Pushing-Down Tensor Decompositions over Unions to Promote Reuse of Materialized Decompositions*

Mijung Kim and K. Selçuk Candan

Arizona State University, USA
{mijung.kim.1,candan}@asu.edu

Abstract. From data collection to decision making, the *life cycle* of data often involves many steps of integration, manipulation, and analysis. To be able to provide end-to-end support for the full data life cycle, today's data management and decision making systems increasingly combine operations for data manipulation, integration as well as data analysis. Tensor-relational model (TRM) is a framework proposed to support both relational algebraic operations (for data manipulation and integration) and tensor algebraic operations (for data analysis). In this paper, we consider joint processing of relational algebraic and tensor analysis operations. In particular, we focus on data processing workflows that involve data integration from multiple sources (through unions) and tensor decomposition tasks. While, in traditional relational algebra, the costliest operation is known to be the join, in a framework that provides both relational and tensor operations, tensor decomposition tends to be the computationally costliest operation. Therefore, it is most critical to reduce the cost of the tensor decomposition task by manipulating the data processing workflow in a way that reduces the cost of the tensor decomposition step. Therefore, in this paper, we consider data processing workflows involving tensor decomposition and union operations and we propose a novel scheme for pushing down the tensor decompositions over the union operations to reduce the overall data processing times and to promote reuse of materialized tensor decomposition results. Experimental results confirm the efficiency and effectiveness of the proposed scheme.

1 Introduction

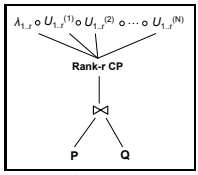
As a higher-order generalization of matrices, tensors provide a suitable data representation for multidimensional data sets and tensor decomposition (which is a higher-order generalization of SVD/PCA for multi-aspect data analysis) helps capture the higher-order latent structure of such datasets. Consequently, the tensor data model is increasingly being used by many application domains including scientific data management [6,9,18,25], sensor data management [24], and social network data analysis [15,14,17]. On the other hand, from data collection to decision making, the *life cycle* of data often involves many steps of integration, manipulation, and analysis. Therefore, to be able to provide end-to-end support for the full data life cycle, today's data management and decision making systems increasingly need to combine different types of operations for data manipulation, integration, and analysis.

* This work is partially funded by NSF grants #116394, RanKloud: Data Partitioning and Resource Allocation Strategies for Scalable Multimedia and Social Media Analysis and #1016921, One Size Does Not Fit All: Empowering the User with User-Driven Integration.

T. Calders et al. (Eds.): ECML PKDD 2014, Part I, LNCS 8724, pp. 688–704, 2014.

Relational operation	Tensor manipulation
Select	Slicing of a tensor (or taking a single or subset of elements across a given mode)
Project	Creating a sub-cube with a smaller set of modes
Cartesian-Product and Equi-Join	Composition of multiple tensors through outer-product
Union	Cell-wise OR (and row/slice insertion)
Intersection	Cell-wise AND (and row/slice elimination)

(a)



(b)

Fig. 1. (a) Implementation of relational operations through tensor manipulation and (b) a query plan with a join operation of two tensors, $\mathcal{P}$ and $\mathcal{Q}$, preceding a tensor decomposition operation

We are currently building TensorDB, which extends a native array database, SciDB [5], with operations needed to support the full *life cycle* of data. TensorDB is based on a tensor-relational model (TRM) [11], which brings together relational algebraic operations (for data manipulation and integration) and tensor algebraic operations (for data analysis) and supports complex data processing plans where multiple relational algebraic and tensor algebraic operations are composed with each other (Figure 1(b)).

1.1 Tensor-Based Relational Model (TRM)

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 $t \in \mathrm{D}_1 \times \ldots \times \mathrm{D}_n$. [11] defines various types of tensors representing relations, including occurrence tensors and value tensor. For example, an *occurrence tensor* $\mathcal{R}_o$ corresponding to the relation instance $\mathcal{R}$ as an n-mode tensor, where each attribute $\mathrm{A}_1, \ldots, \mathrm{A}_n$ is represented by a mode. For the ith 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* $\mathcal{R}_o$ are such that $\mathcal{R}_o[u_1, \ldots, u_n] = 1$ iff $\exists t \in \mathcal{R} \;\; s.t. \;\; \forall_{1 \leq j \leq n} \; idx(t.\mathrm{A}_j) = u_j$ and 0 otherwise. Intuitively, each cell indicates whether the corresponding tuple exists in the multi-set corresponding to the relation or not.

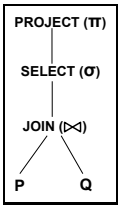
[11] also discusses the implementation of various relational algebraic operations to manipulate relations represented as tensors in TRM (Figure 1(a)) as well as other tensor manipulation operations, such as tensor decomposition.

1.2 Tensors Decomposition

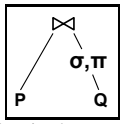
The two most popular tensor decompositions are the CP [6,9] and Tucker [25] decompositions. CP decomposes the input tensor into a sum of component rank-one tensors; i.e., the rank-r CP Decomposition, $CP(\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N})$, of the tensor $\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}$ is defined as $\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(N)}$ such that

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

We also use the formulation where the column vectors of each factor are normalized to the unit length with the weights absorbed into a vector λ; i.e., $CP(\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}) =$



(a) Logical query plan



(b) physical query plan
(selection, and projection are pushed-down)

Fig. 2. Query optimization in relational algebra: (a) A logical query plan involving selection, projection, and join operations: (b) an equivalent physical plan where the selection and projection operations are *pushed-down* to minimize the amount of data fed into the join operator

$\langle \lambda, \mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(N)} \rangle$, such that

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

where λ_i is the ith element of vector λ of size r and $U_i^{(n)}$ is the ith unit-length column vector of the matrix $\mathbf{P}^{(n)}$ of size $I_n \times r$, for $n = 1, \cdots, N$.

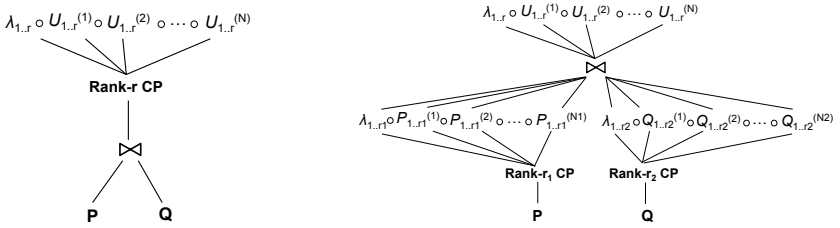
While, as described above, CP decomposition can be represented in the form of a diagonal core tensor and one factor matrix (also called a factor) per mode, the Tucker decomposition results in a dense core tensor multiplied by a matrix along each mode. Many of the algorithms for decomposing tensors are based on an iterative process, such as alternating least squares (ALS), that approximates the solution though iterations until a convergence condition is reached [6,9].

1.3 Decomposition Push-Down Strategy for Optimizating TRM Workflows

One key goal of TensorDB is to deploy optimization strategies for complex queries involving both tensor decomposition and tensor manipulation operations, such as join and union operations that integrate data from multiple sources.

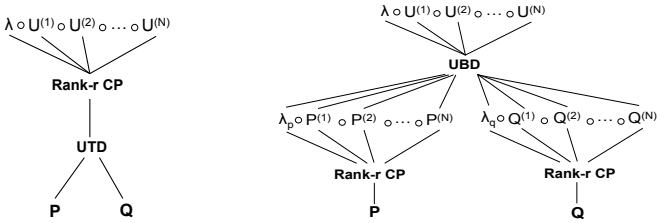
In relational algebra, the costliest operation is the join operation. Consequently, given a complex query plan, the relational optimizers *push-down* data reduction operations, such as selections (which reduce the number of tuples) and projections (which reduce the number of data attributes) over join-operations to reduce the amount of data fed into the join operators (Figure 2). In TensorDB, based on TRM, however, tensor decomposition operation tends to be the computationally costliest operation: for dense tensors, the cost is exponential in the number of modes of the data. While the operation is relatively cheaper for sparse tensors, the cost and memory requirement still outweigh other more traditional relational operators.

Therefore, *a key criterion for optimizing query workplans in TensorDB is to reduce the number of data modes and non-zero data entries in the tensors that need to be decomposed.* In [11], for example, we considered query plans that involve join operations and tensor decompositions (Figures 1(b) and 3(a)) and proposed a *decomposition push-down* strategy that reduces the number of modes of the data tensors being decomposed. This *join-by-decomposition* (JBD) strategy *pushes-down* the tensor-decomposition operation so that the input tensors (which have smaller number of modes than the join



(a) Join-then-Decompose (JTD) (b) Join-by-Decomposition (JBD)

Fig. 3. (a) The query plan in Figure 1(b) and (b) an alternative query plan where the tensor decomposition operation is pushed-down [11]



(a) Union-then-Decompose (UTD) (b) Union-by-Decomposition (UBD)

Fig. 4. (a) A query plan with an union of two tensors, $\mathcal{P}$ and $\mathcal{Q}$ preceding tensor decomposition and (b) an alternative query plan where the decomposition is pushed-down over union

tensor) are decomposed into their spectral components and then these decompositions are combined to obtain the final decomposition as shown in Figure 3(b).

In this paper, *we focus on query plans that involve tensor decomposition and union operations (as in Figure 4(a)) and propose novel* decomposition push-down *strategies (as in Figure 4(b)) that help reduce the overall cost of the query plan.* We refer to the query plan that first performs the union operation on the data and then applies the tensor decomposition on the union of the data as *union-then-decompose* (UTD) plan. The query plan with decomposition push-down, which first performs the tensor decompositions on each input data source and then combines these decomposed tensors as the *union-by-decomposition* (UBD) plan.

1.4 Contributions of This Paper: Union-by-Decomposition (UBD)

A *union-by-decomposition* (UBD) plan, with decomposition push-down, has various advantages over the conventional *union-then-decompose* (UTD) plan:

– Firstly, especially when the overlaps between the input data sources are small, the union operation can combine relatively small and sparse tensors into a larger and denser tensor. Consequently, the decomposition over the union data can be much more expensive than the decompositions over the input data sources. Moreover multiple tensor decompositions on input tensors can run in parallel, which will further reduce the cost.

– Secondly, a *union-by-decomposition* (UBD) based plan provides opportunities for materializing decomposition of data tensors and re-using these materialized decompositions in more complex queries requiring integration of data.

Despite these advantages, however, implementing the UBD strategy requires us to address a number of key challenges:

– **Challenge 1: How can we combine the factor matrices of tensor decompositions with their own eigen basis into the eigen basis of the union tensor?** If tensor decomposition is thought of as a group of clusters, combining different groups of clusters for different tensors into another group of clusters for the union of the tensors is not straightforward.
– **Challenge 2: For the common data elements at the intersection of multiple data sources, which factors (clusters when the clustering analogy is used) among the different tensor decompositions should we choose?** This is critical as the choice can impact the final accuracy of the UBD based plan.

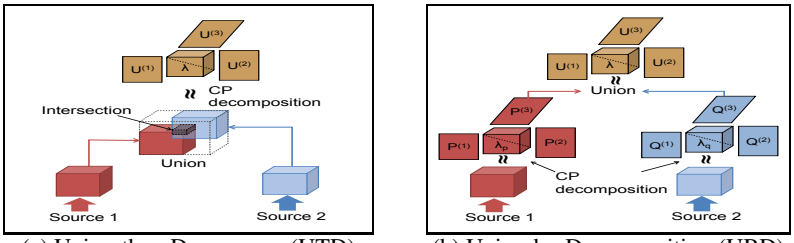
In this paper, we present algorithms and techniques to address these questions. We first review the related work in Section 2. In Section 3, we extend TRM with the proposed *union-by-decomposition* operation: we discuss strategies for combining the tensor decompositions for the union of the tensors from different sources and consider alternative selection measures to choose a group of factors for data entries common to input data sources. We also consider query plans that include both join and union operations along with tensor decomposition. We, then, experimentally evaluate the proposed scheme in Section 5 and conclude the paper in Section 6.

2 Related Work

2.1 Tensors and Tensor Decomposition

The two most popular tensor decompositions are the CANDECOMP/PARAFAC (CP [6,9]) and Tucker [25] decompositions. CANDECOMP [6] and PARAFAC [9] decompositions (together known as the CP decomposition) decompose the input tensor into a sum of component rank-one tensors. While CP decomposition can be represented in the form of a diagonal core tensor and one factor matrix (also called a factor) per mode, the Tucker decomposition results in a dense core tensor. Many of the algorithms for decomposing tensors are based on an iterative alternating least squares (ALS) process that approximates the solution by iteratively improving the decomposition until a convergence condition is reached [6,9]. In [21], the complexity of ALS schemes has been discussed. Non-iterative approaches to tensor decomposition include closed form solutions, such as generalized rank annihilation method (GRAM) [19] and direct trilinear decomposition (DTLD) [20], which fit the model by solving a generalized eigenvalue problem. [13] provides an overview of the tensor decomposition algorithms.

Tensor decomposition is a costly process. In dense tensor representation, 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. [24] uses



(a) Union-then-Decompose (UTD) (b) Union-by-Decomposition (UBD)

Fig. 5. (a) Tensor decomposition on the union of the two relations and (b) the union operation on the two tensor decompositions of the input relations

randomized sampling to approximate the tensor decomposition where the tensor does not fit in the available memory. A modified ALS algorithm proposed in [18] computes Hadamard products instead of Khatri-Rao products for efficient PARAFAC for large-scale tensors. [15] developed a greedy PARAFAC algorithm for large-scale, sparse tensors in MATLAB. [14] proposed a memory-efficient Tucker (MET) decomposition to address the intermediate blowup problem in Tucker decomposition. A parallelization strategy of tensor decomposition on MapReduce has been proposed in [10]. ParCube proposed in [17] is a parallelizable tensor decomposition algorithm, which produces sparse approximation of tensor decompositions. In [11,12], we proposed parallelized tensor decompositions within a tensor relational algebraic framework.

2.2 Array Databases

There are several in-database data models for modeling tensor data. Column-oriented organizations [22] are efficient when many or all rows are accessed, such as during an aggregate computation. Row-oriented organizations, on the other hand, are efficient when many or all of the columns on a single row are accessed or written on a single disk seek. Key-value organizations [1] are useful when working with less structured data, such as documents, which tend not to be relational. The array model [4,5,8,26] is a natural representation to store multidimensional data and facilitate multidimensional data analysis. Approaches to represent array based data can be broadly categorized into four types. (a) The first approach is to represent the array in the form of a table [7,26]. (b) A second approach is to use blob type in a relational database as a storage layer for array data [4,8]. (c) Sparse matrices can also be represented using a graph-based abstraction [16]. For example, in [16], ALS (alternating least squares) is solved using a graph algorithm that represents a sparse matrix as a bipartite graph. (d) The last approach is to consider a native array model and an array-based storage scheme, such as a chunk-store, as in [5].

3 Union-by-Decomposition (UBD) and Decomposition Push-Down

In this section, we describe our proposed union-by-decomposition (UBD) approach that pushes down tensor decompositions over union operators: Unlike the more conventional

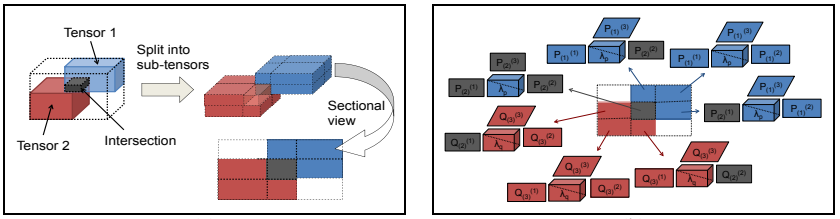


Fig. 6. Naive grid-based UBD: (a) Input tensors are partitioned into an intersecting sub-tensor and non-intersecting sub-tensors; (b) intermediary decompositions of grid-based UBD

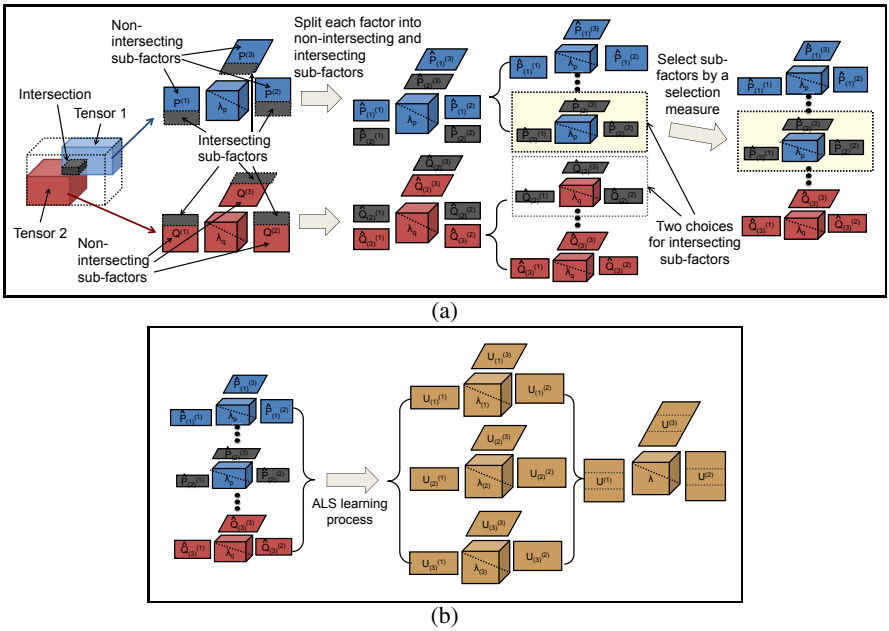


Fig. 7. UBD: (a) first the inputs tensors are decomposed and (b) these decompositions are recombined by considering the common and non-intersection parts of the factor matrices.

union-then-decompose (UTD) scheme, which applies decomposition on the union of the two relations (Figure 5(a)), UBD first performs the tensor decomposition on the input tensors then these decompositions are combined into the final result (Figure 5(b)).

3.1 Challenge 1: Implementing UBD through Partition-Based ALS

Naive Grid-Based UBD. One way to implement the UBD operation is to divide the input tensors into *common (or intersection)* and ($2^N - 1$ many when the number of modes is N) *uncommon* sub-tensors as shown in Figure 6(a) and then considering each partition as a cell of a larger tensor partitioned into a grid as shown in Figure 6(b) and applying the grid-based tensor decomposition strategy proposed in [18] to combine these into a single decomposition.

Algorithm 1. Union-By-Decomposition (UBD) (input: two tensors $\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}$ and $\mathcal{Q}_{J_1 \times J_2 \times \cdots \times J_N}$, optional input: CP decompositions of $\mathcal{P}$ and $\mathcal{Q}$, $\langle \mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(N)} \rangle$ and $\langle \mathbf{Q}^{(1)}, \ldots, \mathbf{Q}^{(N)} \rangle$, respectively, output: factors $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}$ for $\mathcal{P} \cup \mathcal{Q}$)

1: **if** no existing decompositions given **then**
2: Run any available CP algorithm on $\mathcal{P}$ and $\mathcal{Q}$ *in parallel* to get factors $\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(N)}$ and $\mathbf{Q}^{(1)}, \ldots, \mathbf{Q}^{(N)}$
3: **end if**
4: **for** each mode n **do**
5: create sub-factors $\hat{\mathbf{P}}^{(n)}_{(1)}$ and $\hat{\mathbf{P}}^{(n)}_{(2)}$, and $\hat{\mathbf{Q}}^{(n)}_{(2)}$ and $\hat{\mathbf{Q}}^{(n)}_{(3)}$ with non-intersecting and intersecting sub-factors of $\mathbf{P}^{(n)}$ and $\mathbf{Q}^{(n)}$, respectively (see Figure 7(a))
6: **end for**
7: select either $\hat{\mathbf{P}}^{(n)}_{(2)}$ and $\hat{\mathbf{Q}}^{(n)}_{(2)}$ for factors $\mathbf{T}^{(n)}$ for intersection $\mathcal{P} \cap \mathcal{Q}$ by a selection measure (see Section 3.2)
8: repeat the update process for sub-factors $\mathbf{U}^{(n)}_{(1)}$, $\mathbf{U}^{(n)}_{(2)}$, and $\mathbf{U}^{(n)}_{(3)}$ using Equation 9 until a stopping condition is satisfied, which are combined to $\mathbf{U}^{(n)}$ by Equation 7 (see Figure 7(b))

Proposed Implementation of UBD. An obvious shortcoming of the naive grid-based UBD discussed above is that it leads to a very large number of intermediary decompositions and this number increases quickly with the number of modes of the input tensors. To tackle this challenge, we propose to decompose input tensors directly (through decomposition push-down) and recombine the resulting factor matrices in a way that reflects the common and non-intersecting sub-factors of these decompositions as shown in Figure 7. The high-level pseudocode of this partition-based UBD scheme is shown in Algorithm 1. We next present the details of the proposed UBD process:

Let us assume that we are given two tensors $\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}$ and $\mathcal{Q}_{J_1 \times J_2 \times \cdots \times J_N}$ and let us assume we have already computed their CP decompositions

$$CP(\mathcal{P}) = \hat{\mathcal{P}} = \langle \mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(N)} \rangle \quad \text{and} \quad CP(\mathcal{Q}) = \hat{\mathcal{Q}} = \langle \mathbf{Q}^{(1)}, \ldots, \mathbf{Q}^{(N)} \rangle. \tag{3}$$

Our goal is to estimate $CP(\mathcal{P} \cup \mathcal{Q}) = \langle \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \rangle$ efficiently using these decompositions. To achieve this, we solve the ALS problem

$$\min \| (\mathcal{P} \cup \mathcal{Q}) - \langle \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \rangle \| \tag{4}$$

by appropriately combining sub-factors of the input tensors. More specifically, each factor of $\mathcal{P}$ and $\mathcal{Q}$ are split into two: a non-intersecting ($\mathbf{P}^{(n)}_{(1)}$ and $\mathbf{Q}^{(n)}_{(3)}$) and intersecting ($\mathbf{P}^{(n)}_{(2)}$ and $\mathbf{Q}^{(n)}_{(2)}$) partitions. Given these, the CP decompositions of $[k_1, k_2, \ldots, k_N]$-th sub-tensor of $\mathcal{P}$ and $\mathcal{Q}$ are

$$CP(\mathcal{P}^{(\bar{\mathbf{k}})}) = \langle \mathbf{P}^{(1)}_{(k_1)}, \ldots, \mathbf{P}^{(N)}_{(k_N)} \rangle \quad \text{and} \quad CP(\mathcal{Q}^{(\bar{\mathbf{k}})}) = \langle \mathbf{Q}^{(1)}_{(k_1)}, \ldots, \mathbf{Q}^{(N)}_{(k_N)} \rangle, \tag{5}$$

respectively, where $\bar{\mathbf{k}} = [k_1, k_2, \ldots, k_N]$ for $k_n \in \{1, 2\}$ for $\mathcal{P}^{(\bar{\mathbf{k}})}$ and $k_n \in \{2, 3\}$ for $\mathcal{Q}^{(\bar{\mathbf{k}})}$. Given these, we can approximate the decompositions of each sub-tensor of $\mathcal{P}$ and $\mathcal{Q}$ with the CP decompositions of $\mathcal{P}$ and $\mathcal{Q}$, respectively (see Figure 7(a)):

$$CP(\mathcal{P}^{(\bar{\mathbf{k}})}) \approx \langle \hat{\mathbf{P}}^{(1)}_{(k_1)}, \ldots, \hat{\mathbf{P}}^{(N)}_{(k_N)} \rangle \quad \text{and} \quad CP(\mathcal{Q}^{(\bar{\mathbf{k}})}) \approx \langle \hat{\mathbf{Q}}^{(1)}_{(k_1)}, \ldots, \hat{\mathbf{Q}}^{(N)}_{(k_N)} \rangle. \tag{6}$$

Let us denote the CP decomposition of $[k_1, k_2, \ldots, k_N]$-th sub-tensor of $\mathcal{P} \cup \mathcal{Q}$ as

$$CP((\mathcal{P} \cup \mathcal{Q})^{(\bar{\mathbf{k}})}) = CP(\mathcal{Y}^{(\bar{\mathbf{k}})}) = \langle \mathbf{U}^{(1)}_{(k_1)}, \ldots, \mathbf{U}^{(N)}_{(k_N)} \rangle,$$

where $\bar{\mathbf{k}} = [k_1, k_2, \ldots, k_N]$ for $k_n \in \{1, 2, 3\}$. Note that each factor of $CP(\mathcal{P} \cup \mathcal{Q})$ can be split into three partitions

$$\mathbf{U}^{(n)} = [\mathbf{U}^{(n)T}_{(1)} \mathbf{U}^{(n)T}_{(2)} \mathbf{U}^{(n)T}_{(3)}]^T, \tag{7}$$

one corresponding to a non-intersecting sub-factor from one input matrix, the other corresponding to a common sub-factor, and the last corresponding to a non-intersecting sub-factor from the second input matrix. Given these, we can re-formulate the minimization problem in Equation (4) for each sub-tensor $\mathcal{Y}^{(\bar{\mathbf{k}})}$ of $\mathcal{P} \cup \mathcal{Q}$ as minimizing D, where

$$D = \frac{1}{2} \sum_{k_1=1}^{3} \cdots \sum_{k_N=1}^{3} \|\mathcal{Y}^{(\bar{\mathbf{k}})} - \langle \mathbf{U}^{(1)}_{(k_1)}, \ldots, \mathbf{U}^{(N)}_{(k_N)} \rangle\|,$$

or, considering the n-mode matricized tensor $\mathbf{Y}^{(\bar{\mathbf{k}})}_{(n)}$ of $\mathcal{Y}^{(\bar{\mathbf{k}})}$, as minimizing

$$D = \frac{1}{2} \sum_{\bar{\mathbf{k}}} \|\mathbf{Y}^{(\bar{\mathbf{k}})}_{(n)} - \mathbf{U}^{(n)}_{(k_n)} \{\mathbf{U}^{(1)}_{(k_1)} \odot \mathbf{U}^{(2)}_{(k_2)} \odot \cdots \odot \mathbf{U}^{(n-1)}_{(k_{n-1})} \odot \mathbf{U}^{(n+1)}_{(k_{n+1})} \odot \cdots \odot \mathbf{U}^{(N)}_{(k_N)}\}^T\|,$$

where $\odot$ is the Khatri-Rao product.

This minimization problem can be solved using an ALS problem by identifying gradient components with respect to sub-factors as in [18]. More specifically, the gradient component with respect to sub-factor $\mathbf{U}^{(n)}_{(k_n)}$ is

$$\begin{aligned} \Delta_{\mathbf{U}^{(n)}_{(k_n)}} D &= \sum_{\bar{\mathbf{k}}_n = k_n} \left(-\mathbf{Y}^{(\bar{\mathbf{k}})}_{(n)} \mathbf{U}^{\odot_{-n}}_{(\bar{\mathbf{k}})} + \mathbf{U}^{(n)}_{(k_n)} \mathbf{U}^{\odot_{-n}T}_{(\bar{\mathbf{k}})} \mathbf{U}^{\odot_{-n}}_{(\bar{\mathbf{k}})} \right) \\ &= \sum_{\bar{\mathbf{k}}_n = k_n} \left(-\mathbf{Y}^{(\bar{\mathbf{k}})}_{(n)} \mathbf{U}^{\odot_{-n}}_{(\bar{\mathbf{k}})} + \mathbf{U}^{(n)}_{(k_n)} \{\mathbf{U}^{T}_{(\bar{\mathbf{k}})} \mathbf{U}_{(\bar{\mathbf{k}})}\}^{\circledast_{-n}} \right), \end{aligned} \tag{8}$$

where $\circledast$ is the Hadamard (element-wise) product. Given this, each sub-factor $\mathbf{U}^{(n)}_{(k_n)}$ can be updated using the update rule

$$\mathbf{U}^{(n)}_{(k_n)} \leftarrow \left(\sum_{\bar{\mathbf{k}}_n = k_n} \mathbf{Y}^{(\bar{\mathbf{k}})}_{(n)} \mathbf{U}^{\odot_{-n}}_{(\bar{\mathbf{k}})} \right) \left(\sum_{\bar{\mathbf{k}}_n = k_n} (\mathbf{U}^{T}_{(\bar{\mathbf{k}})} \mathbf{U}_{(\bar{\mathbf{k}})})^{\circledast_{-n}} \right)^{-1}. \tag{9}$$

Note that, from Equation 6, for each sub-tensor $\mathcal{Y}^{(\bar{\mathbf{k}})} = \mathcal{P}^{(\bar{\mathbf{k}})}$, considering to the first input matrix we have

$$\mathbf{Y}^{(\bar{\mathbf{k}})}_{(n)} \mathbf{U}^{\odot_{-n}}_{(\bar{\mathbf{k}})} \approx \hat{\mathbf{P}}^{(n)}_{(k_n)} \hat{\mathbf{P}}^{\odot_{-n}T}_{(\bar{\mathbf{k}})} \mathbf{U}^{\odot_{(-n)}}_{(\bar{\mathbf{k}})}. \tag{10}$$

Similarly, for each sub-tensor $\mathcal{Y}^{(\bar{\mathbf{k}})} = \mathcal{Q}^{(\bar{\mathbf{k}})}$, considering to the second input matrix, we have

$$\mathbf{Y}^{(\bar{\mathbf{k}})}_{(n)} \mathbf{U}^{\odot_{-n}}_{(\bar{\mathbf{k}})} \approx \hat{\mathbf{Q}}^{(n)}_{(k_n)} \hat{\mathbf{Q}}^{\odot_{-n}T}_{(\bar{\mathbf{k}})} \mathbf{U}^{\odot_{(-n)}}_{(\bar{\mathbf{k}})}. \tag{11}$$

Finally, for each sub-tensor $\mathcal{Y}^{(\bar{\mathbf{k}})}$ such that $\mathcal{Y}^{(\bar{\mathbf{k}})} = \mathcal{P} \cap \mathcal{Q}$,

$$\mathbf{Y}^{(\bar{\mathbf{k}})}_{(n)} \mathbf{U}^{\odot_{-n}}_{(\bar{\mathbf{k}})} \approx \mathbf{T}^{(n)} \mathbf{T}^{\odot_{-n}T} \mathbf{U}^{\odot_{(-n)}}_{(\bar{\mathbf{k}})}, \tag{12}$$

where $\mathbf{T}^{(n)}$ are the factors of $CP(\mathcal{P} \cap \mathcal{Q})$. Note that $\mathbf{T}^{(n)}$ can be estimated from either the CP decomposition of $\mathcal{P}^{(\bar{\mathbf{2}})}$

$$CP(\mathcal{P} \cap \mathcal{Q}) = CP(\mathcal{P}^{(\bar{\mathbf{2}})}) \approx \langle \hat{\mathbf{P}}^{(1)}_{(2)}, \ldots, \hat{\mathbf{P}}^{(N)}_{(2)} \rangle,$$

where $\bar{\mathbf{2}} = [k_1, k_2, \ldots, k_N]$ for all $k_n = 2$, or the CP decomposition of $\mathcal{Q}^{(\bar{\mathbf{2}})}$

$$CP(\mathcal{P} \cap \mathcal{Q}) = CP(\mathcal{Q}^{(\bar{\mathbf{2}})}) \approx \langle \hat{\mathbf{Q}}^{(1)}_{(2)}, \ldots, \hat{\mathbf{Q}}^{(N)}_{(2)} \rangle.$$

The choice is critical and can impact significantly on the accuracy of the overall process. Therefore, we next discuss how to select whether to use $\hat{\mathbf{P}}^{(n)}_{(2)}$ or $\hat{\mathbf{Q}}^{(n)}_{(2)}$ to estimate $\mathbf{T}^{(n)}$.

3.2 Challenge 2: Selection of Sub-factors for the Overlapping Sub-tensor

As described above, the factors $\mathbf{T}^{(n)}$ of the overlapping sub-tensor, $\mathcal{P} \cap \mathcal{Q}$ (used in the computation of $CP(\mathcal{P} \cup \mathcal{Q})$) can be selected from either $\hat{\mathbf{P}}^{(n)}_{(2)}$ or $\hat{\mathbf{Q}}^{(n)}_{(2)}$. As also explained before, the choice is critical as it may impact the accuracy of the final decomposition, $CP(\mathcal{P} \cup \mathcal{Q})$. Therefore, in this subsection, we explore alternative ways for choosing the sub-factors, $\mathbf{T}^{(n)}$, of $CP(\mathcal{P} \cap \mathcal{Q})$.

Intersection-Based Selection Criteria. When we are choosing between $\hat{\mathbf{P}}^{(n)}_{(2)}$ and $\hat{\mathbf{Q}}^{(n)}_{(2)}$ to use as $\mathbf{T}^{(n)}$, one criteria would be to consider how well $\hat{\mathcal{P}}^{(\bar{\mathbf{2}})} = \langle \hat{\mathbf{P}}^{(1)}_{(2)} \ldots \hat{\mathbf{P}}^{(N)}_{(2)} \rangle$ and $\hat{\mathcal{Q}}^{(\bar{\mathbf{2}})} = \langle \hat{\mathbf{Q}}^{(1)}_{(2)} \ldots \hat{\mathbf{Q}}^{(N)}_{(2)} \rangle$ fit $\mathcal{P} \cap \mathcal{Q}$:

$$IC_1(\hat{\mathcal{P}}^{(\bar{\mathbf{2}})}) = 1 - \frac{\|(\mathcal{P} \cap \mathcal{Q}) - \hat{\mathcal{P}}^{(\bar{\mathbf{2}})}\|}{\|\mathcal{P} \cap \mathcal{Q}\|} \text{ and } IC_1(\hat{\mathcal{Q}}^{(\bar{\mathbf{2}})}) = 1 - \frac{\|(\mathcal{P} \cap \mathcal{Q}) - \hat{\mathcal{Q}}^{(\bar{\mathbf{2}})}\|}{\|\mathcal{P} \cap \mathcal{Q}\|}.$$

One obvious difficulty with this fit-based intersection criterion, IC_1, is that the fit computations can be very costly. Alternatively, if we consider the two tensor decompositions, $\hat{\mathcal{P}}^{(\bar{\mathbf{2}})}$ and $\hat{\mathcal{Q}}^{(\bar{\mathbf{2}})}$ as two groups of clusters, then we need to choose the group of clusters on which the membership of the shared elements (the overlapping part) is more tight and we can use the norms of the sub-factors to quantify how strongly elements belongs to the corresponding clusters. Intuitively, norms of the sub-factors corresponding to the overlapping region

$$IC_2(\hat{\mathcal{P}}^{(\bar{\mathbf{2}})}) = \|\langle \hat{\mathbf{P}}^{(1)}_{(2)}, \ldots, \hat{\mathbf{P}}^{(N)}_{(2)} \rangle\|, \quad IC_2(\hat{\mathcal{Q}}^{(\bar{\mathbf{2}})}) = \|\langle \hat{\mathbf{Q}}^{(1)}_{(2)}, \ldots, \hat{\mathbf{Q}}^{(N)}_{(2)} \rangle\|,$$

explain the contribution of each element to these clusters and the one with the larger intersection criterion measure, IC_2, can be used to $\mathbf{T}^{(n)}$.

Note that the norm of the sub-factors of the overlapping region excludes any knowledge about how the groups fit with the rest of the tensors. Alternatively, we can account for the strengths of the groups in the whole tensor by also considering the core tensor

$$IC_3(\hat{\mathcal{P}}^{(\bar{\mathbf{2}})}) = \|\langle \lambda_p, \hat{\mathbf{P}}^{(1)}_{(2)}, \ldots, \hat{\mathbf{P}}^{(N)}_{(2)} \rangle\|, \quad IC_3(\hat{\mathcal{Q}}^{(\bar{\mathbf{2}})}) = \|\langle \lambda_q, \hat{\mathbf{Q}}^{(1)}_{(2)}, \ldots, \hat{\mathbf{Q}}^{(N)}_{(2)} \rangle\|,$$

and select the tensor which leads to the larger intersection criterion, IC_3, measure. Here, λ_p and λ_q are core vectors of $\hat{\mathcal{P}}^{(\bar{\mathbf{2}})}$ and $\hat{\mathcal{Q}}^{(\bar{\mathbf{2}})}$, respectively.

Note that for IC_2 and IC_3, the columns of $\hat{\mathbf{P}}^{(n)}_{(2)}$ and $\hat{\mathbf{Q}}^{(n)}_{(2)}$ are normalized to length one with the weights absorbed into the vector λ_p and λ_q, respectively.

Union-Based Selection Criteria. The aforementioned intersection-based selection criteria have a potential weakness: as we see later in Section 5, the selection measures based on intersection fit and norm work well when the two input tensors are balanced in size. If the two tensors are unbalanced in size (i.e. one of the tensors is much larger than the other) the non-overlapping region of the larger tensor is likely to have a large impact on the final accuracy and the intersection-based selection criteria which primarily focus on the overlapping region of the tensors may fail to capture this. To address this limit of intersection-based selection criteria, we also consider *union-based* selection criteria that take into account both non-overlapping and overlapping parts of the tensors.

Firstly, we consider the fit of the union of the decomposed tensors to the union of the two original tensors

$$UC_1(\langle \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \rangle) = 1 - \frac{\|(\mathcal{P} \cup \mathcal{Q}) - \langle \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \rangle\|}{\|\mathcal{P} \cup \mathcal{Q}\|},$$

and we choose between the two alternatives by setting the initial $\mathbf{U}^{(n)}$ to $[\hat{\mathbf{P}}^{(n)T}_{(1)} \hat{\mathbf{P}}^{(n)T}_{(2)} \hat{\mathbf{Q}}^{(n)T}_{(3)}]^T$ and to $[\hat{\mathbf{P}}^{(n)T}_{(1)} \hat{\mathbf{Q}}^{(n)T}_{(2)} \hat{\mathbf{Q}}^{(n)T}_{(3)}]^T$ and observing which one leads to a better fit. UC_1 is the *initial* fit of the union of the decomposed tensors to the union of the two original tensors in the beginning of the update process of $\mathbf{U}^{(n)}_{(k_n)}$ for $k_n = 1, 2, 3$ (see Equation 9). Intuitively, this initial fit can be thought of as a rough indicator of whether the final fit of the union of the decomposed tensors solved by the learning process will be close to the decomposition on the union of two tensors or not.

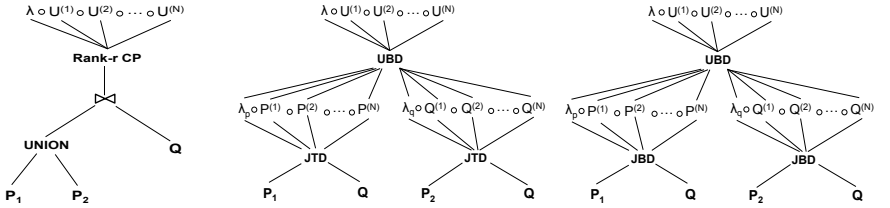
As a second criterion, we consider the density of the input tensors, $\mathcal{P}_{I_1 \times I_2 \times \cdots \times I_N}$ and $\mathcal{Q}_{J_1 \times J_2 \times \cdots \times J_N}$,

$$UC_2(\mathcal{P}) = \frac{|\mathcal{P}|}{\prod_{i=1}^{N} I_i}, \quad UC_2(\mathcal{Q}) = \frac{|\mathcal{Q}|}{\prod_{i=1}^{N} J_i},$$

where $|\mathcal{X}|$ is the number of non-zeros of $\mathcal{X}$. Given this, we set the initial $\mathbf{U}^{(n)}$,

$$\mathbf{U}^{(n)} = [\hat{\mathbf{P}}^{(n)T}_{(1)} \hat{\mathbf{P}}^{(n)T}_{(2)} \hat{\mathbf{Q}}^{(n)T}_{(3)}]^T, \text{ if } \mathcal{P} \text{ has a larger density, or}$$

$$\mathbf{U}^{(n)} = [\hat{\mathbf{P}}^{(n)T}_{(1)} \hat{\mathbf{Q}}^{(n)T}_{(2)} \hat{\mathbf{Q}}^{(n)T}_{(3)}]^T, \text{ if } \mathcal{Q} \text{ has a larger density.}$$



(a) Union, join, decompose (b) Join-then-decompose (JTD) and union-by-decomp. (UBD) (c) Join-by-decomposion (JBD) and union-by-decomp. (UBD)

Fig. 8. Three alternative query plans for implementing a complex query plan with union, join, and decompose operations

Intuitively, the overlapping part will be more tightly connected with the non-overlapping part in the input tensor with the larger density – simply because there are less chances that an entry will be seen only in the overlapping part. Thus, given the choice between using the decompositions (for the overlapping part) of the input tensor with the larger density and of the tensor with the smaller density, the former is likely to lead to lesser errors.

4 Parallelization, Materialization, and Further Optimizations

The proposed union-by-decomposition (UBD) scheme leads to various optimization opportunities. First of all, assuming the availability of multiple computation units, the individual data sources can be decomposed in parallel. Moreover, each individual decomposition of the sub-tensors can also be obtained in parallel, leading to highly parallelizable execution plans. Secondly, as we see in Section 5, in situations where the same data source is integrated (unioned) with different data sources over time, we can decompose this data source once and materialize the decomposition for later reuse within a UBD process, thereby avoiding significant amount of runtime work.

In addition, the proposed union-by-decomposition (UBD) operator is compatible with other novel (decomposition push-down based) operators that are part of TensorDB, including the *join-by-decomposition* (JBD) operator, discussed in Section 1.3, and can be used as part of a general optimization framework. Figure 8 provides an example: in Figure 8(b) first the join is pushed down over union and then the decomposition is pushed down over union, whereas in Figure 8(c) the decomposition is pushed down also over the join operator leading to (as we see in Section 5) a highly efficient query plan.

5 Experimental Evaluation

In this section, we present experimental results assessing the efficiency and effectiveness of the proposed union-by-decomposition (UBD) scheme and the selection criteria.

5.1 Experimental Setup

For these experiments, we used real data tensors (Table 1): (a) MovieLens 1M data set [2] with a 3-mode tensor `(user, movie, rating)` and (b) a 4-mode tensor

Table 1. Tensor data sets

Data set	Attributes	Size	Density (%)
3-mode MovieLens 1M	(user, movie, rating)	$6000 \times 3400 \times 5$	0.8451
3-mode book rating	(user, book, rating)	$105283 \times 340556 \times 11$	0.0003
4-mode Epinions	(user, product, category, rating)	$22111 \times 296000 \times 26 \times 5$	0.000007
4-mode MovieLens 1M	(user, movie, genre, rating)	$6000 \times 3400 \times 18 \times 5$	0.0994

(user movie, genre, rating), (c) a book rating data set [27] with a 3-mode tensor (user, book, rating), and (d) Epinions data set [23] with a 4-mode tensor (user, product, category, rating). From each data tensor, we created pairs of sub-tensors (chosen randomly) with different degrees of intersection (10%, 20%, 40%, 60%). The target rank that we consider for the CP decomposition is 10. The default selection measure is the density-based selection measure, UC_2.

For evaluation, we consider both *execution time* and *degree of fit* defined as

$$\text{fit}(\mathcal{X}, \mathcal{P} \hat{\cup} \mathcal{Q}) = 1 - \frac{\|\mathcal{X} - (\mathcal{P} \hat{\cup} \mathcal{Q})\|}{\|\mathcal{X}\|}, \tag{13}$$

where $\mathcal{X}$ is the union of $\mathcal{P}$ and $\mathcal{Q}$ and $\mathcal{P} \hat{\cup} \mathcal{Q}$ is the tensor obtained by re-composing the decomposition of $\mathcal{P} \cup \mathcal{Q}$ in the considered scheme. Comparing the fit with respect to $\mathcal{X}$ enables us not only to measure how well $\mathcal{P} \cup \mathcal{Q}$ approximates the entries in $\mathcal{P} \cup \mathcal{Q}$, but also whether $\mathcal{P} \hat{\cup} \mathcal{Q}$ includes any spurious entries that are not originally in $\mathcal{P} \cup \mathcal{Q}$.

We ran all the experiments on a machine with Intel Core i5-2400 CPU @ 3.10GHz $\times 4$ with 7.7 GB RAM. We used MATLAB Version 7.13.0.564 (R2011b) 64-bit for the general implementation and MATLAB Parallel Computing Toolbox for the parallel implementations. We used the MATLAB Tensor Toolbox [3] to represent relational tensors as sparse tensors.

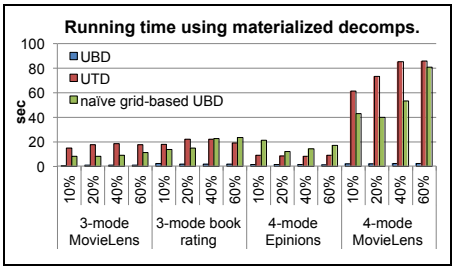
5.2 Results #1: UBD vs. UTD (with and without Materialization)

We first compare the proposed UBD against the more conventional UTD scheme. As a second competitor, we also consider the naive grid-based UBD discussed in Section 3.

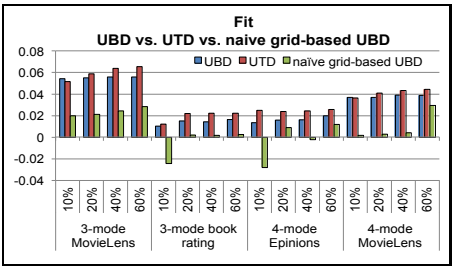
Firstly, as we see in Figure 9(a), when there are opportunities for reusing existing materialized decompositions of the input tensors, as expected, UBD is much faster than the UTD as well as the naive grid-based UBD.

Secondly, in Figure 9(c), we consider the case where there are no opportunities for reusing existing decompositions. As we see in this figure, as expected, when the input tensors have to be decomposed as part of the UBD process, whether UBD outperfoms UTD depends on the characteristics of the input tensors: in particular, as expected, UBD is faster than UTD when (a) *the degree of intersection is low* ($\leq 20\%$) and (b) *the input tensors are not extremely sparse*: if these conditions are not satisfied, the size of the union result is close to the sizes of input tensors and, if the result is also sparse, there is no gain in pushing down the decompositions.

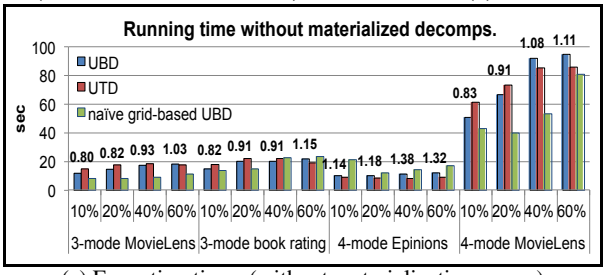
Note that, when materialized decompositions of the input tensors do not exist, grid-based UBD can out-pace the proposed UBD and UTD in many configurations. However, as we see in Figure 9(b), this comes at the cost of a significant drop in accuracy: the proposed UBD scheme achieves fits close to the fit of UTD, whereas the accuracy of the grid-based UBD is much lower. Note also that the accuracy of UBD is especially good in data sets that are not *extremely sparse*.



(a) Execution times (with materialization re-use)

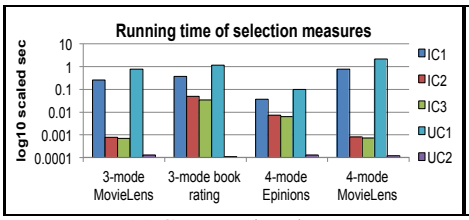


(b) Accuracies

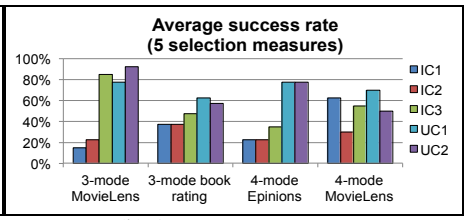


(c) Execution times (without materialization re-use)
Running time ratio (UBD/UTD) is shown on each bar
(the smaller the ratio, the better is the performance of UBD)

Fig. 9. UBD vs. UTD vs. naive grid-based UBD on pairs of tensors with different intersection sizes (10%, 20%, 40%, 60%)



(a) Computation times



(b) Average success rate

Fig. 10. Efficiency and accuracy of the different selection measures in average of different intersection sizes (10%, 20%, 40%, and 60%)

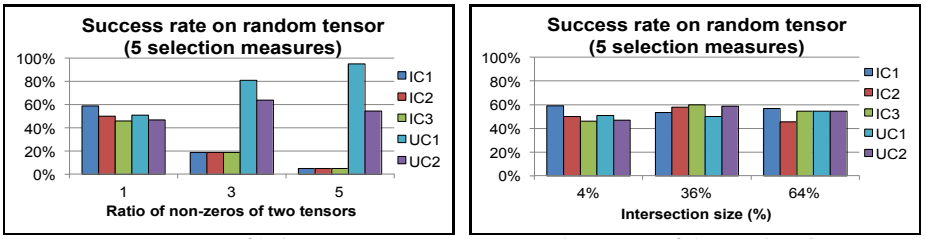
5.3 Results #2: Evaluation of the Alternative Selection Measures

In Section 3.2, we considered various approaches (IC_1, IC_2, IC_3, UC_1, and UC_2) for choosing the sub-factors for the overlapping parts of the input tensors. Figure 10(a) shows that fit-based measures (intersection fit, IC_1 and union fit, UC_1) are more expensive than norm-based measures (IC_2, IC_3). The *density*-based approach (UC_2) has an almost 0 execution cost. Note that, when we compare the computation times of these selection measures to the execution times of the UBD operators (Figure 9), we see that even the most expensive selection strategy is, in practice, affordable. Therefore, the major criterion for selecting among these measures should be accuracy.

For measuring the accuracy of different selection measures, we considered the percentage of the cases where each selection measure returned the best alternative. As

Table 2. Average fit of the different selection measures (The highest fits for each data set are highlighted in bold)

Data set	IC_1	IC_2	IC_3	UC_1	UC_2
3-mode MovieLens 1M	0.0538	0.0539	0.0551	0.0551	**0.0553**
3-mode book rating	0.0127	0.0127	0.0134	**0.0141**	0.0138
4-mode Epinions	0.0133	0.0133	0.0144	**0.0164**	**0.0164**
4-mode MovieLens 1M	**0.0380**	0.0376	0.0378	0.0377	**0.0380**



(a) Impact of balance (b) Impact of the overlap size

Fig. 11. Success rate in predicting the best fit of UBD using the 5 selection measures compared among different (a) ratios of non-zeros of two tensors and (b) intersection sizes

shown in Figure 10(b), the union-based fit (UC_1) measure works best overall. The density measure (UC_2) also works well. The figure also shows that the intersection-based measures (IC_1, IC_2, IC_3) are not good indicators, even behave negatively in some cases: among them the IC_3 works the best since it also accounts for the non-overlapping regions through the cluster strength indicated by the core. Table 2 further studies the average degree of fits returned by the different strategies. The table confirms that the average fits obtained by the union-based selection measures are overall better than the intersection-based selection measures. While the numbers vary, the degrees of fit based on the union-based selection measures are up to 20% better than IC_1 and IC_2.

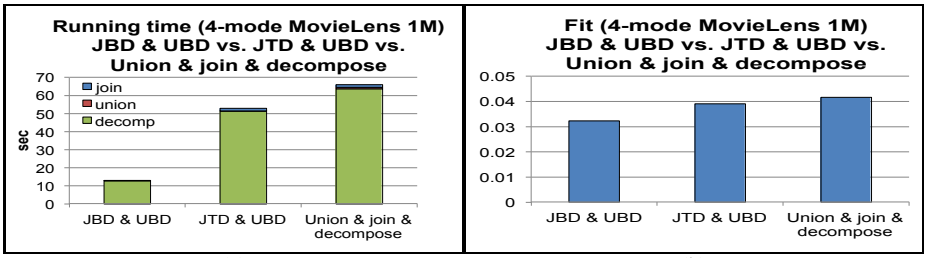
To further study the impacts of various parameters on the selection accuracy, we also created random tensors with different configurations, varying the balance (ratio of densities) of the input tensors and intersection sizes. For each experiment, we created 10 different random tensors of size $5000 \times 5000 \times 10$ and measured the percentage cases in which each measure selected the better fitting tensor. As the default configuration, we set the ratio of non-zeros to 1 (most balanced), intersection size to 4%, and the density of the union tensor to 0.01%.

In Figure 11(a), we first study the impact of balance. Here, the configuration with $ratio = 1$ corresponds to the most balanced configuration. As we expected, when the tensors are balanced, all measures work similarly (with a slight edge to the intersection-based measures); however, as the imbalance among tensors increases, intersection-based measures get worse, while the union based measures, especially UC_1, improve.

Unlike balance, the size of the intersection has no significant impact on the selection accuracy (Figure 11(b)), indicating that all measures are robust in this respect.

5.4 Results #3: Impact of Composition of UBD with Other Operators

As we discussed in Section 4, the proposed union-by-decomposition (UBD) operator is compatible with other operators that are part of TensorDB and can be used as part



(a) (b)

Fig. 12. (a) Running times and (b) fits of three alternative query plans "JBD and UBD" vs. "JTD and UBD" vs. "union, join, and decompose" (see Figure 8) on 4-mode MovieLens 1M

of a general optimization framework. In Figure 12 for a sample data, we study the alternative query plans considered in Figure 8. As expected, the figure shows that pushing decompositions down the join and union operations (i.e., using UBD, proposed in this paper, and/or JBD, proposed in [11]) provides a much faster execution times than the union operation and join operation followed by a final CP decomposition step. As shown in Figure 12(a), among these three alternative query plans, the query plan using JBD and UBD is the fastest (faster than $5\times$ of the union, join, and decompose strategy) but comes with $\sim 20\%$ drop in accuracy (Figure 12(b)). On the other hand, using UBD proposed in this paper along with the conventional join-then-decompose (JTD) strategy instead of JBD reduces the execution time relative to "union, join, and decompose" by $\sim 20\%$ (Figure 12(a)), with a negligible impact on accuracy (Figure 12(b)).

6 Conclusion

TensorDB, which extends array databases with a tensor-relational model (TRM), supports both relational algebraic operations (for data manipulation and integration) and tensor algebraic operations (for data analysis) for the complete life cycle of data that involves consecutive steps of integration, manipulation, and analysis. In TensorDB, we focused on data processing workflows involving both tensor decomposition and data integration (union) operations and proposed a novel scheme for pushing down the tensor decompositions over the union operations to reduce the overall data processing times and to promote reuse of materialized tensor decomposition results. Experimental results confirmed the efficiency and effectiveness of the proposed decomposition push-down strategy and the corresponding union-by-decomposition (UBD) operator.

References

1. Hash tables and associative arrays. In: Algorithms and Data Structures, pp. 81–98. Springer, Heidelberg (2008)
2. Movielens dataset from grouplens research group, `http://www.grouplens.org`
3. Bader, B.W., Kolda, T.G.: Matlab tensor toolbox version 2.2 (January 2007), `http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/`
4. Baumann, P., et al.: The multidimensional database system rasdaman. In: SIGMOD (1998)

5. Brown, P.G.: Overview of scidb: large scale array storage, processing and analysis. In: SIGMOD, pp. 963–968 (2010)
6. Carroll, J., Chang, J.-J.: Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. Psychometrika (1970)
7. Cohen, J., et al.: Mad skills: new analysis practices for big data. In: VLDB (2009)
8. Dobos, L., et al.: Array requirements for scientific applications and an implementation for microsoft sql server. In: AD, pp. 13–19 (2011)
9. Harshman, R.A.: Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. In: UCLA Working Papers in Phonetics (1970)
10. Kang, U., et al.: Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries. In: KDD, pp. 316–324 (2012)
11. Kim, M., Candan, K.S.: Approximate tensor decomposition within a tensor-relational algebraic framework. In: CIKM, pp. 1737–1742 (2011)
12. Kim, M., Candan, K.S.: Decomposition-by-normalization (dbn): Leveraging approximate functional dependencies for efficient tensor decomposition. In: CIKM, pp. 355–364 (2012)
13. Kolda, T., Bader, B.: Tensor decompositions and applications. In *SIAM Review* 51(3), 455–500 (2009)
14. Kolda, T., Sun, J.: Scalable tensor decompositions for multi-aspect data mining. In: ICDM, pp. 363–372 (December 2008)
15. Kolda, T.G., et al.: Higher-order web link analysis using multilinear algebra. In: ICDM (2005)
16. Low, Y., et al.: Distributed graphlab: a framework for machine learning and data mining in the cloud. VLDB 5(8), 716–727 (2012)
17. Papalexakis, E.E., Faloutsos, C., Sidiropoulos, N.D.: Parcube: Sparse parallelizable tensor decompositions. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012, Part I. LNCS, vol. 7523, pp. 521–536. Springer, Heidelberg (2012)
18. Phan, A.H., Cichocki, A.: Parafac algorithms for large-scale problems. Neurocomputing 74(11), 1970–1984 (2011)
19. Sanchez, E., Kowalski, B.R.: Generalized rank annihilation factor analysis. Analytical Chemistry 58(2), 496–499 (1986)
20. Sanchez, E., Kowalski, B.R.: Tensorial resolution: A direct trilinear decomposition. Journal of Chemometrics 4(1), 29–45 (1990)
21. Sorber, L., et al.: Optimization-based algorithms for tensor decompositions: canonical polyadic decomposition, decomposition in rank-(L_r,L_r,1) terms, and a new generalization. SIAM Journal on Optimization 23(2), 695–720 (2013)
22. Stonebraker, M., et al.: C-store: a column-oriented dbms. In: VLDB, pp. 553–564 (2005)
23. Tang, J., et al.: eTrust: Understanding trust evolution in an online world. In: KDD (2012)
24. Tsourakakis, C.E.: Mach: Fast randomized tensor decompositions. In: SDM, pp. 689–700 (2010)
25. Tucker, L.: Some mathematical notes on three-mode factor analysis. Psychometrika 31(3), 279–311 (1966)
26. van Ballegooij, A.R., Cornacchia, R., de Vries, A.P., Kersten, M.L.: Distribution rules for array database queries. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588, pp. 55–64. Springer, Heidelberg (2005)
27. Ziegler, C.-N., et al.: Improving recommendation lists through topic diversification. In: WWW (2005)