 \times 3 \times 3$	4,863
D12		$545 \times 3 \times 17 \times 3 \times 2$	698
D13		$11 \times 3 \times 4 \times 5 \times 3$	27
D14	Mushroom	$10 \times 3 \times 5 \times 2 \times 7$	24
D15	Dermatology	$62 \times 5 \times 5 \times 5 \times 3$	58

8.1 Setup: data sets

For evaluating DBN under different scenarios, we used various data sets from the UCI Machine Learning Repository (Frank and Asuncion 2010). In particular, we considered the two cases introduced in Sect. 5:

Case 1 We first evaluate DBN in situations where the join attribute X determines all attributes of the relation $\mathcal{R}$. For these experiments, we considered 15 different data sets (D1-D15) with different sizes and different attribute sets (Table 7). All tensors were encoded as occurrence tensors. In the cases where a suitable join attribute did not exist in the data, we selected an attribute with FD support $\geq \tau_{support} = 75\,\%$ against all other attributes. We then removed all non-supporting tuples to make sure that the data set $\mathcal{R}$ satisfies the properties of Case 1. Note that, each partitioned data set contains as many tuples (nonzero entries) as the input relation $\mathcal{R}$.

Case 2 Secondly, we evaluate DBN in situations where the join attribute X determines only a subset of the attributes of the relation $\mathcal{R}$. In this case, we considered three different data sets (D16-D18). All tensors were encoded as occurrence tensors. The tensor size and numbers of nonzero entries of each relation are shown in Table 8. Note that the partition $\mathcal{R}_1$ containing X and the attributes determined by X has potentially smaller number of nonzero entries than $\mathcal{R}$; the number of nonzero entries of the other partition $\mathcal{R}_2$ is same as that of $\mathcal{R}$. As we discussed in Sect. 5, for dense tensors and Tucker decompositions, we targeted partitions where sizes of $\mathcal{R}_1$ and $\mathcal{R}_2$ are similar.

Table 8 Relational tensor data sets with different numbers of nonzero entries for partitions for DBN

Data set		Mode	Size	# nonzero of $\mathcal{R}_1$	# nonzero of $\mathcal{R}_2$
D16	Adult (subset)[a]	5	$118 \times 90 \times 1{,}000 \times 5 \times 2$	1,000	1,102
D17	Adult	4	$118 \times 90 \times 20{,}263 \times 94$	20,263	25,331
		5	$118 \times 90 \times 20{,}263 \times 94 \times 72$	20,263	27,351
		6	$118 \times 90 \times 20{,}263 \times 94 \times 72 \times 42$	20,263	27,424
D18	IPUMS	4	$2{,}241 \times 1{,}096 \times 191 \times 209$	1,096	2,359
	Census	5	$3{,}888 \times 2{,}241 \times 1{,}096 \times 191 \times 209$	1,096	5,881
	Database	6	$3{,}890 \times 2{,}241 \times 51 \times 1{,}096 \times 192 \times 209$	1,096	6,436

[a] For D16, we used a subset of randomly selected 1,000 entries from this data set for experiments with dense tensor model: the whole data set is too large for conventional decomposition operators under the dense tensor model

Table 9 Algorithms

Algorithm		Description
CP	DBN-NWAY	DBN-CP using N-way PARAFAC
	DBN-CP-ALS	DBN-CP using single grid NTF (CP-ALS)[a]
	NNCP-NWAY	NNCP using N-way PARAFAC
	NNCP-CP-ALS	NNCP using single grid NTF (CP-ALS)[a]
	NNCP-NWAY-GRID*	NNCP using grid NTF with "*" grid cells (N-way PARAFAC)[a]
	NNCP-CP-GRID*	NNCP-CP-ALS with "*" grid cells
	DBN*-CP-ALS	intraFD-based DBN-CP-ALS with "*" pairs
	DBN*-NWAY	intraFD-based DBN-NWAY with "*" pairs
	pp-DBN*-CP-ALS	pairwise parallel DBN*-CP-ALS
	pp-DBN*-NWAY	pairwise parallel DBN*-NWAY
Tucker	MET*	"*" modes element-wise Memory-Efficient Tucker
	DBN-MET*	DBN-Tucker using MET*
	pp-DBN-MET*	pairwise parallel DBN-MET*

[a] The algorithms in parentheses are the base PARAFAC for grid NTF

8.2 Setup: target ranks

Both for CP and Tucker decomposition experiments, we considered three target ranks: 6, 12, and 24. These lead to 4 rank pairs ($\langle 1, 6\rangle$, $\langle 2, 3\rangle$, $\langle 3, 2\rangle$, $\langle 6, 1\rangle$) to be considered for the target rank 6, 6 pairs ($\langle 1, 12\rangle$, $\langle 2, 6\rangle$, $\langle 3, 4\rangle$, $\langle 4, 3\rangle$, $\langle 6, 2\rangle$, $\langle 12, 1\rangle$) for the target rank 12, and 8 pairs ($\langle 1, 24\rangle$, $\langle 2, 12\rangle$, $\langle 3, 8\rangle$, $\langle 4, 6\rangle$, $\langle 6, 4\rangle$, $\langle 8, 3\rangle$, $\langle 12, 2\rangle$, $\langle 24, 1\rangle$) for the target rank 24.

8.3 Setup: alternative tensor decomposition algorithms

We experimented with various alternative algorithms for CP and Tucker decompositions. Table 9 lists the various algorithms we use in our experiments. We used MAT-

LAB Version 7.11.0.584 (R2010b) 64-bit (glnxa64) and MATLAB Parallel Computing Toolbox.

8.3.1 CP decomposition (single core)

The first decomposition algorithm we considered is the N-way PARAFAC algorithm with nonnegativity constraint (we call this N-way PARAFAC in the rest of the paper) which is available in the N-way Toolbox for MATLAB (Andersson and Bro 2000). We refer to DBN-CP and conventional non-negative CP (NNCP) implemented using this N-way PARAFAC implementation as DBN-NWAY and NNCP-NWAY, respectively.

Since MATLAB's N-way PARAFAC implementation uses a dense tensor (multi-dimensional array) representation, it is too costly to be practical for sparse tensors. Therefore, we implemented a variant of the single grid NTF (Phan and Cichocki 2011) using CP-ALS as the base PARAFAC algorithm. We refer to DBN-CP and NNCP based on CP-ALS as DBN-CP-ALS and NNCP-CP-ALS respectively.

8.3.2 CP decomposition (parallel, multi-core)

For the parallel version of the NNCP, we implemented the grid NTF algorithm (Phan and Cichocki 2011) with different number of grid cells (2, 4, 6, and 8 grid cells along the join mode) using N-way PARAFAC and CP-ALS as the base PARAFAC algorithms. Each grid is run with the base PARAFAC algorithm separately in parallel. We refer to the grid NTF algorithm for parallel NNCP implemented using N-way PARAFAC as NNCP-NWAY-GRID* (* denotes the number of partitions). Similarly, we refer to CP-ALS based implementations of parallel NNCP as NNCP-CP-GRID*.

The parallel version of DBN-CP are implemented using pairwise parallel DBN-NWAY and DBN-CP-ALS strategies where each pair is assigned to a separate processing unit; these are referred to as pp-DBN-NWAY and pp-DBN-CP-ALS respectively.

8.3.3 Tucker decomposition (single core)

Conventional Tucker decomposition algorithms, such as Andersson and Bro (2000), are ineffective on large dense data sets. Therefore, we focus on Tucker decompositions of sparse data sets. We consider MET (Memory-Efficient Tucker) in Kolda and Sun (2008). For MET, we considered different variants, denoted as MET* according to the number of modes handled element-wise; MET* is also used as the base Tucker algorithm for DBN-Tucker; this is referred to as DBN-MET*.

8.3.4 Tucker decomposition (parallel, multi-core)

Since MET does not support parallelization, we only consider parallelization of DBN-MET*, which is referred to as pp-DBN-MET*.

Table 10 Different attribute sets, join attributes (X), supports of X (the lowest of all the supports of $X \rightarrow *$), and execution times for FDs discovery for D1-D18 where A_n is the nth attribute of each data set

Data set	Attributes		Join attr. (X)	Support of X (%)	Exec. time for FDs
D1	$\{A_{11}, A_{12}, A_3, A_9, A_{10}\}$		A_3	97	0.024s
D2	$\{A_2, A_3, A_9, A_8, A_4\}$		A_3	80	0.022s
D3	$\{A_1, A_3, A_{12}, A_{15}, A_{10}\}$		A_3	80	0.025s
D4	$\{A_3, A_7, A_{15}, A_8, A_{13}\}$		A_3	75	0.023s
D5	$\{A_3, A_9, A_{15}, A_{12}, A_1\}$		A_3	80	0.023s
D6	$\{A_1, A_4, A_7, A_{11}, A_6\}$		A_1	96	0.004s
D7	$\{A_4, A_1, A_{10}, A_8, A_9\}$		A_1	96	0.003s
D8	$\{A_6, A_5, A_7, A_8, A_1\}$		A_1	96	0.002s
D9	$\{A_{11}, A_9, A_6, A_3, A_1\}$		A_1	98	0.003s
D10	$\{A_5, A_4, A_1, A_{10}, A_8\}$		A_1	96	0.003s
D11	$\{A_8, A_{17}, A_{19}, A_3, A_2\}$		A_8	99	0.007s
D12	$\{A_{53}, A_2, A_{21}, A_3, A_4\}$		A_{53}	98	0.006s
D13	$\{A_{13}, A_{48}, A_{17}, A_{14}, A_2\}$		A_{13}	98	0.005s
D14	$\{A_4, A_9, A_{18}, A_{17}, A_2\}$		A_2	88	0.004s
D15	$\{A_{34}, A_{24}, A_{33}, A_{25}, A_{11}\}$		A_{34}	80	0.002s
D16	$\{A_{11}, A_{12}, A_3, A_9, A_{10}\}$		A_3	98	0.024s
D17	4-mode	$\{A_{11}, A_{12}, A_3, A_{13}\}$	A_3	96	0.024s
	5-mode	$\{A_{11}, A_{12}, A_3, A_{13}, A_1\}$			
	6-mode	$\{A_{11}, A_{12}, A_3, A_{13}, A_1, A_{14}\}$			
D18	4-mode	$\{A_{49}, A_{50}, A_{51}, A_{54}\}$	A_{50}	95	0.007s
	5-mode	$\{A_8, A_{49}, A_{50}, A_{51}, A_{54}\}$			
	6-mode	$\{A_8, A_{49}, A_{50}, A_{51}, A_{54}, A_{58}\}$			

8.4 Setup: rank pruning

For the experiments where we assess the impact of the intraFD-based rank pruning strategy described in Sect. 6, we considered 2, 3 and 4 pairs as limits; these are referred to as DBN2, DBN3, and DBN4, respectively (e.g., DBN-CP-ALS with 2 pairs selected is referred to as DBN2-CP-ALS).

8.5 Setup: functional dependency discovery

As discussed in Sect. 3.2, we extended TANE (Huhtala et al. 1999) to find approximate FDs. The supports of the approximate FDs for each attribute set of different relational data sets are shown in Table 10. The table also shows the execution times needed to discover the FDs for each data set. As the table shows, the modified TANE algorithm is very efficient for the considered numbers of attributes, i.e., 4 to 6 attributes in these experiments.[1] Since the execution times for finding approximate FDs are negligible

[1] Note that the cost increases linearly in the size of the input relation (Huhtala et al. 1999).

compared to the tensor decomposition time, in the rest of the paper, we focus only on the decomposition times.

8.6 Setup: evaluation criteria

Each experiment is run at least 5 times and we report the average accuracy and execution time of these runs.

8.6.1 Accuracy

We use the following *fit* function to measure tensor decomposition accuracy:

$$\mathrm{fit}(\mathcal{X}, \hat{\mathcal{X}}) = 1 - \frac{\|\mathcal{X} - \hat{\mathcal{X}}\|}{\|\mathcal{X}\|}. \tag{9}$$

Here, $\|\mathcal{X}\|$ is the Frobenius norm of a tensor $\mathcal{X}$. The fit is a normalized measure of how accurate a tensor decomposition of $\mathcal{X}$, $\hat{\mathcal{X}}$ w.r.t. a tensor $\mathcal{X}$.

8.6.2 Execution time

The execution times are measured by MATLABs `tic` and `toc` commands to start and stop the clock at the beginning and the end of the decomposition process, respectively.

8.6.3 Memory

Especially for dense tensors and Tucker decomposition, memory usage can be a major bottleneck. For Tucker decompositions, we report the maximum intermediate memory use provided by the MET* algorithm. For single core DBN, we report the maximum of the memory used by each rank-pair (evaluated one after the other). For parallel DBN, we report the sum of the memory used by each rank-pair (evaluated in parallel).

8.7 Execution time results for CP decompositions

We first present experimental results assessing the efficiency of the proposed DBN scheme relative to the conventional implementation of the CP based tensor decomposition in both stand-alone and parallelized versions. Note that, as discussed in Sect. 7, for sparse tensors, CP decomposition cost depends largely on the number of nonzero entries and this necessitates a way to leverage the join selectivity (of the partitioning attribute) to predict whether DBN will outperform conventional CP decomposition schemes. In Table 11, we report the join selectivity ϕ and selectivity cut-off, $\phi_{\perp}$, values (with and without rank pruning) for each of the data sets we considered in our experiments: we predict that DBN will most easily outperform the conventional CP decomposition schemes (even for sparse data) in the cases

Table 11 Join selectivity (ϕ) and thresholds with and without rank pruning ($\phi'_{\perp}$ and $\phi_{\perp}$)

Data set	Mode	Rank	$\phi_{\perp}$	$\phi'_{\perp}$ with rank pruning	ϕ
D1–D5	5-mode	Rank-6	0.00012	0.00006	0.00005
		Rank-12	0.00014	0.00007	
D6–D10	5-mode	Rank-6	0.00381	0.00190	0.00159
		Rank-12	0.00444	0.00222	
D11	5-mode	Rank-6	0.00049	0.00025	0.00021
		Rank-12	0.00058	0.00029	
D12	5-mode	Rank-6	0.00344	0.00172	0.00143
		Rank-12	0.00401	0.00201	
D13	5-mode	Rank-6	0.08889	0.04444	0.03704
		Rank-12	0.10370	0.05185	
D14	5-mode	Rank-6	0.10000	0.05000	0.04167
		Rank-12	0.11667	0.05833	
D15	5-mode	Rank-6	0.04138	0.02069	0.01724
		Rank-12	0.04828	0.02414	
D16	5-mode	Rank-6	0.00229	0.00117	0.001
		Rank-12	0.00267	0.00137	
D17	4-mode	Rank-6	0.00011	0.00007	0.00005
		Rank-12	0.00013	0.00008	
	5-mode	Rank-6	0.00010	0.00006	0.00005
		Rank-12	0.00012	0.00007	
	6-mode	Rank-6	0.00010	**0.00004**	**0.00005**
		Rank-12	0.00012	0.00007	
D18	4-mode	Rank-6	0.00179	0.00113	0.00091
		Rank-12	0.00209	0.00136	
	5-mode	Rank-6	0.00130	**0.00087**	**0.00091**
		Rank-12	0.00152	0.00105	
	6-mode	Rank-6	0.00137	**0.00042**	**0.00091**
		Rank-12	0.00191	0.00122	

The bold fonts are the cases where the join selectivity is greater than the threshold

where $\phi > \phi_{\perp}$. As we see in this table, however, in many cases, ϕ is lower than $\phi_{\perp}$. Therefore, we expect these situations to be challenging for DBN against conventional schemes. We evaluate this prediction in the following subsections and also show that DBN provides advantages even in these cases when parallel execution plans are considered.

As described in Sect. 5, for CP decompositions, we need to consider two distinct situations. In the first of these (D1-D15), the join attribute X determines all attributes of the relation $\mathcal{R}$; i.e., $\text{nnz}(\mathcal{R}_1) = \text{nnz}(\mathcal{R}_2) = \text{nnz}(\mathcal{R})$, where $\text{nnz}(\mathcal{X})$ denotes the number of nonzero entries of $\mathcal{X}$. In the second case, the join attribute X determines only a subset of the attributes of the relation $\mathcal{R}$; i.e., $\text{nnz}(\mathcal{R}_1) \leq \text{nnz}(\mathcal{R})$ and $\text{nnz}(\mathcal{R}_2) = \text{nnz}(\mathcal{R})$.

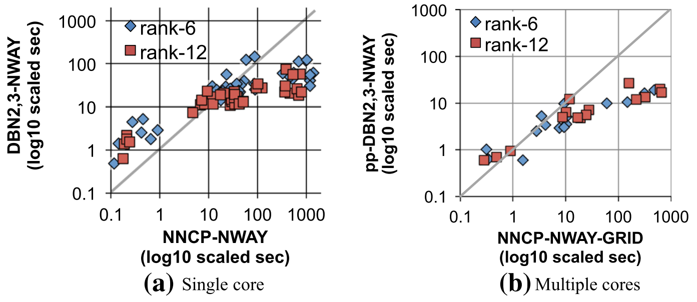


Fig. 9 Average running times of DBN2,3-NWAY (DBN2 and DBN3 for rank-6 and rank-12, respectively) vs. NNCP-NWAY on **a** a single core and **b** 4 and 6 cores for rank-6 and rank-12, respectively (NNCP-NWAY-GRID is avg of GRID2 and GRID4 and avg of GRID2 and GRID6 for rank-6 and rank-12, respectively). On both a single core and multi cores, majority of data points are located under the diagonal (25 and 26 out of 41 for rank-6 and 12, respectively for a single core and 11 out of 15 for both rank-6 and 12 for multi cores), which indicates that DBN-NWAY outperforms NNCP-NWAY, especially when running times are bigger. Note that rank-24 results have been excluded from these charts because the conventional NWAY based NNCP is not feasible for this target rank with the hardware setup used for the experiments

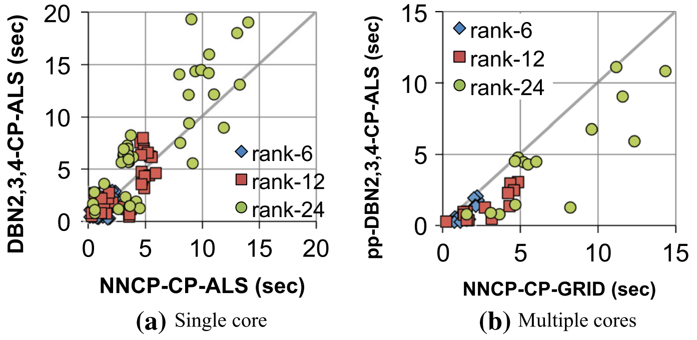
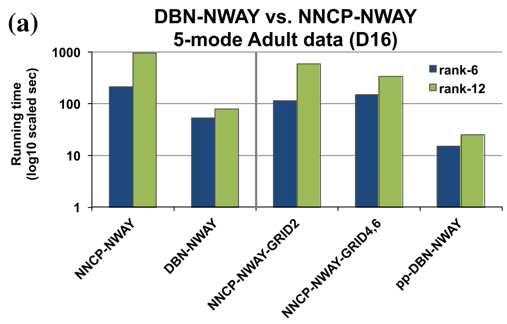


Fig. 10 Average running times of DBN2,3,4-CP-ALS (DBN2, DBN3, and DBN4 for rank-6, rank-12, and rank-24, respectively) vs. NNCP-CP-ALS on **a** a single core and **b** 4, 6, and 8 cores for rank-6, rank-12, and rank-24, respectively (NNCP-CP-GRID is avg of GRID2 and GRID4, avg of GRID2 and GRID6, and avg of GRID2 and GRID8 for rank-6, rank-12, and rank-24, respectively). On a single core, more than half points are upper the diagonal (24, 15, and 10 out of 41 for rank-6, 12, and 24, respectively); i.e., DBN-CP is beaten by NNCP. However, when DBN-CP and NNCP are parallelized, DBN-CP outperforms NNCP in most cases (15, 14, and 15 out of 15 for rank-6, 12, and 24, respectively)

8.7.1 Case 1: X determines all attributes of $\mathcal{R}$

Dense Tensors In Fig. 9, we first compare the execution times for (NWAY based) DBN with NNCP for dense tensors. The figure includes results both for single-core and multi-core setups. As we see in these results, in both setups, DBN outperforms NNCP when the problems get more difficult to solve and tensor decomposition algorithms require more time. As the problem difficulty increases DBN provides $\sim$ 1 order (for single core) to $\sim$ 2 orders (for multi core) time gains over NNCP.

Sparse Tensors In Fig. 10, we compare the execution times for (CP-ALS based) DBN with NNCP for sparse tensors. The figure includes results both for single-core and multi-core setups. Remember that in this case, we predict that when ϕ of the input



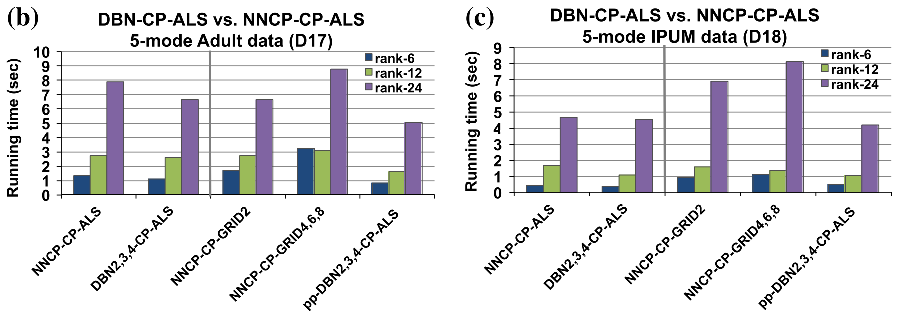


Fig. 11 Running times of **a** DBN-CP vs. NWAY based algorithms on 5-mode `Adult`(subset) data set (D16) and DBN-CP vs. CP-ALS based algorithms on **b** 5-mode `Adult` data set (D17) and **c** 5-mode `IPUMS` data set (D18) with different target ranks in both single core and multi-core

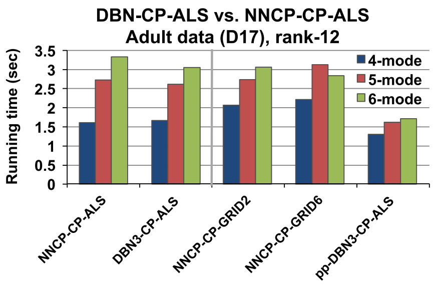
relations are lower than $\phi_\perp$, we expect DBN to have difficulties. This is confirmed in Fig. 10a, where we see that in single core scenarios, DBN-CP based schemes are not as competitive as NNCP as predicted based on the ϕ and $\phi_\perp$ values in Table 11.

It is important to note, however, that DBN still provides significant advantages even when $\phi < \phi_\perp$ when parallel execution opportunities are leveraged. As we see in Fig. 10b, on the same data, when using multiple cores, DBN based scheme outperforms NNCP in most cases.

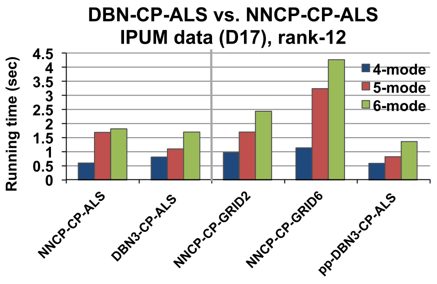
8.7.2 Case 2: X does not determine all attributes of $\mathcal{R}$

Figure 11a shows the results for the corresponding subset of the `Adult` data set (D16) for which the conventional NWAY based decomposition schemes is feasible. As expected, DBN-CP based schemes outperform conventional CP decomposition schemes for different target ranks (rank-6 and rank-12) in both single-core and multi-core settings.

In Fig. 11b, c, we compare DBN-CP against the CP-ALS based algorithms for different target ranks (rank-6, rank-12, and rank-24) on `Adult` (D17) and `IPUMS` (D18) data sets, respectively and in Fig. 12 a, b, we compare DBN-CP against the CP-ALS based algorithms for different number of modes (4-mode, 5-mode, and 6-mode) for rank-12 decomposition on `Adult` (D17) and `IPUMS` (D18) data sets, respectively.



(a) `Adult` data set (D17)



(b) `IPUMS` data set (D18)

Fig. 12 Running times of DBN-CP vs. CP-ALS based algorithms for **a** `Adult` data set (D17) and **b** `IPUMS` data set (D18) for 4-mode, 5-mode, and 6-mode input tensors for rank-12 decomposition in both single-core and multi-core

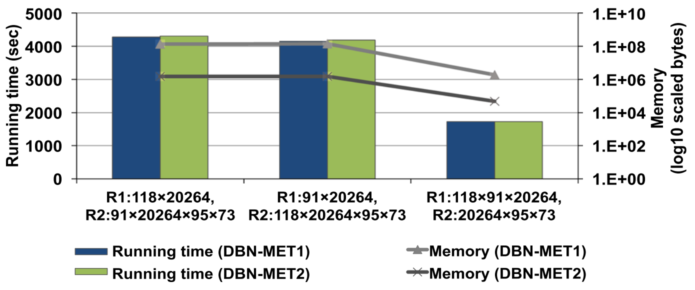
As we see here, in almost all cases (especially when the data modality is high), DBN-CP based schemes outperform CP-ALS based schemes and pp-DBN-CP-ALS is the fastest in all cases. Note that these high modality cases are also the cases where the join selectivity ϕ of the relations are higher than the lower bound $\phi'_{\perp}$ with rank pruning (see Table 11) and the results confirm that DBN-CP is more advantageous in these cases as discussed in Sect. 7.1.2. It is interesting to note that while GRID-based parallel version of CP-ALS may in practice negatively impact performance (since the underlying ALS-based combining approach involves significant communication overheads), parallelized DBN-CP is effective in reducing execution times.

8.8 Execution time results for tucker decompositions

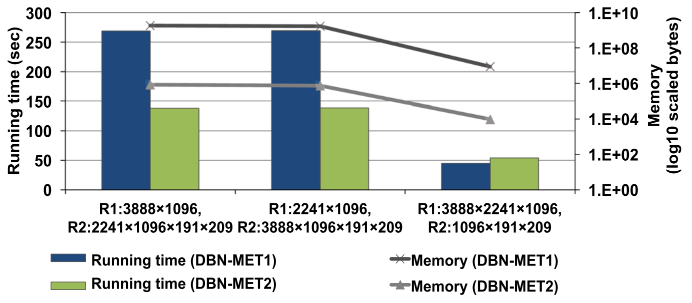
For the Tucker decomposition experiments, we focus on the data sets, D17 and D18, where the sizes of the modes are large. For comparison against the DBN strategy, we consider the MET (Memory-Efficient Tucker) Kolda and Sun (2008) implementation of Tucker. Since there are multiple mode-selection strategies for MET, unless otherwise specified, we present the results for the strategy that leads to best running time and memory consumption for MET. We also consider different implementations of MET, denoted as MET1 and MET2. The difference between MET1 and MET2 is that in the first one only one mode is handled element-wise, whereas in the second one two modes are handled element-wise.

8.8.1 Impact of partition balance

Before we compare the DBN strategy against conventional Tucker decompositions, we investigate the impact of partition balance on the performance of DBN. As we discussed in Sect. 5, for Tucker decompositions, we expect that partition strategies that lead to balanced sub-relations will lead to better DBN performance. In Fig. 13, we present execution time and memory consumption results for three different partitioning strategies for each of the D17 and D18 data sets in Table 12. We note that($\mathcal{R}_1$: $\mathbf{118 \times 91 \times 20{,}264}$, $\mathcal{R}_2$: $\mathbf{20{,}264 \times 95 \times 73}$) and ($\mathcal{R}_1$: $\mathbf{3{,}888 \times 2{,}241 \times 1{,}096}$,



(a) `Adult` data set (D17)



(b) `IPUMS` data set (D18)

Fig. 13 Running time and bottleneck memory for different partitioning cases where the two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ have different sizes (see Table 12), which are run by DBN-MET1 and DBN-MET2 for 5-mode **a** `Adult` data set (D17) **b** `IPUMS` data set (D18) for rank-(12,12,12,12,12) decomposition

Table 12 Different partitioning cases for `Adult` (D17) and `IPUMS` (D18) data sets

Data set	Partition size	
	$\mathcal{R}_1$	$\mathcal{R}_2$
D17	118 × 20,264	91 × 20,264 × 95 × 73
	91 × 20,264	118 × 20,264 × 95 × 73
	118 × 91 × 20,264	**20,264 × 95 × 73**
D18	3,888 × 1,096	2241 × 1,096×191 × 209
	2,241 × 1,096	3,888 × 1,096×191 × 209
	3,888 × 2,241 × 1,096	**1,096 × 191 × 209**

The partitions in bold are the most balanced among all three

$\mathcal{R}_2$: **1,096** × **191** × **209**) partitioning alternatives are the most balanced among all three for the D17 and D18 data sets, respectively.

The results confirm that, as expected, the most balanced partitioning case (in terms of both size and number of modes) shows the best performance in terms of both running time and memory consumption.

8.8.2 DBN-MET vs. conventional MET: impact of the number of modes

We next compare the DBN strategy against conventional MET with respect to the impact of the number of modes of the tensor on the decomposition performance. Since

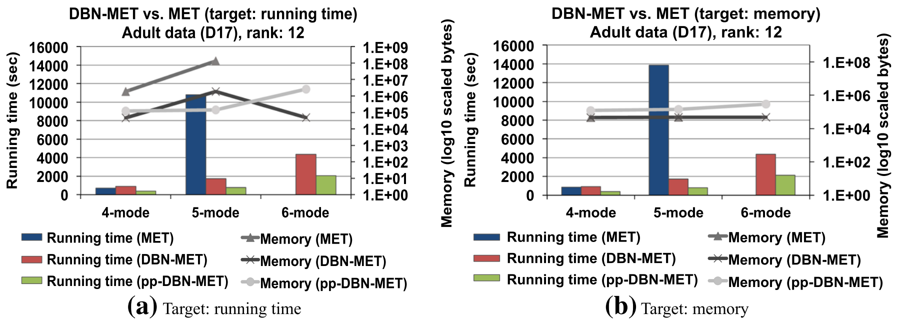


Fig. 14 The running time and bottleneck memory consumption for `Adult` data set (D17) of DBN-MET vs. MET: in **a** the optimization target is the running time, whereas in **b** the optimization target is the memory. Here, we use rank 12 for each mode of the relation. When the tensor has 6 modes none of the conventional MET algorithms fit the available memory and thus they are not included in the plots

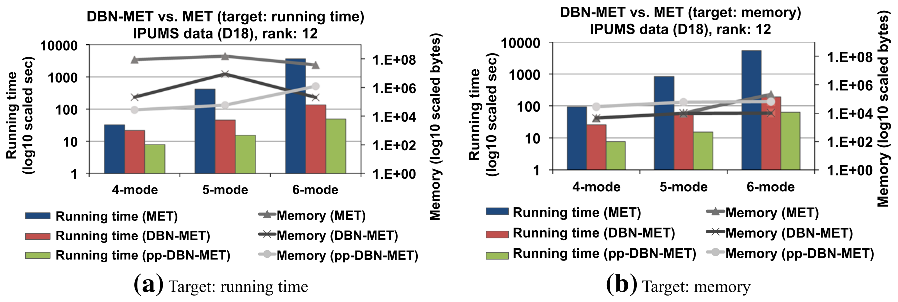


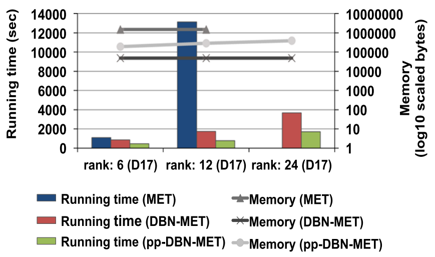
Fig. 15 The running time and bottleneck memory consumption for `IPUM` data set (D18) of DBN-MET vs. MET: in **a** the optimization target is the running time, whereas in **b** the optimization target is the memory. Here, we use rank 12 for each mode of the relation

there are multiple MET strategies with different run-times and memory consumptions, we present two sets of results, the first targeting better MET run-time and the second better MET memory consumption.

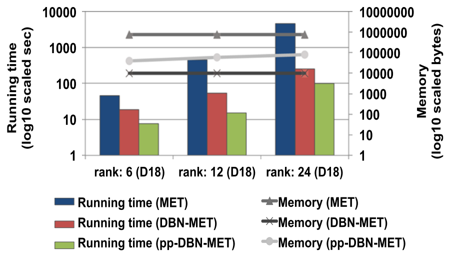
Figure 14 presents results for the `Adult` data set (D17). As we see here, as the number of modes increases, the running times of all decomposition algorithms increase. Experiment results confirm that the increase in the execution time is much slower for the DBN based decompositions and, as expected, the parallelized version of DBN (pp-DBN) is the fastest among all alternatives. The results with the `IPUMS` data set (D18), reported in Fig. 15 re-confirm these results. Note that, the time results for this `IPUM` data set are presented in *log-scale* due to the significant differences in execution times between DBN-based and conventional decomposition strategies.

8.8.3 DBN-MET vs. conventional MET: impact of the rank

In Fig. 16, we compare the performance of DBN against conventional MET for different target ranks. Again, since there are different MET implementations, we present results for the strategy that provides the best running time for MET.



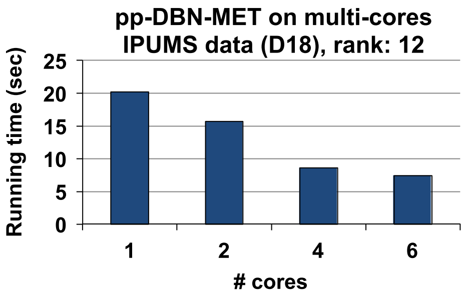
(a) 5-mode `Adult` data set (D17)



(b) 5-mode `IPUMS` data set (D18)

Fig. 16 The running time and bottleneck memory consumption of DBN-MET vs. MET for rank-(6,6,6,6,6), rank-(12,12,12,12,12), vs. rank-(24,24,24,24,24) (here we simply denote rank: 6, rank: 12, and rank: 24 respectively) for 5-mode `Adult` data set (D17) and `IPUM` data set (D18). For the D17 data set, for target rank 24, none of the conventional MET algorithms fit the available memory and thus they are not included in the plot

Fig. 17 The running time of pp-DBN-MET on multiple cores for rank-(12,12,12,12,12) on `IPUM` data set (D18)



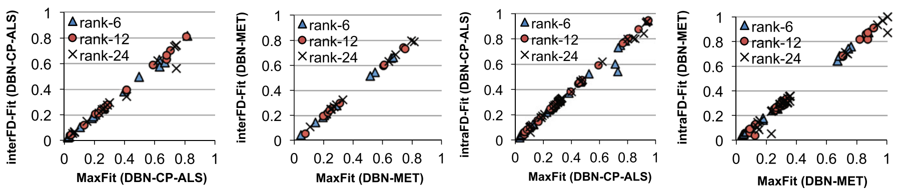
As we see in the figure, as the target rank increases, the running time of the conventional MET algorithm increases very quickly. In contrast, the running times of DBN-based strategies increase much more slowly. Again, the parallelized version of DBN (pp-DBN) is the fastest among all alternatives. Note that, as expected, the memory consumption of pp-DBN is higher than DBN-MET; however, it is still at least an order lesser than MET.

8.9 DBN: scalability

In this section, we study how pp-DBN scales as the number of cores increases. We use pp-DBN-MET (pp-DBN-CP works similarly), which parallels each rank pair, e.g, rank-12 can parallel on up to 6 cores. As we see in Fig. 17, the running time of pp-DBN-MET decreases as the number of cores increases and the time gets fastest when 6 cores are used since the entire rank pairs are parallel. The time drops down most from 2 cores to 4 cores.

8.10 Accuracy results

So far, we have shown that DBN-based strategies are significantly more efficient than their conventional counterparts. In this subsection, we experimentally assess the accuracy of DBN-based strategies.



(a) interFD-Fit vs. MaxFit **(b)** intraFD-Fit vs. MaxFit

Fig. 18 **a** InterFD-based fit vs. maximum fit and **b** intraFD-based fit vs. maximum fit of DBN-CP-ALS and DBN-MET for D1-D15 (we omit D13 and D15 for Tucker decomposition as the sizes of the join mode in the data sets are smaller than rank 24)

8.10.1 Impacts of the intraFD-based rank pruning and interFD-based partitioning on accuracy

Before we compare DBN-based strategies against conventional decompositions, we first study the impacts of the interFD-based partitioning (Sect. 5) and intraFD-based rank pruning (Sect. 6) strategies on accuracy. These results are presented in Fig. 18a and b, respectively, where we compare the fit values obtained when using the proposed strategies against the maximum potential fit values one can obtain using a DBN-based strategy. In these plots, the closer to the 45 degree line the results are, the more effective are the FD based rank pruning and data partitioning strategies.

In Fig. 18a, we study the impact of the interFD-based partition selection approach on the decomposition accuracy. The results show that, for both CP and Tucker decompositions, the interFD-based partitioning strategy results in accuracies that are very close to the maximum possible accuracy, using an optimal partitioning strategy for all rank configuration.

In Fig. 18b, we investigate the impact of intraFD-based rank pruning (with only the best 50 % of the rank pairs considered by the JBD module among the potential rank pairs). Since the intraFD strategy ignores pairwise FDs involving the join attribute, we consider only the situations where each sub-tensor has more than 2 attributes (the join attribute and a determined attribute). As we see in Fig. 18b, the intraFD-based rank pruning strategy is very effective: except in a very few cases, the intraFD-based rank pruning does not eliminate the rank pair that will lead to the maximum possible fit with a DBN strategy.

8.10.2 DBN vs. conventional decompositions

We next evaluate the accuracy of DBN based decompositions against conventional decomposition algorithms, NNCP-CP-ALS for CP decomposition (since results for DBN-NWAY and NNCP-NWAY are similar we only present NNCP-CP-ALS) and MET for Tucker decomposition. We report accuracy results for data sets D1-D15 since reconstructing the decomposed tensor needed for computing the fit value on larger data sets is not feasible with the available resources. Figures 19 and 20 present the accuracy results of DBN vs. NNCP and MET for CP and Tucker decompositions for different data sets, respectively. We also present the correlations between accuracies of DBN and conventional decomposition strategies in Table 13.

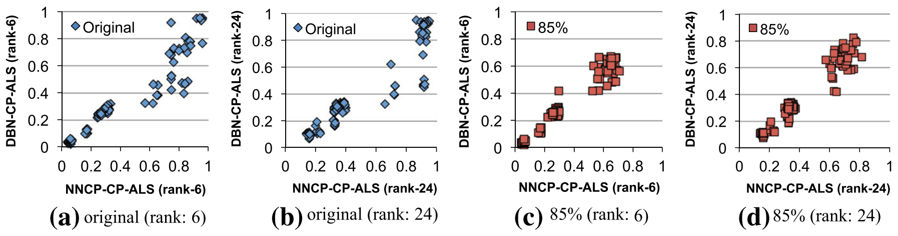


Fig. 19 Accuracies of DBN vs. NNCP for original and outlier-eliminated (85 % cumulative value preserved) CP decompositions for all partitioning cases of D1-D15

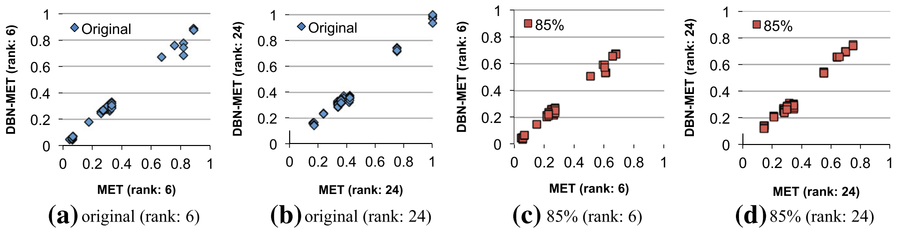


Fig. 20 Accuracies of DBN vs. MET in original and outlier-eliminated (85 % cumulative value preserved) Tucker decomposition for all partitioning cases of D1-D15 (we omit D13 and D15 as the sizes of the join mode in the data sets are smaller than rank 24)

Table 13 Correlation between fits of DBN vs. NNCP and MET in original and outlier-eliminated (85 % cumulative value preserved) CP and Tucker decomposition for all partitioning cases of D1-D15

		Correlation	
		Original	85 %
DBN-CP-ALS vs. NNCP-CP-ALS	Rank-6	0.94	0.97
	Rank-24	0.93	0.96
DBN-MET vs. MET	Join mode rank: 6	0.99	0.99
	Join mode rank: 24	0.99	0.99

Figure 19a, b and Table 13 shows that the accuracy of DBN is highly correlated with the accuracy of NNCP. There are, however, cases in which DBN has lower accuracies than NNCP. In order to understand whether this is a fundamental limitation of the DBN strategy or whether it is due to simple outliers in the data, we next consider whether the problem also occurs in the cases where the decomposition results are sparcified by ignoring the outliers: for this purpose we leverage a commonly used decomposition sparcification strategy (Allen 2012): treating each core element as a cluster and each factor entry as a cluster membership probability, we eliminate those elements that have very small likelihood of being a member of a given cluster. In particular, we remove sufficient outlier elements to eliminate the lowest 15 % of the membership probabilities. Figure 19c, d and Table 13 show that once the outliers are removed from consideration, DBN-based strategies perform as good as the conventional CP decomposition strategies.

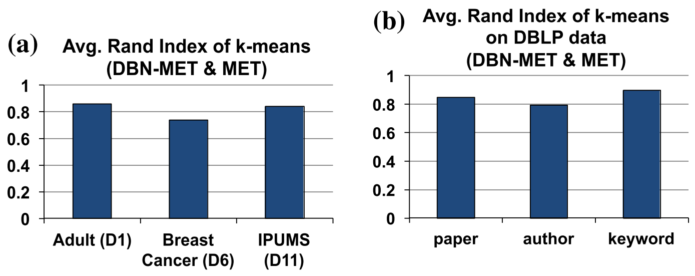


Fig. 21 Avg. Rand Index of k-means clustering (k= 10, 30, 50 and 100) on tensor decompositions of DBN-MET and MET on the join attribute of **a** Adult, Breast Cancer, and IPUMS data sets and **b** `paper`, `author`, and `keyword` modes of DBLP data set

In fact, once the outliers are ignored in the decomposition, in a significant portion of the cases, the DBN strategy results in higher accuracies than the conventional DBN (indicated by an increase in the number of results above the 45 degree line).

For Tucker decomposition, the correlations between the fits of DBN and MET are very high (almost 1.0) for both original and outlier-eliminated tensors (see Fig. 20 and Table 13). These results confirm that the proposed DBN scheme is especially effective for Tucker decompositions.

8.10.3 Clustering results

To contextualize the fit results, we have also run additional experiments where we have considered a “clustering” task as the ground truth. Here, we present clustering results performed on the tensor decompositions obtained by DBN. In particular, we applied the k-means clustering algorithm on each factor matrix by treating each row as a high-dimensional point. We then used Rand Index (Rand 1971) to measure how similar the clustering results of DBN to those of MET.

We first evaluated the clustering on the three data sets (Adult, Breast Cancer, and IPUMS data sets). We performed DBN-MET and MET with the target rank 10 for each mode and then ran k-means (k=10, 30, 50 and 100) on the factor matrices of the join mode of each data set. As shown in Fig. 21a, overall clustering results obtained by DBN agree with those of MET ($\sim$80 %).

We also considered a 4-mode DBLP data set (Tang et al. 2008) `(paper, author, venue, keyword)` of size $11{,}941 \times 300 \times 6 \times 300$ containing 11,941 papers including 300 most frequent keywords in their abstracts written by 300 most prolific authors from 6 database and data mining-related venues.

We measured the average Rand Index of the k-means clustering results (k = 10, 30, 50 and 100) on tensor decompositions obtained by DBN-MET vs. MET on each mode of the DBLP data set (except the `venue` mode whose size is too small). As shown in Fig. 21, the two clustering results are similar ($\sim$80 to $\sim$85 %). Table 14 shows several sample clustering results from DBN-MET vs. MET.

Table 14 Sample keyword clustering results on DBLP data set

DBN-MET	MET
'learn' 'feature' 'train'	'test' 'knowledge' 'practice' 'engine' 'train'
'object' 'structure' 'concept' 'define' 'semantic' 'rule' 'represent' 'orient' 'view'	'object' 'support' 'type' 'access' 'security' 'semantic' 'orient' 'formal'
'data' 'database' 'query'	'data' 'database'
'communication' 'parallel' 'memory' 'processor' 'schedule'	'communication' 'architecture' 'fault'

8.10.4 DBN on multi-attribute dependencies

In Sect. 7.3.2, we discussed how DBN handles data sets where there is no single key that sufficiently determines other attributes and formulated a version of the process in cases where we need to use multiple attributes as a join key and, thus, need to have a compound key represented as a single attribute.

We evaluated this approach using MovieLens 10M data set (Movielens 2013) whose attributes are `(user, movie, rating, time)` of size $5{,}000 \times 5{,}000 \times 10 \times 11{,}431$ (we captured most frequent 5,000 users and movies). It is easy to see that in this data set `user` and `movie` attributes together would determine `rating` and `time` attributes. For example, the supports of FD are 100 vs. 33 % for (`user, movie` $\rightarrow$ `rating`) vs. (`user` $\rightarrow$ `rating`), respectively. We use CP decomposition with the target rank 10 for this experiment (The tensor size is too big to run Tucker decomposition).

We first measured the fit provided by DBN-CP-ALS using the `user` attribute as the join key. Since the `user` attribute does not determine all the other attributes by itself, we expect that the partitioning the data on the `user` attribute will generate errors. In fact, we obtained a negative fit (−0.0052) in this case.

Next we compared the clustering results from DBN-CP-ALS with one join attribute `(user)` vs. two join attributes `(user, movie)` using average Rand Index on k-means (k = 10, 30, 50, and 100) performed on each factor matrix. As the ground truth clusters, we use clustering results obtained by NNCP-CP-ALS.

As we see in Fig. 22, as expected, the clustering results on `rating` and `time` attributes determined when using both `user` and `movie` attributes as join key are significantly better than the case where when we only use the `user` attribute as the join key. Of course, when we use `user` and `movie` together as a compound join key, the clustering result for the `user` factor is less precise than when we use `user` as the join key, but this is expected. In fact, the overall fit for the whole tensor, when using the compound key `(user, movie)` is 0.0025 as opposed to −0.0052 for `(user)` alone, indicating that the use of the compound key indeed helps improve the overall accuracy.[2]

[2] Note that the fit is low in this experiment due to the extremely tight target rank (10) used for the decomposition for a very high dimensional tensor. The fit obtained by the conventional tensor decomposition technique, NNCP-CP-ALS, on the same tensor with the same rank is also similarly low, 0.0028.

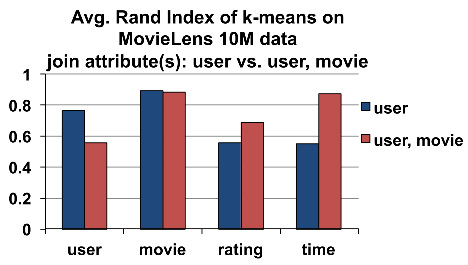


Fig. 22 Avg. Rand Index of k-means clustering (k = 10, 30, 50 and 100) on each factor matrix of MovieLens 10M data set by DBN-CP-ALS and NNCP-CP-ALS of `(user)` vs. `(user, movie)` attributes as the join key

9 Conclusions

The lifecycles of most data sets include a diverse set of operations, from capture, integration, projection, to data analysis. In general, analysis operations, including tensor decompositions, are among the costliest of these. To tackle the often prohibitively high cost of tensor decomposition tasks, in this paper we proposed a highly efficient, effective, and parallelizable *decomposition-by-normalization* (DBN) strategy for obtaining decompositions by a large and high-modal data set into the smaller tensors and combining the (CP or Tucker) decompositions of these smaller tensors to obtain the final decomposition. We also proposed interFD-based partitioning and intraFD-based rank pruning strategies for DBN to improve accuracy and to reduce execution time, respectively. Experiment results confirmed the efficiency and effectiveness of the proposed DBN scheme and its optimization strategies.

Acknowledgments This work is partially funded by NSF Grants #116394 “RanKloud: Data Partitioning and Resource Allocation Strategies for Scalable Multimedia and Social Media Analysis”.

References

Allen GI (2012) Sparse higher-order principal components analysis. In: Proceedings of the 15th international conference on artificial intelligence and statistics (AISTATS)
Andersson CA, Bro R (2000) The n-way toolbox for matlab. Chemom Intell Lab Syst 52(1):1–4. http://www.models.life.ku.dk/source/nwaytoolbox/
Antikainen J, Havel J, Josth JR, Herout A, Zemcik P, Hauta-Kasari M (2011) Nonnegative tensor factorization accelerated using gpgpu. IEEE Trans Parallel Distrib Syst 22(7):1135–1141
Bader BW, Kolda TG (2006) Efficient matlab computations with sparse and factored tensors. Technical Report SAND2006-7592, Sandia National Laboratories
Bader BW, Kolda TG (2007) Matlab tensor toolbox version 2.2. http://csmr.ca.sandia.gov/tgkolda/TensorToolbox/
Carroll J, Chang JJ (1970) Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. Psychometrika 35:283–319
Chu W, Ghahramani Z (2009) Probabilistic models for incomplete multi-dimensional arrays. In: Proceedings of the 12th international conference on artificial intelligence and statistics
Elmasri R, Navathe SB (1994) Fundamentals of database systems, 2nd edn. Benjamin-Cummings, Redwood City
Frank A, Asuncion A (2010) UCI machine learning repository. http://archive.ics.uci.edu/ml
Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, New York
Harshman RA (1970) Foundations of the PARAFAC procedure: models and conditions for an “explanatory” multi-modal factor analysis. UCLA Working Papers in Phonetics 16(1):84

Hoff PD (2011) Hierarchical multilinear models for multiway data. Comput Stat Data Anal 55(1):530–543. doi:10.1016/j.csda.2010.05.020

Huhtala Y, Kärkkäinen J, Porkka P, Toivonen H (1999) Tane: an efficient algorithm for discovering functional and approximate dependencies. Comput J 42(2):100–111

Ilyas IF, Markl V, Haas PJ, Brown P, Aboulnaga (2004) A Cords: automatic discovery of correlations and soft functional dependencies. In: SIGMOD conference, pp. 647–658

Karmarker N, Karp RM (1983) The differencing method of set partitioning. Technical report, Berkeley

Kim M, Candan KS (2011) Approximate tensor decomposition within a tensor-relational algebraic framework. In: Proceedings of the 20th ACM international conference on information and knowledge management, pp. 1737–1742 doi:10.1145/2063576.2063827

Kolda T, Sun J (2008) Scalable tensor decompositions for multi-aspect data mining. In: Proceedings of the 8th IEEE international conference on data mining, pp. 363–372. doi:10.1109/ICDM.2008.89

Kolda TG, Bader BW (2009) Tensor decompositions and applications. SIAM Rev 51(3):455–500. doi:10.1137/07070111X

Kolda TG, Bader BW, Kenny JP (2005) Higher-order web link analysis using multilinear algebra. In: Proceedings of the 5th IEEE international conference on data mining, pp. 242–249. doi:10.1109/ICDM.2005.77

Kruskal JB (1977) Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. Linear Algebr Appl 18(2):95–138

Lopes S, Petit JM, Lakhal L (2000) Efficient discovery of functional dependencies and armstrong relations. In: Proceedings of the 7th international conference on extending database technology: advances in database technology, EDBT ’00. Springer, London, pp. 350–364

Mahoney MW, Maggioni M, Drineas P (2006) Tensor-cur decompositions for tensor-based data. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 327–336. doi:10.1145/1150402.1150440

Mangasarian OL, Wolberg WH (1990) Cancer diagnosis via linear programming. SIAM News 23(5):1–18

Mannila H, Räihä KJ (1992) On the complexity of inferring functional dependencies. Discret Appl Math 40(2):237–243. doi:10.1016/0166-218X(92)90031-5

Movielens dataset from grouplens research group (2013). http://www.grouplens.org

Phan AH, Cichocki A (2011) Parafac algorithms for large-scale problems. Neurocomputing 74(11):1970–1984. doi:10.1016/j.neucom.2010.06.030

Rand WM (1971) Objective criteria for the evaluation of clustering methods. J Am Stat Assoc 66(336):846–850

Ruggles S, Sobek M (1997) Integrated public use microdata series: Version 2.0 minneapolis: historical census projects http://www.ipums.umn.edu/

Sanchez E, Kowalski BR (1986) Generalized rank annihilation factor analysis. Anal Chem 58(2):496–499. doi:10.1021/ac00293a054

Sanchez E, Kowalski BR (1990) Tensorial resolution: a direct trilinear decomposition. J Chemom 4(1):29–45. doi:10.1002/cem.1180040105

Stoer M, Wagner F (1997) A simple min-cut algorithm. J ACM 44(4):585–591. doi:10.1145/263867.263872

Sun J, Papadimitriou S, Lin CY, Cao N, Liu S, Qian W (2009) Multivis: content-based social network exploration through multi-way visual analysis. In: Proceedings SDM, vol 9. SIAM, pp. 1063–1074

Sun J, Tao D, Papadimitriou S, Yu PS, Faloutsos C (2008) Incremental tensor analysis: theory and applications. ACM Trans Knowl Discov Data 2(3):11:1–11:37. doi:10.1145/1409620.1409621

Tang J, Zhang J, Yao L, Li J, Zhang L, Su Z (2008) Arnetminer: extraction and mining of academic social networks. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp. 990–998

Tsourakakis CE (2010) Mach: fast randomized tensor decompositions. In: Proceedings of the 10th SIAM International Conference on Data Mining, pp. 689–700

Tucker L (1966) Some mathematical notes on three-mode factor analysis. Psychometrika 31(3):279–311. doi:10.1007/BF02289464

Wyss C, Giannella C, Robertson EL (2001) Fastfds: a heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances: extended abstract. In: Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery, DaWaK ’01. Springer, London, pp 101–110

Xu Z, Yan F, Qi A (2012) Infinite tucker decomposition: nonparametric bayesian models for multiway data analysis. In: ICML. icml.cc/Omnipress

Zhang Q, Berry M, Lamb B, Samuel T, Allen G, Nabrzyski J, Seidel E, van Albada G, Dongarra J, Sloot P (2009) A parallel nonnegative tensor factorization algorithm for mining global climate data, vol 5545. Springer, Berlin/Heidelberg, pp. 405–415

Zhou G, He Z, Zhang Y, Zhao Q, Cichocki A (2009) Canonical polyadic decomposition: from 3-way to n-way. In: Eighth international conference on computational intelligence and security (CIS), pp 391–395. doi:10.1109/CIS.2012.94