Decomposition-by-Normalization (DBN): Leveraging Approximate Functional Dependencies for Efficient Tensor Decomposition[*]

Mijung Kim
Arizona State University
Tempe, AZ 85287, USA
mijung.kim.1@asu.edu

K. Selçuk Candan
Arizona State University
Tempe, AZ 85287, USA
candan@asu.edu

ABSTRACT

For many multi-dimensional data applications, tensor operations as well as relational operations need to be supported throughout the data lifecycle. Although tensor decomposition is shown to be effective for multi-dimensional data analysis, the cost of tensor decomposition is often very high. 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 and then performs the decomposition using these smaller tensors. 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. Experiments confirm the efficiency and effectiveness of the proposed `decomposition-by-normalization` scheme compared to the conventional nonnegative CP decomposition approach.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing, Relational databases*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*

General Terms

Algorithms, Experimentation

Keywords

Tensor Decomposition, Tensor-based Relational Data Model

[*]This work is supported by the NSF Grant #1043583 - 'MiNC: NSDL Middleware for Network- and Context-aware Recommendations' and the NSF Grant #1116394 'RanKloud: Data Partitioning and Resource Allocation Strategies for Scalable Multimedia and Social Media Analysis'

1. INTRODUCTION

For many multi-dimensional data applications, tensor operations as well as relational operations need to be supported throughout the data lifecycle (from collection to analysis). Tensors are high-dimensional arrays. Due to the convenience they provide when representing relationships among different types of entities, tensor representation has been increasingly used for relational data in various fields from scientific data management to social network data analysis. Tensor based representations have proven to be useful for multi-aspect data analysis and tensor decomposition has been an important tool to capture high-order structures in multi-dimensional data [2, 11, 12, 17, 20, 23].

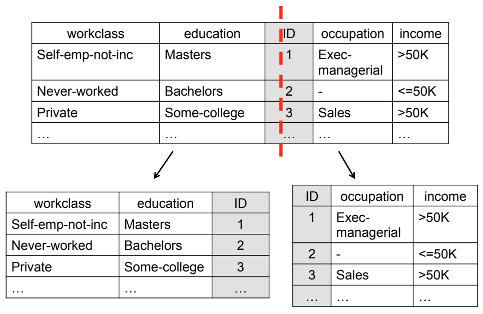
Although tensor decomposition is shown to be effective for multi-dimensional data analysis, the cost of tensor decomposition is often very high, especially in dense tensor representations where the cost increases exponentially with the number of modes of the tensor. While decomposition cost increases more slowly (linearly with the number of nonzero entries in the tensor) for sparse tensors, the operation can still be very expensive for large data sets. In fact, in many data intensive systems, data is commonly high-dimensional and large-scale and, consequently, of all the operations that need to be supported to manage the data, tensor decompositions tend to be the costliest ones. Recent attempts to parellelize tensor decomposition [2, 17, 23] face difficulties, including large synchronization and data exchange overheads.

1.1 Contribution of this Paper: Decomposition-by-Normalization (DBN)

Our goal is to tackle the high computational cost of tensor decomposition process. Since, especially for dense data sets, the number of modes of the tensor data is one of the main factors contributing to the cost of tensor operations, we focus on how we reduce the dimensionality and the size of the input tensor. In particular, we argue that if

- a high-dimensional data set (i.e., a tensor with large number of modes) can be normalized (i.e., vertically partitioned) into lower-dimensional data sets (i.e., tensors with smaller number of modes) and
- each vertical partition (sub-tensor) is decomposed independently,

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

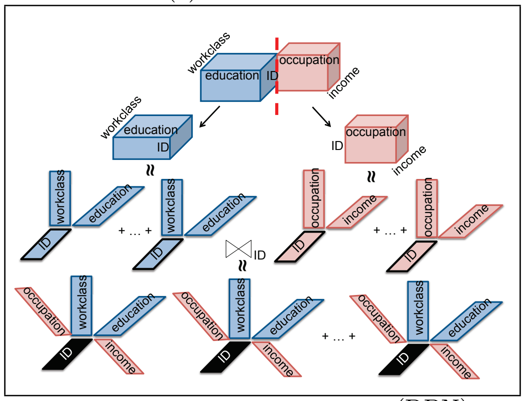


workclass	education	ID	occupation	income
Self-emp-not-inc	Masters	1	Exec-managerial	>50K
Never-worked	Bachelors	2	-	<=50K
Private	Some-college	3	Sales	>50K
…	…	…	…	…

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



(b) `decomposition-by-normalization` (DBN) process

Figure 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`); (b) `decomposition-by-normalization`: normalization of $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$ and 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 CP decomposition of $\mathcal{R}$.

Example 1.1. *Consider the 5-attribute relation, $\mathcal{R}$(`workclass, education, ID, occupation, income`), in Figure 1(a) and assume that we want to decompose the corresponding tensor for multi-dimensional analysis.*

Since high-dimensional tensor decomposition is expensive, we argue that a better processing scheme would involve first normalizing this relation into the smaller tensors (based on the functional dependencies of the relation) and then performing the decompositions of these smaller tensors. These normalized tensors are lower-dimensional than the original input tensor, therefore these decompositions are likely to be much less expensive than the decomposition of the original input tensor. Figure 1(a) illustrates an example normalization which divides the 5-attribute relations into two smaller relations with 3 attributes, $\mathcal{R}_1$(`workclass, education, ID`) and $\mathcal{R}_2$(`ID, occupation, income`) respectively.

Figure 1(b) illustrates the proposed DBN scheme: Once the two partitions are decomposed, we combine the resulting core and factor matrices to obtain the decomposition of the original tensor corresponding to the relation $\mathcal{R}$.

In the above example, if the relation $\mathcal{R}$ is *dense*, we expect that decompositions of relations $\mathcal{R}_1$ and $\mathcal{R}_2$ will be much faster than that of the relation $\mathcal{R}$ and the gain will more than compensate for the normalization cost of the initial step and the combination cost of the last step of DBN.

If the 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. As a result, unless the partitioning provides smaller number of tuples for both relations, DBN may not provide sufficient savings to compensate the normalization and combination overheads. However, as we will experimentally verify in Section 6, the decomposition and recombination steps of the DBN scheme fit naturally in multiple cores. This leads to a highly efficient, effective, and parallelized DBN strategy under both dense and sparse tensors.

In general, a relation can be vertically partitioned into two in many ways: we need to first select a join attribute and then, we need to decide which attributes to include in which vertical partition. The join attribute (`ID` in the above example) around which we partition the data is important for two different reasons. 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 need to partition the data in such a way that the later step in which decompositions of the individual partitions are combined into an overall decomposition does not introduce errors.

Task 1.1. 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.* This is difficult because the total number of functional dependencies in the data can be exponential.

Task 1.2. A second difficulty is that many data sets may not have perfect functional dependencies to leverage for normalization. *In that case, we need to identify and rely on approximate functional dependencies in the data.* In this paper, we argue that we can rely on pairwise approximate functional dependencies that are incomplete, yet very efficient to compute.

Task 1.3. Once the approximate functional dependencies are identified, *we need 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 recombination step. 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.

1.2 Organization of the Paper

The organization of the paper is as follows:

- We first provide the relevant background and discuss the related works in Section 2.
- We next provide an overview of the proposed DBN scheme in Section 3.
- We then focus on selecting the best partitions for the normalization step of DBN (Section 4).
- In Section 5, we present rank-pruning strategies to further reduce the cost of DBN.
- Next, we experimentally evaluate DBN in Section 6 in both stand-alone and parallel configurations. We focus on the accuracy and the running time of the alternative algorithms. Experimental results provide evidence that while being significantly faster, DBN can approximate well the fitness of the conventional tensor decomposition with respect to the original tensor.

We conclude the paper in Section 7.

2. BACKGROUND AND RELATED WORK

Tensor Decomposition. The two most popular tensor decompositions are the Tucker [21] and the CANDECOMP/PARAFAC [7, 4] decompositions. The Tucker decomposition generalizes singular value matrix decomposition (SVD) to higher-dimensional matrices. CANDECOMP [4] and PARAFAC [7] 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 CP Decomposition of $\mathcal{P}$ is a weighted sum of rank-one tensors as the following (an alternative way to view the decomposition is in the form of a diagonal tensor and a set of factor matrices of the input tensor):

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

where λ is a vector of size r and each $U^{(n)}$ is a matrix of size $I_n \times r$, for $n = 1, \cdots, N$.

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 [4, 7]. ALS estimates, at each iteration, one factor matrix, maintaining other matrices fixed; this process is repeated for each factor matrix associated to the dimensions of the input tensor.

Nonnegative Tensor Decomposition (NTF). Note that these decompositions can be interpreted probabilistically, if additional constraints (nonnegativity and summation to 1) are imposed. In the case of the CP decomposition, for example, each nonzero element in the core and the values of entries of the factor matrices can be thought of as a cluster and the conditional probabilities of the entries given clusters, respectively. The N-way Toolbox for MATLAB [1] provides both CP and Tucker decomposition with nonnegativity constraints.

Scalable Tensor Decomposition. Tensor decomposition is a costly process. In dense tensor representation where the cost increases exponentially with the number of modes of the tensor. While decomposition cost increases more slowly (linearly with the number of nonzero entries in the tensor) for sparse tensors, the operation can still be very expensive for large data sets due to the high computational cost and memory requirement to build up the approximate tensor [17]. A modified ALS algorithm proposed in [17] computes Hadamard products instead of Khatri-Rao products for efficient PARAFAC for large-scale tensors. Kolda *et al.* [11] developed a greedy PARAFAC algorithm for large-scale, sparse tensors in MATLAB.

The ALS in Tucker decompositions involves SVD which is the most computationally challenging part [20]. To address this, randomized sampling is used in [20]. MET decomposition proposed in [12] addresses intermediate blowup problem in Tucker decomposition.

Parallel Tensor Decomposition. Phan and Cichocki [17] 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 combined into the final decomposition. The grid PARAFAC can be converted to grid NTF by enforcing nonnegativity.

Zhang *et al.* [23] parallelized NTF for mining global climate data in such a way that the original 3-mode tensor is divided into three semi-NMF sub-problems based on ALS approach and these matrices are distributed to independent processors to facilitate parallelization. Antikainen *et al.* [2] presented an algorithm for NTF that is specialized for Compute Uniform Device Architecture (CUDA) framework that provides a parallel running environment.

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

Functional Dependency (FD). A functional dependency (FD) is a constraint between two sets of attributes X and Y in a relation denoted by $X \to Y$, which specifies that the values of the X component of a tuple uniquely determine the values of the Y component.

The discovery of FDs in a data set is a challenging problem since the complexity increases exponentially in the number of attributes [15]. Many algorithms for FD and approximate FD discovery exist [8, 13, 15, 22]. TANE proposed in [8] used the definition of approximate FDs based on the minimum fraction of tuples that should be removed from the relation to hold the exact FDs.

The computation of FDs in TANE and Dep-Miner [13] is based on levelwise search [16]. Dep-Miner finds the minimal FD cover of a hypergraph using a levelwise search. Similarly, FastFD [22] finds the minimal cover, however, differently from Dep-Miner, it uses a depth-first search to address the problem in the levelwise search that the cost increases exponentially in the number of attributes. The main factor in the cost of FastFD is the input size. FastFD works well when the number of attributes is large. TANE takes linear time with respect to the size of the input whereas FastFD takes more than linear time of the input size.

CORDS [9] generalized FDs to determine statistical dependencies, which is referred to as soft FD. In a soft FD, a value of an attribute determines a value of another attribute with high probability. CORDS only discovers pairwise correlations reducing a great amount of complexity that nevertheless can remove most of correlation-induced selectivity error. In this paper, we also leverage pairwise FDs to measure dependency *between* partitions (interFD) and *within* a partition (intraFD).

3. DECOMPOSITION-BY-NORMALIZATION (DBN)

As we discussed earlier, our goal in this paper is to tackle the high computational cost of decomposition process. Since, especially for dense data sets, the number of modes of the tensor data is one of the main factors contributing to the cost of tensor operations, we focus on how we reduce the dimensionality and the size of the input tensor. In particular, we argue that a large relation can be vertically partitioned into multiple relations relying on the approximate functional dependencies in the data, and the decompositions of these relations can be combined to obtain the overall decomposition. Without loss of generality, in this paper, we consider

2-way partitioning of the input data. The process can easily be generalized for multiple partitions.

In this section, we provide an overview of this DBN process. We first introduce the relevant notations and provide background on key concepts.

3.1 Background and Key Concepts

Without loss of generality, we assume that relations are represented in the form of *occurrence tensors.*

3.1.1 Tensor Representation of Relational Data

Let $\mathrm{A}_1, \ldots, \mathrm{A}_n$ be a set of attributes in the schema of a relation, R, and $\mathrm{D}_1, \ldots, \mathrm{D}_n$ be the attribute domains. Let the relation instance $\mathcal{R}$ be a finite multi-set of tuples, where each tuple $t \in \mathrm{D}_1 \times \ldots \times \mathrm{D}_n$.

Definition 3.1 (Occurrence Tensor). *An occurrence tensor $\boldsymbol{\mathcal{R}}_o$ corresponding to the relation instance $\mathcal{R}$ is an n-mode tensor, where each attribute $\mathrm{A}_1, \ldots, \mathrm{A}_n$ is represented by a mode. For the i^{th} mode, which corresponds to A_i, let $\mathrm{D}'_i \subseteq \mathrm{D}_i$ be the (finite) subset of the elements such that*

$$\forall v \in \mathrm{D}'_i \; \exists t \in \mathcal{R} \;\; s.t. \;\; t.\mathrm{A}_i = v$$

and let $idx(v)$ denote the rank of v among the values 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 *$\boldsymbol{\mathcal{R}}_o$ are such that*

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

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 Normalization (Vertical Partitoning) and Functional Dependencies

First introduced by Codd [5], the normalization process of the relational model evaluates relations based on the underlying functional dependencies of the data and vertically partitions a large relation into smaller relations (with lesser number of attributes) to minimize redundancy and insertion, deletion, and update anomalies.

Functional Dependency (FD). The functional dependency, between two sets of attributes $\mathcal{X}$ and $\mathcal{Y}$, in a relational model is defined as the following.

Definition 3.2 (Functional dependency). *A functional dependency (FD), denoted by $\mathcal{X} \to \mathcal{Y}$, holds for relation instance $\mathcal{R}$, if and only if for any two tuples t_1 and t_2 in $\mathcal{R}$ that have $t_1[\mathcal{X}] = t_2[\mathcal{X}]$, $t_1[\mathcal{Y}] = t_2[\mathcal{Y}]$ also holds.*

The key idea of the normalization process is that if $\mathcal{A} = \{\mathrm{A}_1, \ldots, \mathrm{A}_n\}$ is a set of attributes in the schema of a relation, R, and $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{A}$ are two subsets of attributes such that $\mathcal{X} \to \mathcal{Y}$, then the relation instance $\mathcal{R}$ can be vertically partitioned into two relation instances $\mathcal{R}_1$, with attributes $\mathcal{A} \setminus \mathcal{Y}$, and $\mathcal{R}_2$, with attributes $\mathcal{X} \cup \mathcal{Y}$, such that $\mathcal{R} = \mathcal{R}_1 \bowtie \mathcal{R}_2$; in other words the set of attributes $\mathcal{X}$ serves as a foreign key and joining vertical partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ on $\mathcal{X}$ gives back the relation instance $\mathcal{R}$ without any missing or spurious tuples. Note that, while for some data sets, FDs are known at the database design time, for many data sets they need to be discovered by analyzing the available data sets.

Approximate FD. As described above, the search for exact FDs is a costly process. Moreover, in many data sets, attributes may not have perfect FDs due to exceptions and outliers in the data. In such cases, we may search for approximate FDs instead of exact FDs.

Definition 3.3 (Approximate FD). *An approximate FD (aFD), denoted by $\mathcal{X} \overset{\sigma}{\to} \mathcal{Y}$ 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 $t_1[\mathcal{X}] = t_2[\mathcal{X}]$, $t_1[\mathcal{Y}] = t_2[\mathcal{Y}]$ 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 $\mathcal{X} \overset{\sigma}{\to} \mathcal{Y}$.

As described above, the concept of relational normalization (vertical partitioning) relies on the relational join operation. Since in this paper we represent relations as occurrence tensors, we also need to define a corresponding *tensor join* operation that operates in the domain of tensors.

3.1.3 Tensor Join

Let $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{Q}}$ be two tensors, representing relation instances $\mathcal{P}$ and $\mathcal{Q}$, with attribute sets, $\mathrm{A}^P = \{\mathrm{A}^P_1, \ldots, \mathrm{A}^P_n\}$ and $\mathrm{A}^Q = \{\mathrm{A}^Q_1, \ldots, \mathrm{A}^Q_m\}$, respectively. In the rest of this section, we denote the index of each cell of $\boldsymbol{\mathcal{P}}$ as $(i_1, i_2, ..., i_n)$; similarly, the index of each cell of $\boldsymbol{\mathcal{Q}}$ is denoted as $(j_1, j_2, ..., j_m)$. The cell indexed as $(i_1, \ldots, i_n)$ of $\boldsymbol{\mathcal{P}}$ is denoted by $\boldsymbol{\mathcal{P}}[i_1, \ldots, i_n]$ and the cell indexed as $(j_1, \ldots, j_m)$ of $\boldsymbol{\mathcal{Q}}$ is denoted by $\boldsymbol{\mathcal{Q}}[j_1, \ldots, j_m]$.

Definition 3.4 (Join ($\bowtie$)). *In relational algebra, given two relations $\mathcal{P}$ and $\mathcal{Q}$, and a condition φ, the* join *operation is defined as a cartesian product of the input relations followed by the selection operation. Therefore, given two relational tensors $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{Q}}$, and a condition φ, we can define their join as*

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

Given two relations $\mathcal{P}$ and $\mathcal{Q}$, with attribute sets, $\mathrm{A}^P = \{\mathrm{A}^P_1, \ldots, \mathrm{A}^P_n\}$ and $\mathrm{A}^Q = \{\mathrm{A}^Q_1, \ldots, \mathrm{A}^Q_m\}$, and a set of attributes $\mathrm{A} \subseteq \mathrm{A}^P$ and $\mathrm{A} \subseteq \mathrm{A}^Q$, the equi-join *operation, $\bowtie_{=,\mathrm{A}}$, 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* A *from the resulting relation.*

3.2 Overview of the Decomposition-by-Normalization (DBN) Process

The pseudo-code of the DBN algorithm is shown in Figure 2. In this subsection, we provide an overview of the steps of the algorithm. Then, in the following sections, we study in detail the key steps of the process.

In its first step, DBN evaluates the pairwise FDs among the attributes of the input relation. For this purpose, we employ and extend TANE [8], an efficient algorithm for the discovery of FDs. Our modification of the TANE algorithm returns a set of (approximate) FDs between the attribute pairs and, for each candidate dependency, $A_i \to A_j$, it provides the corresponding support, $\sigma_{i,j}$.

The next steps involves selecting the attribute, X, that will serve as the foreign key and partitioning the input relation $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$ around X. Note that if the selected

```
DBN algorithm (input: a relation $\mathcal{R}$)
1: Evaluate pFD (pairwise FDs between all 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(pFD,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(pFD,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 con-
          strained 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: Decompose the tensors corresponding to $\mathcal{R}_1$ and $\mathcal{R}_2$, join the de-
    compositions to obtain candidate combined decompositions, and
    select the one that is likely to provide the best fit to $\mathcal{R}$
```

Figure 2: Pseudo-code of DBN (see Section 4)

```
findInterFDPartition ( input: pFD, balanced)
1: Create a complete pairwise FD graph, where the nodes are at-
   tributes and edge weights are the support values of pFD.
2: if balanced == false then
3:    Run minimum average cut on the pairwise FD graph to find a
      maximally independent partitioning (Desideratum 5)
4: else {i.e., balanced == true}
5:    Run minimum average cut on the pairwise FD graph to find a
      balanced, maximally independent partitioning (Desideratum 3
      and 5)
6: end if
```

Figure 3: Pseudo-code of interFD-based partition algorithm; this is detailed in Section 4.3

join attribute X does not perfectly determine the attributes of $\mathcal{R}_1$, then to prevent introduction of spurious tuples, we will need to remove sufficient number of outlier tuples from $\mathcal{R}$ to enforce the FDs between the attribute, X, and the attributes selected to be moved to $\mathcal{R}_1$.

Desideratum 1: Therefore, to prevent over-thinning of the relation $\mathcal{R}$, the considered approximate FDs need to have high support; i.e., $\sigma_{i,j} \geq \tau_{support}$, for a sufficiently large support lower-bound, $\tau_{support}$.

We next consider various criteria for vertical partitioning of the input relation:

- First of all, as we discussed earlier, when we vertically partition the relation $\mathcal{R}$ with attributes $\mathcal{A} = \{A_1, \ldots, A_n\}$ 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 vertical partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ on X gives back $\mathcal{R}$ without any missing or spurious tuples.

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

- Since for dense tensors, the complexity of the decomposition process is exponential in the number of modes [11], we would prefer that the number of attributes in each partition should be balanced.

  Desideratum 3: For dense tensors, the vertical partitioning should be such that the number of attributes of $\mathcal{R}_1$ and $\mathcal{R}_2$ are similar.

- For sparse tensors the number of modes impact the decomposition cost only linearly. In fact, in this case, the major contributor to the decomposition cost is the number of nonzero entries in the tensor [11].

  Desideratum 4: Therefore, for sparse tensors, the vertical partitioning should be such that the total number of tuples of $\mathcal{R}_1$ and $\mathcal{R}_2$ are minimized.

- Any information encoded with the FDs crossing relations $\mathcal{R}_1$ and $\mathcal{R}_2$ is risked to be lost when $\mathcal{R}_1$ and $\mathcal{R}_2$ are individually decomposed in the final step of the DBN algorithm. 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.

We elaborate on these desiderata and their implications on the vertical partitioning of $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$ in Section 4.

Once $\mathcal{R}_1$ and $\mathcal{R}_2$ are obtained, the final steps of the DBN process include decomposition of the tensors corresponding to $\mathcal{R}_1$ and $\mathcal{R}_2$, joining of the decompositions to obtain candidate combined decompositions, and selecting the one that is likely to provide the best fit to $\mathcal{R}$. For these, we rely on the `join-by-decomposition` scheme proposed in [10]: in order to construct a rank-r decomposition of a joined tensor, `join-by-decomposition` first finds all rank-r_1 and rank-r_2 decompositions of the two input tensors, such that $r_1 \times r_2 = r$. The rank-r_1 and rank-r_2 decompositions of the input decompositions are then combined along the given factor matrix, which corresponds to the join attribute in the equi-join operation. Finally the algorithm finds the $r_1 \times r_2$ pair which provides the least approximation error, given all possible factorizations, $r_1 \times r_2$ of r. Note that `join-by-decomposition` involves creation of multiple alternative join pairs (corresponding to different $r_1 \times r_2$ factorizations of r), which are independently evaluated for accuracy and the one that is predicted to provide the best accuracy is used for obtaining the final result. This provides a natural way to parallelize the entire operation by associating each pair of rank decompositions (and the computation of the corresponding pair selection measure) to a different processor core. In Section 6, we also evaluate the applicability of such parallelizations in the context of the DBN operation.

4. VERTICAL PARTITIONING STRATEGIES

Based on the desiderata discussed in Section 3.2, here we discuss the vertical partitioning strategies for sparse and dense tensors. For each situation, we consider the cases: (Case 1) the join attribute (approximately) determines a subset of attributes and (Case 2) it determines all attributes of the input relation.

4.1 Sparse Tensors

For the first case (i.e., 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. Since inter-partition FDs of the form $X \rightarrow *$ are all less than $\tau_{support}$, this also reflects Desideratum 5.

Note that by construction, the size of $\mathcal{R}_2$ is equal to the number of tuples in $\mathcal{R}$ independent of which attributes are included in it. The size of $\mathcal{R}_1$, on the other hand, can be minimized (to satisfy Desideratum 4) down to the number of unique values of X by eliminating all the duplicate tuples.

For the second case, where the join attribute X determines all attributes of $\mathcal{R}$, we apply the interFD-based vertical partitioning strategy detailed later in Section 4.3.

4.2 Dense Tensors

For the first case, similarly to sparse tensors, we create partitions, $\mathcal{R}_1$ and $\mathcal{R}_2$, by considering the FDs of the form $X \rightarrow *$, where X is the join attribute. Differently from the sparse tensors, in this case we also consider Desideratum 3, which prefers balanced partitions:

- When there are fewer attributes with support higher than $\tau_{support}$, we move the attributes with the highest support from $\mathcal{R}_2$ to $\mathcal{R}_1$ to promote balance. This is equivalent to relaxing the support threshold.
- When there are more attributes with support higher than $\tau_{support}$, we move the attributes with the lowest support from $\mathcal{R}_1$ to $\mathcal{R}_2$ to promote balance. This is equivalent to tightening the support threshold.

For the second case, as in the sparse tensors, we apply the interFD-based vertical partitioning strategy detailed later in Section 4.3. The major difference is that in this case, we also promote balance.

4.3 Vertical Partitioning Based on Supports of Attribute Pair Dependencies

Desideratum 5 implies that the vertical partitioning strategy should minimize inter-partition FDs. As discussed earlier, this helps minimize the likelihood of error when the individual partitions are decomposed and the resulting decompositions are combined to obtain the overall decomposition. Kim and Candan [10] have shown that, given a join operation, $\mathcal{R} = \mathcal{R}_1 \bowtie_A \mathcal{R}_2$, it is possible to obtain a rank-r decomposition of $\mathcal{R}$ by combining rank-r_1 and rank-r_2 decompositions of $\mathcal{R}_1$ and $\mathcal{R}_2$, as long as $r = r_1 \times r_2$ and that rank r_1 and r_2 decompositions of the input tensors lead to clusters that are independent relative to the join attribute, A. Authors have also argued theoretically and experimentally that the accuracy of the decomposition is especially high if the other attributes of the two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ are independent from each other. Relying on this observation (which we also validate in Section 6), 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.

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. We refer to this as *interFD* and the partitioning based on interFD as the interFD-based partitioning strategy.

We formulate this interFD-based partitioning problem as a graph partitioning problem: 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 and the weights of the edge between nodes v_i and v_j is the average support of the approximate FDs $v_i \rightarrow v_j$ and $v_j \rightarrow v_i$. The problem is then to locate a cut on G_{pfd} with the minimum *average* weight.

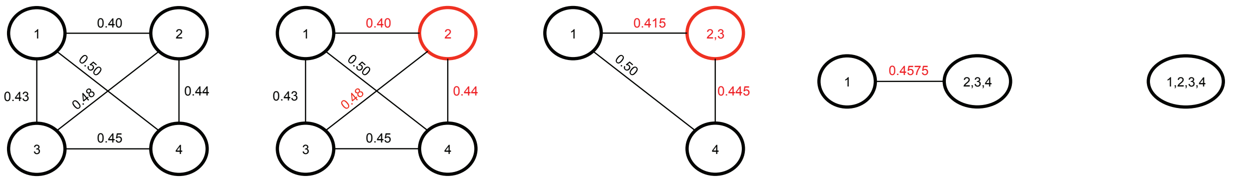
This graph partitioning problem is similar to the minimum cut problem [19], with some key differences. The major difference is that we do not seek a cut with minimum *total* weight, but a cut with minimum *average* weight. Also, depending on whether we are operating on dense or sparse networks, we may or may not seek to impose a balance criterion on the partitions: for sparse tensors, the cost of tensor decomposition increases linearly with the number of modes and since the total number of modes of the two partitions is constant, we do not need to seek balance.

In DBN, we use a modified version of the minimum cut algorithm proposed in [19] to seek minimum average cuts. Given an undirected graph $G_{pfd}(V, E)$, for each vertex $v \in V$, the algorithm finds a vertex with the *minimum average cut* that separates it from the rest of the graph and creates a subset of vertices $V^{'}$ where the vertex is merged with its neighbor vertex, connected to the rest of the graph with the least average weight; the edges from these two vertices are replaced by a new edge weighted by the *average* of the weights of the original edges. The process is repeated while $V^{'} \neq V$. The minimum of the minimum average cuts at each step of the algorithm is returned as the overall minimum average cut. When balance of the number of attributes is needed, the minimum is selected among the steps that lead to similar number of attributes. The complexity of this minimum average cut algorithm is $O(|V||E| + |V|^2 log|V|)$. Figure 4 shows an example of the process.

5. RANK PRUNING BASED ON INTRA-PARTITION DEPENDENCIES

Given a partitioning of $\mathcal{R}$ into of $\mathcal{R}_1$ and $\mathcal{R}_2$, to obtain a rank-r decomposition of $\mathcal{R}$, 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 $\langle r_1, r_2 \rangle$ pair which is likely to minimize recombination errors.

We note, however, that we can rely on the supports of the dependencies that make up the partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ to prune $\langle r_1, r_2 \rangle$ 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



(a) $G_{pfd}(V, E)$ (b) Step 1 (c) Step 2 (d) Step 3 (e) Step 4

Figure 4: Minimum average cut algorithm for (a) $G_{pfd}(V, E)$. (b) Step 1: a cut between {2} and {1,3,4} is the minimum average cut (0.44). (c) Step 2: vertex 2 is merged with vertex 3 which has the minimum average weight to the rest of the vertices and a cut between {2,3} and {1,4} is the minimum average cut (0.43). (d) Step 3: vertices 2 and 3 are merged with vertex 4 and a cut between {1} and {2,3,4} is the minimum average cut (0.4575). (e) Step 4: the process ends since the subset of vertices is equal to V. The minimum of the minimum average cuts at each step is (c) {2,3} and {1,4}.

Data set		Size
D1	Adult	$118 \times 90 \times 20263 \times 5 \times 2$
D2		$7 \times 20263 \times 5 \times 6 \times 16$
D3		$72 \times 20263 \times 90 \times 2 \times 2$
D4		$20263 \times 14 \times 2 \times 6 \times 94$
D5		$20263 \times 5 \times 2 \times 90 \times 72$
D6	Breast Cancer Wisconsin [14]	$645 \times 10 \times 11 \times 2 \times 10$
D7		$10 \times 645 \times 9 \times 10 \times 10$
D8		$10 \times 10 \times 11 \times 10 \times 645$
D9		$2 \times 10 \times 10 \times 10 \times 645$
D10		$10 \times 10 \times 645 \times 9 \times 10$
D11	IPUMS Census Database [18]	$3890 \times 4 \times 13 \times 3 \times 3$
D12		$545 \times 3 \times 17 \times 3 \times 2$
D13		$11 \times 3 \times 4 \times 5 \times 3$
D14	Mushroom	$10 \times 3 \times 5 \times 2 \times 7$
D15	Dermatology	$62 \times 5 \times 5 \times 5 \times 3$

Table 1: Relational tensor data sets with 5 attributes

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

Table 2: Different attribute sets, join attributes (X), supports of X (the lowest of all the supports of $X \to *$), and execution times for FDs discovery for D1-D15 where A_n is the n^{th} attribute of each data set

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 $\langle r_1, r_2 \rangle$, 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 the next section, we evaluate the effect of the interFD-based partitioning and intraFD-based rank pruning strategy of DBN for both dense and sparse tensor decomposition in terms of the efficiency and the accuracy.

6. EXPERIMENTAL EVALUATION

Here, we present experimental results assessing the efficiency and effectiveness of the proposed DBN scheme relative to the conventional implementation of the tensor decomposition in both stand-alone and parallelized versions. We used various data sets from UCI Machine Learning Repository [6].

6.1 Partitioning Cases

We consider two partitioning cases discussed in Section 4:

Case 1. Firstly, we evaluate DBN for the case where the join attribute X determines only a subset of the attributes of the relation $\mathcal{R}$. In this case, as long as one partition $\mathcal{R}_1$ has X and the determined attributes of X, the number of nonzero entries of $\mathcal{R}_1$ is less than or equal to that of $\mathcal{R}$ and the number of nonzero entries of the other partition $\mathcal{R}_2$ is same as that of $\mathcal{R}$. For dense tensor decomposition, we make the number of attributes of $\mathcal{R}_1$ and $\mathcal{R}_2$ similar.

For this case, we used a 5-mode relational tensor of dimensions $118 \times 90 \times 20263 \times 5 \times 2$ of the Adult data set (D1). We ran TANE [8] to find FDs on this data set. TANE identified almost exact FDs $A_3 \to A_9$ and $A_3 \to A_{10}$ (with 99% and 98% confidence respectively) where A_n is the n^{th} attribute. Now we take A_3 as the join attribute and remove unmatched tuples (1.72%) for these two FDs to get the exact FDs. Next the tensor is normalized into two 3-mode tensors $\mathcal{R}_1$ $\{A_3, A_9, A_{10}\}$ and $\mathcal{R}_2$ $\{A_3, A_{11}, A_{12}\}$. We then create relational tensors corresponding to different table sizes by randomly selecting entries from the data.

Case 2. Secondly, we evaluate DBN in the case where the join attribute X determines all attributes of the relation $\mathcal{R}$. In this case, all partitioning cases generate as many tuple (nonzero entries) as the relation $\mathcal{R}$ and thus, it is not possible to select a configuration which better minimizes the total number of nonzero entries than the rest. In this case, for each pairwise FD support above a support threshold, $\tau_{support} = 75\%$, we take an attribute with the highest total pairwise FD support among all attributes of the input relation as the join attribute. For these experiments, we took 15 different data sets (D1-D15) with different sizes and different attribute sets (see Table 1) and we experimented all partitioning cases with the same number of tuples for $\mathcal{R}$, $\mathcal{R}_1$, and $\mathcal{R}_2$ for each data set.

All tensors were encoded as occurrence tensors (see Definition 3.1), where each entry is set to 1 or 0, indicating whether the corresponding tuple exists or not. Therefore, we report the number of nonzero entries. We also report the tensor size for the dense tensor decomposition since the decomposition cost of dense tensor decomposition depends on the size of dense tensors.

We experimented with rank-12 decompositions. DBN uses 6 combinations (1×12, 2×6, 3×4, 4×3, 6×2, and 12×1) for rank-12 decomposition for each relation in Table 1.

6.2 Implementation

Discovery of FDs. We employed TANE [8] and made an extension to detect approximate FDs for all pairs of attributes for pairwise FDs and remove all unmatched tuples to get the exact FDs for the normalization process based on the supports of the approximate FDs.

The supports of the approximate FDs for each attribute set of different relational data sets are shown in Table 2. The table also shows the execution times to discover FDs for each data set. The modified TANE algorithm is efficient when the number of attributes is small (5 attributes in these experiments). The overall cost increases linearly in the size of the input [8]. Since the execution times for finding approximate FDs are negligible compared to the tensor decomposition time, we focus on the decomposition times.

Tensor Decomposition. We experimented with alternative algorithms for both nonnegative CP tensor decomposition on the original tensors (NNCP) and DBN. Table 3 shows the various algorithms we use in our experiments.

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 [1]. We refer to DBN and NNCP using 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, especially for sparse tensors. Another PARAFAC implementation, the CP-ALS algorithm [3], on the other hand, can run with both sparse and dense tensors. In the sparse tensor model, the cost increases linearly as the number of nonzero entries of the tensor increases. The CP-ALS, however, does not support nonnegative constraints. Therefore, we implemented a variant of the single grid NTF [17] using CP-ALS as the base PARAFAC algorithm. We refer to DBN and NNCP based on CP-ALS as DBN-CP and NNCP-CP respectively.

For the parallel version of the NNCP, we implemented the grid NTF algorithm [17] with two different partition strategies (2 and 6 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 and results are iteratively combined into the final decomposition based on an ALS-like approach. We refer to the grid NTF algorithm for the parallel NNCP using N-way PARAFAC with 2 and 6 grid cells as NNCP-NWAY-GRID2, and NNCP-NWAY-GRID6 respectively. Similarly, we refer to CP-ALS based implementations as NNCP-CP-GRID2 and NNCP-CP-GRID6.

For the parallel version of DBN, we implemented pairwise parallel DBN-NWAY and DBN-CP, referred to as pp-DBN-NWAY and pp-DBN-CP, respectively.

Finally, for DBN with a subset of pairs, DBN with 2 and 3 pairs selected based on the intraFD-based rank pruning strategy are referred to as DBN2 and DBN3 respectively. For all alternative DBN strategies can be DBN2 or DBN3

Algorithm	Description
DBN-NWAY	DBN using N-way PARAFAC
DBN-CP	DBN using single grid NTF (CP-ALS)
NNCP-NWAY	NNCP using N-way PARAFAC
NNCP-CP	NNCP using single grid NTF (CP-ALS)
pp-DBN-NWAY	pairwise parallel DBN-NWAY
NNCP-NWAY-GRID2	NNCP using grid NTF with 2 grid cells (N-way PARAFAC)
NNCP-NWAY-GRID6	NNCP using grid NTF with 6 grid cells (N-way PARAFAC)
pp-DBN-CP	pairwise parallel DBN-CP
NNCP-CP-GRID2	NNCP-CP with 2 grid cells
NNCP-CP-GRID6	NNCP-CP with 6 grid cells
DBN2	intraFD-based DBN with 2 pairs
DBN3	intraFD-based DBN with 3 pairs

Table 3: Algorithms. Note that the decomposition algorithms in parentheses are the base PARAFAC algorithm for the grid NTF.

according to the number of pairs selected (e.g., DBN2-CP for DBN-CP with 2 pairs selected).

We ran our experiments on an 6 cores Intel(R) Xeon(R) CPU X5355 @ 2.66GHz with 24GB of RAM. We used MATLAB Version 7.11.0.584 (R2010b) 64-bit (glnxa64) for the general implementation and MATLAB Parallel Computing Toolbox for the parallel implementation of DBN and NNCP.

6.2.1 *Accuracy*

We use the following *fit* function to measure tensor decomposition accuracy: The fit measure is defined as

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

where $\|\mathcal{X}\|$ is the Frobenius norm of a tensor $\mathcal{X}$. The fit is a normalized measure of how accurate a tensor decomposition of $\mathcal{X}$, $\hat{\mathcal{X}}$ with respect to a tensor $\mathcal{X}$.

$\|\hat{\mathcal{X}}\|$ is also used as an approximate fit measure of $\mathcal{X}$ to substitute for fit computation for large data sets. The intuition behind this measure is as follows: For any $\mathcal{W}$ it can be shown that $\|\mathcal{W} - \hat{\mathcal{W}}\| \geq \|\mathcal{W}\| - \|\hat{\mathcal{W}}\|$. Therefore, as long as $\|\mathcal{W}\| \geq \|\hat{\mathcal{W}}\|$ holds, we can minimize the term $\|\mathcal{W} - \hat{\mathcal{W}}\|$ by maximizing $\|\hat{\mathcal{W}}\|$.

Our evaluation criteria also include *fit ratio* which indicates how accurate one strategy compared with the other strategy in terms of *fit* to the input tensor. For example, the fit_ratio of DBN to NNCP is defined as

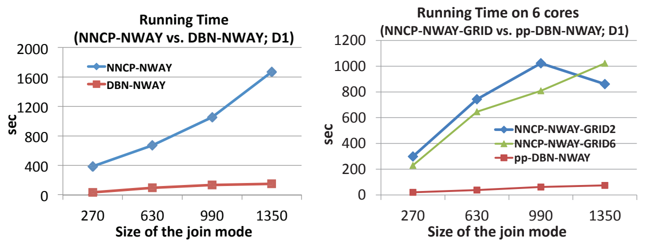
$$\mathrm{fit\_ratio} = \frac{\mathrm{fit}(\mathcal{X}, \hat{\mathcal{X}}_{dbn})}{\mathrm{fit}(\mathcal{X}, \hat{\mathcal{X}}_{nncp})} \tag{2}$$

where $\mathcal{X}$ is the input tensor, $\hat{\mathcal{X}}_{nncp}$ is the tensor obtained by re-composing the NNCP tensor, and $\hat{\mathcal{X}}_{dbn}$ is the tensor obtained by re-composing the DBN tensor.

6.3 Results

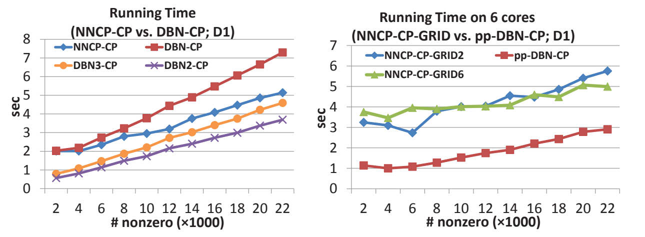
6.3.1 *Execution time*

As discussed in Section 6.1, we evaluate the execution times of DBN vs. the conventional nonnegative CP (NNCP) algorithms (Table 3) for two partitioning cases: in the first case, the join attribute X determines only a subset of the attributes of the relation $\mathcal{R}$; that is, $\mathrm{nnz}(\mathcal{R}_1) \leq \mathrm{nnz}(\mathcal{R})$ and $\mathrm{nnz}(\mathcal{R}_2) = \mathrm{nnz}(\mathcal{R})$ where $\mathrm{nnz}(\mathcal{X})$ denotes the number of nonzero entries of $\mathcal{X}$. In the second case, the join attribute X



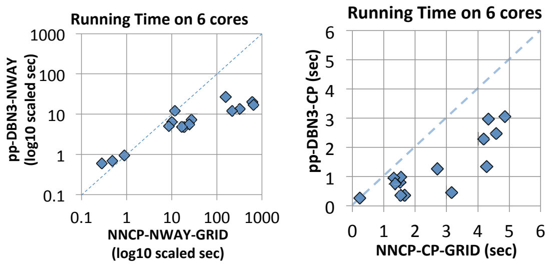
(a) (b)

Figure 5: Average running times of (a) NNCP-NWAY vs. DBN-NWAY and (b) NNCP-NWAY-GRID vs. pp-DBN-NWAY on 6 cores for Adult data set (D1)



(a) (b)

Figure 6: Average running times of (a) NNCP-CP vs. DBN-CP with different pair selections and (b) NNCP-CP-GRID vs. pp-DBN-CP on 6 cores for Adult data set (D1)



(a) (b)

Figure 7: Average running times of (a) NNCP-NWAY-GRID (avg of GRID2 and GRID6) vs. pp-DBN3-NWAY and (b) NNCP-CP-GRID vs. pp-DBN3-CP on 6 cores for D1-D15. In both cases, most of data points are located under the diagonal, which indicates that DBN outperforms NNCP

determines all attributes of the relation $\mathcal{R}$; that is nnz($\mathcal{R}_1$) = nnz($\mathcal{R}_2$) = nnz($\mathcal{R}$).

Case 1: X does not determine all attributes of $\mathcal{R}$.
First we present the execution times for dense tensor decomposition. As seen in Figure 5, for NNCP-NWAY, as the tensor size increases, the execution time increases fast. DBN-NWAY, running on smaller number of modes than NNCP, saves significant amount of time.

Figure 5 (b) shows the running time of NNCP-NWAY-GRID2 and NNCP-NWAY-GRID6 vs. pp-DBN-NWAY on 6 cores. Comparing the execution times in the figure against the single core execution times in Figure 5 (a) shows that the DBN-NWAY benefits more from parallelization. Since the grid NTF divides a larger tensor into smaller sub-tensors but with the same number of modes as the original tensor, parallelization does not save as much as in pp-DBN-NWAY where a larger tensor is divided into sub-tensors with a smaller number of modes.

Figure 6 presents results for sparse tensor decompositions, NNCP-CP vs. DBN-CP with different pair selections on single-core and NNCP-CP-GRID vs. pp-DBN-CP on 6 cores. Since we select a foreign key for normalizing the input relation, one of the normalized relations in DBN-CP has the same number of nonzero entries as that of the original relation. As a result, when using a single core, the sparse tensor DBN time (which is determined by the number of tuples) exceeds the time required to decompose the original relation. As discussed in Section 5, we address this using the intraFD-based rank pruning strategy. Figure 6 (a) shows that we achieve the better performance by choosing 2 pairs or 3 pairs in DBN based on intraFD.

Furthermore, as shown in Figure 6 (b), when using multiple cores, the proposed pp-DBN-CP achieves greater execution time savings and, therefore, significantly outperforms parallelized versions of NNCP. As seen in the figure, pp-DBN-CP has the lowest execution time. The grid based parallelization of NNCP does not improve the running time much, since the underlying ALS-based combining approach involves significant communication overheads.

Case 2: X determines all attributes of $\mathcal{R}$.
Since in this case, selection of a configuration is especially difficult, we also consider intraFD-based rank pruning strategy that can maximize the performance of DBN. We experiment NNCP-NWAY-GRID (avg. of NNCP-NWAY-GRID2 and NNCP-NWAY-GRID6) vs. pp-DBN3-NWAY for dense tensor decomposition and NNCP-CP-GRID (avg. of NNCP-CP-GRID2 and NNCP-CP-GRID6) vs. pp-DBN3-CP for sparse tensor decomposition for D1-D15 on 6 cores. We use the average of all different partitioning cases for each data set. As shown in Figure 7, in most cases of both experiments, DBN outperforms NNCP, especially for dense tensor decomposition. In few cases, the running times are too small to make a significant difference between NNCP and DBN.

6.3.2 *Result Accuracy*

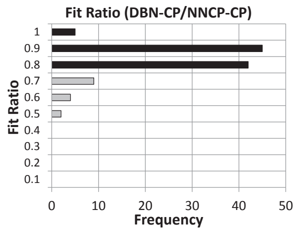
We compare the accuracy of DBN against standard NNCP. Note that we include results for DBN-CP and NNCP-CP (results for DBN-NWAY and NNCP-NWAY are similar). We measured the fit ratios for all different partitioning cases of D1-D15 (Table 1). Note that fit computation requires a lot of memory (the computation is not feasible for some larger datasets). Thus, for large datasets with the size of the join mode bigger than 1,000, we created subsets of each tensor by sampling random 1,000 entries from the join mode and compute fit for NNCP-CP and DBN-CP.

As shown in Figure 8 (a), for an overwhelming majority of the experimented data sets, fit ratio falls in the range between 0.8 and 1 indicating that the proposed DBN scheme is, in addition to being efficient, also highly effective.

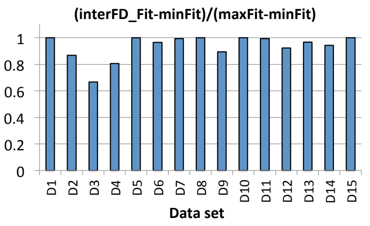
6.3.3 *InterFD-based Partitioning*

As discussed in Section 4.3, the proposed DBN strategy first identifies alternative normalized partitions of the relational tensor and then selects the most promising pair of partitions to compute the final decomposition. More specifically, the algorithm picks the most independent partitions according to the interFD.

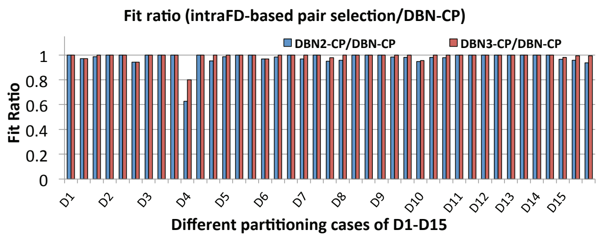
In these experiments, we use 15 different configurations in Table 1. In order to quantify the benefits of the interFD-based partitioning strategy, we measure the ratio of the difference between the approximate fits of interFD-based partitions and the minimum approximate fits against the difference between the maximum and minimum approximate fits among all partitioning alternatives of each data set.



(a)



(b)



(c)

Figure 8: (a) Frequencies of fit ratios DBN-CP/NNCP-CP for all partitioning cases (108 cases) of D1-D15 (Table 1), (b) ratios of the difference between the approximate fits of interFD-based partitions and the minimum approximate fits against the difference between the maximum and minimum approximate fits among different partitioning cases of each data set, D1-D15, and (c) approximate fit ratio of DBN-CP with 2 and 3 pairs to DBN-CP with original 6 pairs for the partitioning cases (41 cases) of D1-D15 where each partition has more than 2 attributes

Results shown in Figure 8 (b) indicates that the interFD-based partitioning strategy results in fits very close to the optimal partitioning strategy. The few cases where the interFD-based strategy provides only $70 - 80\%$ of the optimal attribute partitioning is largely due to the pairwise nature of interFD. This tends to miss multi-attribute dependencies of the form $AB \rightarrow C$ with high support, e.g., in D3, we miss $A_1A_3 \rightarrow A_{15}$ whose support is 94%.

6.3.4 IntraFD-based Rank Pruning

We compare the approximate fit of DBN-CP with 2 and 3 pairs selected based on the intraFD-based rank pruning strategy against that of DBN-CP using the original 6 pairs. Note that we use only the cases (41 cases) where each subtensor has more than 2 attributes (only the join attribute and a determined attribute) for each data set since the intraFD does not include pairwise FDs involving the join attribute. As shown in Figure 8 (c), the intraFD-based rank pruning strategy of DBN-CP has a good accuracy. The approximate fit ratios in the most of cases of DBN3-CP are close to 1; in other words, one of the 3 pairs in DBN3-CP gives the best approximate fit. Note that the data set D4 (with partitions $\{A_3, A_7, A_{15}\}$ and $\{A_3, A_8, A_{13}\}$) gives the worst approximate fit for both DBN2-CP and DBN3-CP, with fit ratios 0.63 and 0.80 respectively. This is because, the intraFD strategy with only pairwise dependencies may on occasion fail to take into account critical dependencies, such as $A_3A_7 \rightarrow A_{15}$ with 92% support.

7. CONCLUSIONS

Lifecycle of most data includes various operations, from capture, integration, projection, to data decomposition and analysis. To address the high cost of tensor decompsition, which is highest among these operations, we proposed a highly efficient, effective, and parallelized DBN strategy for approximately evaluating decompositions by normalizing a large relation into the smaller tensors based on the FDs of the relation and then performing the decompositions of these smaller tensors. We also proposed interFD-based partitioning and intraFD-based rank pruning strategies for DBN based on pairwise FDs across the normalized partitions and within each normalized partition, respectively.

Experimental results confirmed the efficiency and effectiveness of the proposed DBN scheme, and its interFD and intraFD based optimization strategies, compared to the conventional NNCP approach.

8. REFERENCES

[1] C. A. Andersson and R. Bro. The N-way Toolbox for MATLAB. Chemometr. Intell. Lab., 52(1):1-4, 2000.
[2] J. Antikainen *et al.* Nonnegative Tensor Factorization Accelerated Using GPGPU. In TPDS, 22(7):1135-1141, 2011. National Laboratories, 2006.
[3] B. W. Bader and T. G. Kolda. MATLAB Tensor Toolbox Ver. 2.2, 2007.
[4] J. D. Carroll and J. J. Chang. Analysis of individual differences in multidimensional scaling via an N-way generalization of 'Eckart-Young' decomposition. Psychometrika, 35:283-319, 1970.
[5] E. F. Codd. Further normalization of the data base relational model. In Rustin, 1972.
[6] A. Frank and A. Asuncion. UCI Machine Learning Repository. Irvine, CA: U. of California, School of ICS, 2010.
[7] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. UCLA working papers in phonetics, 1970.
[8] Y. Huhtala *et al.* TANE: An efficient algorithm for discovering functional and approximate dependencies. Comput. J., 42 (2):100-111, 1999.
[9] I. F. Ilyas *et al.* CORDS: Automatic discovery of correlations and soft functional dependencies. In SIGMOD, 647-658, 2004.
[10] M. Kim and K. S. Candan. Approximate tensor decomposition within a tensor-relational algebraic framework. In CIKM, 2011.
[11] T. G. Kolda *et al.* Higher-order web link analysis using multilinear algebra. In ICDM, 2005.
[12] T. G. Kolda and J. Sun. Scalable tensor decompositions for multi-aspect data mining. In ICDM, 363-372, 2008.
[13] S. Lopes *et al.* Efficient discovery of functional dependencies and armstrong relations. In EDBT, 2000.
[14] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. SIAM News, 23(5):1-18, 1990.
[15] H. Mannila and K. Räihä. On the complexity of inferring functional dependencies. Discrete Appl. Math., 40:237-243, 1992.
[16] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. Data Min. Knowl. Discov., 1(3):259-289, 1997.
[17] A. H. Phan and A. Cichocki. PARAFAC algorithms for large-scale problems. Neurocomputing, 74(11):1970-1984, 2011.
[18] S. Ruggles and M. Sobek. Integrated Public Use Microdata Series: Version 2.0 Minneapolis: Historical Census Projects, U. of Minnesota, 1997.
[19] M. Stoer and F. Wagner. A simple min-cut algorithm. J. ACM, 44 (4):585-59, 1997.
[20] C. E. Tsourakakis. Mach: Fast randomized tensor decompositions. In SDM, 2010.
[21] L. R. Tucker. Some mathematical notes on three-mode factor analysis. Psychometrika, 31, 1966.
[22] C. M. Wyss *et al.* FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In DaWak, 2001.
[23] Q. Zhang *et al.* A parallel nonnegative tensor factorization algorithm for mining global climate data. ICCS, 2009.