Localized Matrix Factorization for Recommendation based on Matrix Block Diagonal Forms

Yongfeng Zhang Min Zhang Yiqun Liu Shaoping Ma Shi Feng
State Key Laboratory of Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science & Technology, Tsinghua University, Beijing, 100084, China
zhangyf07@gmail.com {z-m,yiqunliu,msp}@tsinghua.edu.cn fredfsh@gmail.com

ABSTRACT

Matrix factorization on user-item rating matrices has achieved significant success in collaborative filtering based recommendation tasks. However, it also encounters the problems of data sparsity and scalability when applied in real-world recommender systems. In this paper, we present the *Localized Matrix Factorization (LMF)* framework, which attempts to meet the challenges of sparsity and scalability by factorizing *Block Diagonal Form (BDF)* matrices. In the LMF framework, a large sparse matrix is first transformed into *Recursive Bordered Block Diagonal Form (RBBDF)*, which is an intuitionally interpretable structure for user-item rating matrices. Smaller and denser submatrices are then extracted from this RBBDF matrix to construct a BDF matrix for more effective collaborative prediction. We show formally that the LMF framework is suitable for matrix factorization and that any *decomposable* matrix factorization algorithm can be integrated into this framework. It has the potential to improve prediction accuracy by factorizing smaller and denser submatrices independently, which is also suitable for parallelization and contributes to system scalability at the same time. Experimental results based on a number of real-world public-access benchmarks show the effectiveness and efficiency of the proposed LMF framework.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Filtering; H.3.5 [**Online Information Services**]: Web-based services; G.1.6 [**Mathematics of Computing**]: Optimization

Keywords

Matrix Factorization; Collaborative Filtering; Block Diagonal Form; Graph Partitioning

1. INTRODUCTION

Latent factor model has been one of the most powerful approaches for collaborative filtering. Some of the most successful realizations of latent factor models are based on Matrix Factorization (MF) techniques [17]. The fundamental idea of these approaches is that user preferences can be determined by a relatively small number of latent factors. A variety of matrix factorization methods have been proposed and applied to various collaborative filtering tasks successfully, such as Singular Value Decomposition (SVD) [17, 33], Non-negative Matrix Factorization (NMF) [18, 19], Max-Margin Matrix Factorization (MMMF) [34, 23] and Probabilistic Matrix Factorization (PMF) [26, 25].

However, MF approaches have also encountered a number of problems in real-world recommender systems, such as data sparsity, frequent model retraining and system scalability. As the number of ratings given by most users is relatively small compared with the total number of items in a typical system, data sparsity usually decreases prediction accuracy and may even lead to over-fitting problems. In addition, new ratings are usually made by users continuously in real-world recommender systems, leading to the need for refactoring rating matrices periodically, which is time consuming for systems with millions or even billions of ratings, and further restricts the scalability of MF approaches.

In this study, we propose a novel MF framework named *Localized Matrix Factorization (LMF)*, which is general and intrinsically compatible with many widely-adopted MF algorithms. Before problem formalization, we would like to use an intuitional example to briefly introduce the matrix structures used in LMF. Figure 1(a) is a sparse matrix where each row/column/cross represents a user/item/rating. By permuting Row4, Row9 and Column7 to 'borders', the remaining part is partitioned into two 'diagonal blocks', which results in a *Bordered Block Diagonal Form (BBDF)* [4] matrix in Figure 1(b). By 'recursively' permuting the first diagonal block, we obtain a *Recursive Bordered Block Diagonal Form (RBBDF)* matrix in Figure 1(c). BBDF and RBBDF structures are generalizations of *Block Diagonal Form (BDF)* structure which has no 'border'.

RBBDF structure is intuitionally interpretable in collaborative filtering tasks. Consider movie recommendation as an example. Different users may have different preferences on movie genres, which form different communities, corresponding to the diagonal blocks in the BBDF structure. However, there does exist 'super users' whose interests are relatively broad and thus fall into different communities. This type of user is represented by row borders in the BBDF structure. There are also some classical or hot movies widely known and enjoyed by users from different communities, which are 'super items' making up column borders. The structure may recurse at multiple finer-grained levels in a community, resulting in the generation of RBBDF structures. As different communities may have different rating patterns, it would be better to factorize them independently.

The LMF framework transforms a sparse matrix into RBBDF structure and further extracts denser submatrices to construct a BDF matrix. Factorization of the BDF matrix is used to approximate the original sparse matrix. The framework brings several attractive benefits to recommender systems: 1) Factorizing extracted dense submatrices instead of the whole sparse matrix improves the prediction accuracy of matrix factorization algorithms. 2) The locality property of LMF makes it possible to refactorize only the recently-updated submatrices rather than the whole matrix. 3) The framework is suitable for parallelization, which further contributes to the scalability of recommender systems.

In summary, the main contributions of this work are:

- The RBBDF structure of rating matrices is investigated, which is intuitionally interpretable in CF tasks.
- A density-based algorithm is designed to transform a sparse matrix into RBBDF structure.
- The LMF framework is proposed and its rationality is shown through theoretical analyses.
- Through a comprehensive experimental study on four benchmark datasets, both the efficiency and effectiveness of the LMF framework is verified.

The remainder of this paper will be organized as follows: Section 2 reviews some related work, and Section 3 presents some preliminaries. In Section 4, the LMF framework is introduced and investigated. Experimental results are shown in Section 5. Some discussions will be made in Section 6, and the work is concluded in Section 7.

2. RELATED WORK

Collaborative Filtering (CF) [35] techniques have been known to have several attractive advantages over other recommendation strategies, such as Content-based Filtering [22] in Personalized Recommender Systems [21]. Early CF algorithms mainly focus on memory-based approaches such as User-based [24] and Item-based [29] methods, which calculate the similarities of users or items to make rating predictions [21]. To gain better prediction accuracies and to overcome the shortcomings of memory-based algorithms, model-based approaches have been investigated extensively, which estimate or learn a model on user-item rating matrices to make rating predictions [35, 21].

Latent Factor Models (LFM) based on Matrix Factorization (MF) [36] techniques have been an important research direction in model-based CF methods. Recently, MF approaches have gained great popularity as they usually outperform traditional methods [35, 12] and have achieved state-of-the-art performance, especially on large-scale recommendation tasks [17]. A variety of MF algorithms have been proposed and investigated in different CF settings, such as Principle Component Analysis (PCA) [1], Singular Value Decomposition (SVD) [16, 17, 33], Non-negative Matrix Factorization (NMF) [18, 19], Max-Margin Matrix Factorization (MMMF) [34, 23], and Probabilistic Matrix Factorization (PMF) [26, 25]. They aim at learning latent factors from a matrix, with which to make rating predictions.

According to the unified view of MF in [32], MF algorithms are optimization problems over given loss functions and regularization terms. Different choices of loss functions and regularization terms lead to different MF methods.

However, MF approaches also suffer from a number of problems in real-world recommender systems, such as data

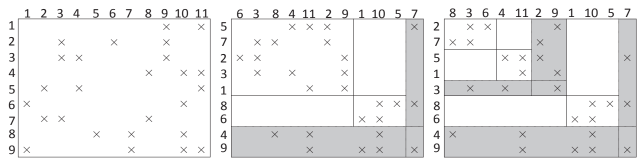

(a) Original matrix (b) BBDF matrix (c) RBBDF matrix

Figure 1: An example of (R)BBDF structure

sparsity, frequent model retraining and system scalability. To overcome the problem of data sparsity, earlier systems rely on imputation to fill in missing ratings and to make the rating matrix dense [28]. However, imputation can be very expensive as it significantly increases the amount of ratings, and inaccurate imputation may distort the data considerably [17]. The problem of frequent model retraining and scalability results from the fact that the total number of users and items is usually very large in practical systems, and new ratings are usually made by users continuously.

Most efforts to improve system scalability focus on matrix clustering techniques [38, 31, 10, 9, 37] or designing incremental and distributed versions of existing MF algorithms [30, 20, 11]. Usually, they can achieve only approximated results compared with factorizing the whole matrix directly, and many of them restrict themselves to one specific MF algorithm. In contrast with these approaches, we demonstrate that the LMF framework on a BDF matrix is theoretically equal to factorizing the whole matrix directly, and that it is compatible with any existing *decomposable* MF algorithm.

Another related research field is graph partitioning, as permuting a sparse matrix into BBDF structure is equivalent to conducting *Graph Partitioning by Vertex Separator (GPVS)* on its corresponding bipartite graph [4]. Graph partitioning is known to be NP-hard [7], but this problem has been investigated extensively, and many efficient and high-quality heuristic-based methods have been proposed [15], such as multilevel methods [14, 6], spectral partitioning [3] and kernel-based methods [2]. It is verified both theoretically and experimentally that multilevel approaches can give both fast execution time and very high quality partitions [27, 4, 14, 6, 15], which guides us to choosing the multilevel graph partitioning approach in this work.

3. PRELIMINARIES

3.1 Matrix Factorization

We take the unified view of MF proposed in [32], which is sufficient to include most of the existing MF algorithms. Let $X \in \mathbb{R}^{m\times n}$ be a sparse matrix, and let $U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{n\times r}$ be its factorization. An MF algorithm $\mathcal{P} = (f, \mathcal{D}_W, \mathcal{C}, \mathcal{R})$ can be defined by the following choices:

1. Prediction link $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$.
2. Optional data weights $W \in \mathbb{R}_{+}^{m\times n}$, which if used must be an argument of the loss function.
3. Loss function $\mathcal{D}_W(X, f(UV^T)) \geq 0$, which is a measure of the error when approximating X with $f(UV^T)$.
4. Hard constraints on factors: $(U, V) \in \mathcal{C}$.
5. Regularization penalty: $\mathcal{R}(U, V) \geq 0$.

For an MF model $X \approx f(UV^T) \triangleq X^*$, we solve:

$$\underset{(U,V)\in\mathcal{C}}{\operatorname{argmin}} \left[\mathcal{D}_W(X, f(UV^T)) + \mathcal{R}(U, V)\right]. \quad (1)$$

The loss $\mathcal{D}(\cdot,\cdot)$ is typically convex in its second argument, and often decomposes into a (weighted) sum over elements of X [32]. For example, the loss function of WSVD [33] is:

$$\mathcal{D}_W(X, f(UV^T)) = \|W \odot (X - UV^T)\|_{Fro}^2 \tag{2}$$

where $\odot$ denotes the element-wise product of matrices.

In this paper, we refer to $X = UV^T$ as an Accurate Matrix Factorization of X, and refer to $X \approx f(UV^T) \triangleq X^*$ as an Approximate Matrix Factorization of X.

3.2 BDF, BBDF and RBBDF

We consider permuting the rows and/or columns of a sparse matrix to reform its structure. X is a Block Diagonal Form (BDF) matrix if:

$$X = \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_k \end{bmatrix} \triangleq \operatorname{diag}(D_i) \tag{3}$$

It is not always the case that a sparse matrix can be permuted into BDF, but usually it can be permuted into Bordered Block Diagonal Form (BBDF) [4] shown in (4). Each $D_i (1 \le i \le k)$ is a 'diagonal block'. $R_b \triangleq [R_1 \cdots R_k B]$ and $C_b \triangleq [C_1^T \cdots C_k^T B^T]^T$ are row and column 'borders':

$$X = \left[\begin{array}{c:c} D & C \\ \hdashline R & B \end{array}\right] = \left[\begin{array}{ccc:c} D_1 & & & C_1 \\ & \ddots & & \vdots \\ & & D_k & C_k \\ \hdashline R_1 & \cdots & R_k & B \end{array}\right] \tag{4}$$

Any of the k diagonal blocks D_i in (4) may be permuted into BBDF structure recursively, resulting in Recursive Bordered Block Diagonal Form (RBBDF). To avoid notational clutter, we present the following example, where $\mathcal{I}_*$ and $\mathcal{J}_*$ denote the row and column index sets:

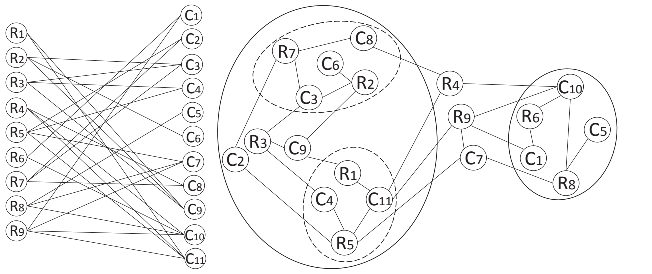
$$X = \begin{array}{c} \begin{array}{ccc} \mathcal{J}_1 & \mathcal{J}_2 & \mathcal{J}_B \end{array} \\ \left[\begin{array}{cc:c} D_1 & & C_1 \\ & D_2 & C_2 \\ \hdashline R_1 & R_2 & B \end{array}\right] \end{array} \begin{array}{c} \mathcal{I}_1 \\ \mathcal{I}_2 \\ \mathcal{I}_B \end{array} = \begin{array}{c} \begin{array}{ccccc} \mathcal{J}_{11} & \mathcal{J}_{12} & \mathcal{J}_{B_1} & \mathcal{J}_2 & \mathcal{J}_B \end{array} \\ \left[\begin{array}{cccc:c} D_{11} & & C_{11} & & C_1^1 \\ & D_{12} & C_{12} & & C_1^2 \\ R_{11} & R_{12} & B_1 & & C_1^3 \\ & & & D_2 & C_2 \\ \hdashline R_1^1 & R_1^2 & R_1^3 & R_2 & B \end{array}\right] \end{array} \begin{array}{c} \mathcal{I}_{11} \\ \mathcal{I}_{12} \\ \mathcal{I}_{B_1} \\ \mathcal{I}_2 \\ \mathcal{I}_B \end{array} \tag{5}$$

In (5), D_1 is permuted into BBDF recursively. Note that permuting rows and columns related to D_1 affects R_1 and C_1, but it only changes the order of the non-zeros therein. Diagonal blocks D_{11}, D_{12} and D_2 may be further permuted depending on certain stopping rules. This will be introduced in our algorithm for constructing RBBDF structures.

3.3 Graph Partitioning by Vertex Separator

A sparse rating matrix can be equally represented by a bipartite graph. Consider Figure 1(a) and Figure 2(a) as examples. Each row or column of the matrix corresponds to an R-node or a C-node in the bipartite graph.

GPVS partitions a graph into disconnected components by removing a set of vertices (vertex separator) and their incident edges. As demonstrated by [4], permuting a sparse matrix into BBDF structure is equivalent to conducting GPVS on its bipartite graph. For example, removing nodes R_4, R_9 and C_7 in Figure 2(b) corresponds to permuting Row4, Row9 and Column7 to borders in Figure 1(b), and the two resulting disconnected components correspond to two diagonal blocks. GPVS is conducted recursively on the left component, and the RBBDF matrix in Figure 1(c) is constructed.



(a) Bipartite graph (b) GPVS on the bipartite graph

Figure 2: The bipartite graph for a sparse matrix and graph partitioning by vertex separator on it.

4. LMF

4.1 Definitions and Theorems

We present some definitions, propositions and theorems in this section, which will be the basis of the LMF framework.

4.1.1 Accurate Matrix Factorization

A matrix in BDF or (R)BBDF structure has some important properties in terms of accurate matrix factorization.

PROPOSITION 1. *For a BDF matrix $X = \operatorname{diag}(D_i)$ in (3), if we have $D_i = U_i V_i^T$ for each diagonal block D_i; we then have $X = \operatorname{diag}(U_i) \cdot \operatorname{diag}(V_i^T)$ as a factorization for X.* □

This proposition shows the independence of diagonal blocks from each other in a BDF matrix in terms of accurate matrix factorization. As stated above, it is not guaranteed that a sparse matrix can be permuted into BDF structure. However, we have the following proposition for a BBDF matrix.

PROPOSITION 2. *For a BBDF matrix X in (4), let:*

$$\tilde{X}_i \triangleq \begin{bmatrix} D_i & C_i \\ R_i & B \end{bmatrix} = U_i V_i^T = \begin{bmatrix} U_{i1} \\ U_{i2} \end{bmatrix} \begin{bmatrix} V_{i1}^T & V_{i2}^T \end{bmatrix} \tag{7}$$

be a factorization of $\tilde{X}_i$; thus, we have:

$$D_i = U_{i1}V_{i1}^T \quad R_i = U_{i2}V_{i1}^T \quad C_i = U_{i1}V_{i2}^T \quad B = U_{i2}V_{i2}^T$$

and let:

$$U = \begin{bmatrix} U_{11} & & & \\ & U_{21} & & \\ & & \ddots & \\ & & & U_{k1} \\ U_{12} & U_{22} & \cdots & U_{k2} \end{bmatrix} \quad V = \begin{bmatrix} V_{11} & & & \\ & V_{21} & & \\ & & \ddots & \\ & & & V_{k1} \\ V_{12} & V_{22} & \cdots & V_{k2} \end{bmatrix}$$

We then have:

$$UV^T = \begin{bmatrix} U_{11}V_{11}^T & & & U_{11}V_{12}^T \\ & U_{21}V_{21}^T & & U_{21}V_{22}^T \\ & & \ddots & \vdots \\ & & U_{k1}V_{k1}^T & U_{k1}V_{k2}^T \\ U_{12}V_{11}^T & U_{22}V_{21}^T \cdots & U_{k2}V_{k1}^T & \sum_{i=1}^{k} U_{i2}V_{i2}^T \end{bmatrix} = \begin{bmatrix} D_1 & & C_1 \\ & \ddots & \vdots \\ & D_k & C_k \\ R_1 \cdots & R_k & kB \end{bmatrix}$$

The only difference between UV^T and X in (4) is that the border cross B in matrix X is multiplied by the number of diagonal blocks k. □

Proposition 2 is in fact factorizing a block diagonal form matrix $\tilde{X} = \operatorname{diag}(\tilde{X}_i) = \operatorname{diag}\left(\left[\begin{smallmatrix} D_i & C_i \\ R_i & B \end{smallmatrix}\right]\right) (1 \le i \le k)$, as denoted in (7). According to Proposition 1, if $\tilde{X}_i = U_i V_i^T$, then we have $\tilde{X} = \operatorname{diag}(U_i) \cdot \operatorname{diag}(V_i^T)$. By averaging the

$$X = \begin{bmatrix} X_1 & & & \\ & X_2 & & \\ & & \ddots & \\ & & & X_k \end{bmatrix} \approx f(UV^T) = f\left(\begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_k \end{bmatrix} \begin{bmatrix} V_1^T & V_2^T & \cdots & V_k^T \end{bmatrix}\right) = f\left(\begin{bmatrix} U_1V_1^T & U_1V_2^T & \cdots & U_1V_k^T \\ U_2V_1^T & U_2V_2^T & \cdots & U_2V_k^T \\ \vdots & \vdots & \ddots & \vdots \\ U_kV_1^T & U_kV_2^T & \cdots & U_kV_k^T \end{bmatrix}\right) \quad (6)$$

duplicated submatrices, for example, submatrix B in (4), the original matrix X is reconstructed with the factorizations of $\tilde{X}_i = U_iV_i^T$, where $1 \leq i \leq k$.

This property can be generalized to an RBBDF matrix. To avoid notational clutter, the example in (5) is again used here. To transform X into BDF, diagonal block D_1 is transformed into BDF first, resulting in an intermediate matrix:

$$\tilde{X}_{int.} = \begin{array}{cccccc:cc} \mathcal{J}_{11} & \mathcal{J}_{B_1} & \mathcal{J}_{12} & \mathcal{J}_{B_1} & \mathcal{J}_2 & & \mathcal{J}_B & \\ D_{11} & C_{11} & & & & & C_1^1 & \mathcal{I}_{11} \\ R_{11} & B_1 & & & & & C_1^3 & \mathcal{I}_{B_1} \\ & & D_{12} & C_{12} & & & C_1^2 & \mathcal{I}_{12} \\ & & R_{12} & B_1 & & & C_1^3 & \mathcal{I}_{B_1} \\ & & & & D_2 & & C_2 & \mathcal{I}_2 \\ \hdashline R_1^1 & R_1^3 & R_1^2 & R_1^3 & R_2 & & B & \mathcal{I}_B \end{array} \quad (8)$$

This is a BBDF matrix with 3 diagonal blocks. By conducting the same procedure on $\tilde{X}_{int.}$, it is transformed into a BDF matrix ($\tilde{X}_{ij} = \mathbf{0}$ for $i \neq j$):

$$\tilde{X} = \begin{array}{cccccccc|c} \mathcal{J}_{11} & \mathcal{J}_{B_1} & \mathcal{J}_B & \mathcal{J}_{12} & \mathcal{J}_{B_1} & \mathcal{J}_B & \mathcal{J}_2 & \mathcal{J}_B & \\ D_{11} & C_{11} & C_1^1 & & & & & & \mathcal{I}_{11} \\ R_{11} & B_1 & C_1^3 & & \tilde{X}_{12} & & \tilde{X}_{13} & & \mathcal{I}_{B_1} \\ R_1^1 & R_1^3 & B & & & & & & \mathcal{I}_B \\ & & & D_{12} & C_{12} & C_1^2 & & & \mathcal{I}_{12} \\ & \tilde{X}_{21} & & R_{12} & B_1 & C_1^3 & \tilde{X}_{23} & & \mathcal{I}_{B_1} \\ & & & R_1^2 & R_1^3 & B & & & \mathcal{I}_B \\ & \tilde{X}_{31} & & & \tilde{X}_{32} & & D_2 & C_2 & \mathcal{I}_2 \\ & & & & & & R_2 & B & \mathcal{I}_B \end{array} \quad (9)$$

$$\triangleq \mathrm{diag}(\tilde{X}_1, \tilde{X}_2, \tilde{X}_3)$$

Similarly, $\tilde{X}_1, \tilde{X}_2$ and $\tilde{X}_3$ can be factorized independently, and duplicated submatrices are averaged to reconstruct the original matrix X in (5).

In fact, (9) can be derived from (5) directly without constructing intermediate matrices. Each diagonal block $\tilde{X}_i$ in $\tilde{X}$ corresponds to a diagonal block D_i in X. By piecing together D_i with the parts of borders on the right side, down side and right bottom side, $\tilde{X}_i$ can be constructed directly. Additionally, permuting any D_i into BBDF structure recursively in (5) would not affect other block diagonals in $\tilde{X}$.

4.1.2 Approximate Matrix Factorization

In practical applications, approximate matrix factorization algorithms formalized by (1) are used. Consider the BDF matrix $X = \mathrm{diag}(X_i)(1 \leq i \leq k)$ in terms of the approximate matrix factorization denoted by (6). For notational clarity, the superscript 'tilde' of $\tilde{X}$ is removed in this section. Decomposable properties will be investigated in different aspects in detail in this section, as they are of core importance with respect to what types of matrix factorization algorithms the framework can handle.

In the following definitions and theorems, $X_{ij} = \begin{cases} X_i & (i{=}j) \\ \mathbf{0} & (i{\neq}j) \end{cases}$ is used to denote submatrices of X in (6), and W_{ij} denotes the weight matrix of X_{ij}. $f(UV^T)_{ij}$ denotes the submatrix in $f(UV^T)$ that approximates X_{ij}, namely, $X_{ij} \approx f(UV^T)_{ij}$. Specifically, $f(UV^T)_i$ is used for $f(UV^T)_{ii}$, and W_i is used for W_{ii} when $i = j$.

Definition 1. ***Decomposable prediction link.*** *A prediction link $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ is decomposable if:*

$$f(UV^T)_{ij} = f(U_iV_j^T) \quad (1 \leq i, j \leq k) \qquad (10)$$

A large class of MF algorithms use element-wise prediction links for each pair of element in $Y = f(X)$, namely, $y_{ij} = f(x_{ij})$. For example, the prediction link is $f(x) = x$ in SVD, and in NMF, $f(x) = \log(x)$. Element-wise link functions lead to the decomposable property above naturally.

Definition 2. ***Decomposable loss function.*** *A loss function $\mathcal{D}_W(X, f(UV^T))$ is decomposable if:*

$$\mathcal{D}_W(X, f(UV^T)) = \sum_{i=1}^{k} \mathcal{D}_{W_i}(X_i, f(UV^T)_i) \qquad (11)$$

This property can be viewed in two aspects here.

First, a substantial number of MF algorithms restrict $\mathcal{D}$ to be expressed as the sum of losses over elements, *e.g.*, SVD[17, 33], NMF[19], MMMF[34] and PMF[26]. Various decomposable regular Bregman divergences are the most commonly used loss functions that satisfy this property [5]. The per-element effect gives the following property:

$$\mathcal{D}_W(X, f(UV^T)) = \sum_{i,j} \mathcal{D}_{W_{ij}}(X_{ij}, f(UV^T)_{ij}) \qquad (11.1)$$

Second, rating matrices in CF tasks are usually incomplete and very sparse in practical recommender systems. A 'zero' means only that the user did not make a rating on the corresponding item, rather than rating it zero. As a result, many MF algorithms optimize loss functions on observed ratings. Specifically, $W_{ij} = \mathbf{0}$ $(i \neq j)$ in a BDF matrix, and:

$$\mathcal{D}_{W_{ij}}(X_{ij}, f(UV^T)_{ij}) = 0 \quad (i \neq j) \qquad (11.2)$$

(11.1) and (11.2) gives the decomposable property of loss functions in Definition 2.

Definition 3. ***Decomposable hard constraint.*** *A hard constraint $\mathcal{C}$ is decomposable if:*

$$(U, V) \in \mathcal{C} \ \text{ iff. } \ (U_i, V_i) \in \mathcal{C} \quad (1 \leq i \leq k) \qquad (12)$$

Many MF algorithms do not apply hard constraints to target factorizations, but there are MF methods that require (U, V) to meet some special requirements.

Some commonly used hard constraints are non-negativity (the elements of U,V are non-negative), orthogonality (the columns of U,V are orthogonal), stochasticity (each row of U,V sums to one), sparsity (the row vectors of U,V meet a desired sparseness constraint) and cardinality (the number of non-zeros in each row of U,V satisfies a given constraint).

In this sense, non-negativity, stochasticity, sparsity and cardinality constraints are decomposable hard constraints. For example, each row of (U, V) sums to one if and only if the same property holds for any (U_i, V_i) $(1 \leq i \leq k)$. However, orthogonality is not decomposable: the orthogonality in (U, V) does not ensure the orthogonality in each (U_i, V_i). Our primary focus is on decomposable hard constraints.

Definition 4. ***Decomposable regularization penalty.*** *A regularization penalty $\mathcal{R}(U,V)$ is decomposable if:*

$$\mathcal{R}(U,V) = \sum_{i=1}^{k} \mathcal{R}(U_i, V_i) \tag{13}$$

The most commonly used regularization penalty is the ℓ_p-*norm* regularizer, which is decomposable:

$$\begin{aligned} & \mathcal{R}(U,V) = \lambda_U \|U\|_p^p + \lambda_V \|V\|_p^p \\ = & \sum_{i=1}^{k} \left(\lambda_U \|U_i\|_p^p + \lambda_V \|V_i\|_p^p\right) = \sum_{i=1}^{k} \mathcal{R}(U_i, V_i) \end{aligned}$$

The Frobenius norm is ℓ_p-*norm* where $p = 2$. The basic MMMF algorithm takes the trace-norm $\|X\|_\Sigma$ (the sum of singular values of X) [34], which is unfortunately not a decomposable regularizer. However, a fast MMMF algorithm based on the equivalence $\|X\|_\Sigma = \min_{X=UV^T} \frac{1}{2}(\|U\|_F^2 + \|V\|_F^2)$ is proposed in [23], which also takes ℓ_p-*norm* regularizers.

Definition 5. ***Decomposable matrix factorization.*** *A matrix factoirzation algorithm $\mathcal{P} = (f, \mathcal{D}_W, \mathcal{C}, \mathcal{R})$ is decomposable if $f, \mathcal{D}_W, \mathcal{C}, \mathcal{R}$ are decomposable. Namely, properties (10)$\sim$(13) hold. $(U,V) = \mathcal{P}(X,r)$ denotes the factorization of X by $\mathcal{P}$ using r factors.*

It is necessary to point out that many commonly used MF algorithms are decomposable, including some of the state-of-the-art techniques, although they are required to satisfy all these four decomposable properties, which seems to be somewhat too strict. Some typical examples are SVD, NMF, PMF, MMMF and their variations, which will be primarily considered and investigated in this work.

Theorem 1. *Suppose X is a BDF matrix in (6), and $\mathcal{P} = (f, \mathcal{D}_W, \mathcal{C}, \mathcal{R})$ is decomposable. Let $(U,V) = \mathcal{P}(X,r)$ and $(U_i, V_i) = \mathcal{P}(X_i, r)(1 \le i \le k)$. We have:*
i. $U = [U_1^T U_2^T \cdots U_k^T]^T$, $V = [V_1^T V_2^T \cdots V_k^T]^T$
ii. $X_{ij} \approx f(U_i V_j^T)$ $(1 \le i,j \le k)$

Proof. i. Consider the optimization problem defined in (1) with decomposable properties of prediction link f, loss function $\mathcal{D}_W$, hard constraint $\mathcal{C}$, and regularizer $\mathcal{R}$; we have:

$$\begin{aligned} & (U,V) = \mathcal{P}(X,r) \\ = & \operatorname*{argmin}_{(U,V)\in\mathcal{C}} \left[\mathcal{D}_W(X, f(UV^T)) + \mathcal{R}(U,V)\right] \\ = & \operatorname*{argmin}_{(U,V)\in\mathcal{C}} \sum_{i=1}^{k} \left[\mathcal{D}_{W_i}(X_i, f(UV^T)_i) + \mathcal{R}(U_i,V_i)\right] \\ = & \operatorname*{argmin}_{(U,V)\in\mathcal{C}} \sum_{i=1}^{k} \left[\mathcal{D}_{W_i}(X_i, f(U_i V_i^T)) + \mathcal{R}(U_i,V_i)\right] \\ = & \bigwedge_{i=1}^{k} \left\{ \operatorname*{argmin}_{(U_i,V_i)\in\mathcal{C}} \left[\mathcal{D}_{W_i}(X_i, f(U_i V_i^T)) + \mathcal{R}(U_i,V_i)\right] \right\} \\ = & \bigwedge_{i=1}^{k} \left\{\mathcal{P}(X_i, r)\right\} = \bigwedge_{i=1}^{k} \left\{(U_i, V_i)\right\} \end{aligned}$$

thus, $U = [U_1^T U_2^T \cdots U_k^T]^T$ and $V = [V_1^T V_2^T \cdots V_k^T]^T$.

ii. This can be derived directly from the decomposable property of prediction link f in (10):

$$X_{ij} \approx f(UV^T)_{ij} = f(U_i V_j^T)$$

and it holds for any $1 \le i,j \le k$, including zero submatrices where $i \ne j$. $\square$

According to Theorem 1, each diagonal block can be factorized independently, and the results can be used directly to approximate not only the non-zero diagonal blocks but also the zero off-diagonal blocks.

4.2 LMF for Collaborative Prediction

Consider predicting the missing values of an incomplete sparse rating matrix through the LMF framework. A sparse rating matrix is permuted into an RBBDF matrix (5) first and further transformed into a BDF matrix (9). LMF is then performed on the resulting BDF matrix to make rating predictions for the original matrix.

Suppose an RBBDF matrix X is transformed into a BDF matrix $\tilde{X} = \text{diag}(\tilde{X}_i)(1 \le i \le k)$. $X_{\mathcal{I}_*\sim\mathcal{J}_*}$ and $\tilde{X}_{\mathcal{I}_*\sim\mathcal{J}_*}$ are used to denote the submatrices in X and $\tilde{X}$ correspondingly. For example, $R_{12} = X_{\mathcal{I}_{B_1}\sim\mathcal{J}_{12}}$ in (5), and it is duplicated twice by $\tilde{X}_{\mathcal{I}_{B_1}\sim\mathcal{J}_{12}}$ in (9). The LMF framework approximates the original matrix X through the approximations of $\tilde{X}$ with three steps:

i. For a decomposable matrix factorization algorithm $\mathcal{P} = (f, \mathcal{D}_W, \mathcal{C}, \mathcal{R})$, obtain the factorization $(U_i, V_i) = \mathcal{P}(\tilde{X}_i, r)$ of each diagonal block $\tilde{X}_i$. Then:

$$\tilde{X}_i \approx f(U_i V_i^T) \triangleq \tilde{X}_i^* \triangleq \tilde{X}_{ii}^* \tag{14}$$

where $\tilde{X}_i^*$ denotes the approximation of $\tilde{X}_i$.

ii. Predict zero blocks $\tilde{X}_{ij}(i \ne j)$ in $\tilde{X}$ using factorizations of $\tilde{X}_i$ and $\tilde{X}_j$:

$$\tilde{X}_{ij} \approx f(U_i V_j^T) \triangleq \tilde{X}_{ij}^* \tag{15}$$

Now $\tilde{X}^* \triangleq \left\{\tilde{X}_{ij}^* | 1 \le i,j \le k\right\}$ approximates $\tilde{X}$.

iii. Average duplicated submatrices in $\tilde{X}^*$ to approximate the corresponding submatrix in X.

Suppose that $X_{\mathcal{I}_*\sim\mathcal{J}_*}$ is duplicated k times in $\tilde{X}$, and the t^{th} duplication is in block $\tilde{X}_{i_t j_t}$, whose approximation is $\tilde{X}^{*(i_t j_t)}_{\mathcal{I}_*\sim\mathcal{J}_*}$. Then the approximation of $X_{\mathcal{I}_*\sim\mathcal{J}_*}$ is:

$$X^*_{\mathcal{I}_*\sim\mathcal{J}_*} = \frac{1}{k}\sum_{t=1}^{k} \tilde{X}^{*(i_t j_t)}_{\mathcal{I}_*\sim\mathcal{J}_*} \tag{16}$$

To make it easier to understand, take $R_{12} = X_{\mathcal{I}_{B_1}\sim\mathcal{J}_{12}}$ in (5) as an example. $X_{\mathcal{I}_{B_1}\sim\mathcal{J}_{12}}$ is duplicated twice in (9): one in $\tilde{X}_{12}$ and the other in $\tilde{X}_{22}$. As a result:

$$X^*_{\mathcal{I}_{B_1}\sim\mathcal{J}_{12}} = \frac{1}{2}(\tilde{X}^{*(12)}_{\mathcal{I}_{B_1}\sim\mathcal{J}_{12}} + \tilde{X}^{*(22)}_{\mathcal{I}_{B_1}\sim\mathcal{J}_{12}})$$

Approximation $X^*_{\mathcal{I}_*\sim\mathcal{J}_*}$ is constructed for each submatrix $X_{\mathcal{I}_*\sim\mathcal{J}_*}$ in X. By piecing them together, approximation $X^* = \{X^*_{\mathcal{I}_*\sim\mathcal{J}_*}\}$ is finally achieved for X.

4.3 Algorithm for RBBDF Permutation

As shown in Section 3.3, permuting a matrix into (R)BBDF structure is equivalent to performing GPVS (recursively) on its bipartite graph. In this work, both the performance and efficiency of graph partitioning algorithms are concerned, as the datasets to experiment on are huge[1]. As a result, multilevel graph partitioning approach is chosen. Perhaps the

[1]One of the four datasets used is Yahoo! Music from KDDCUP 2011, containing approximately 1m users and 0.6m items, which is the largest in present-day datasets.

most widely known and used package for graph partitioning is Metis [13] by Karypis, which is based on multilevel approach. The core routine for GPVS in Metis is Node-based Bisection, which partitions a graph into two disconnected components by a vertex separator.

A density-based algorithm to permute sparse matrices into RBBDF structure is designed, as dense submatrices or subgraphs are usually interpreted as communities, which is widely used in community detection and graph clustering problems [8]. The density of a matrix X is defined as $\rho(X) = \frac{n(X)}{s(X)}$, where $n(X)$ is the number of non-zeros in X, and $s(X)$ is the area of X. The average density of k matrices $X_1X_2\cdots X_k$ is defined as $\bar{\rho}(X_1X_2\cdots X_k) = \frac{\sum_{i=1}^{k} n(X_i)}{\sum_{i=1}^{k} s(X_i)}$. Note that the density of a matrix is equal to the density of its corresponding bipartite graph [8].

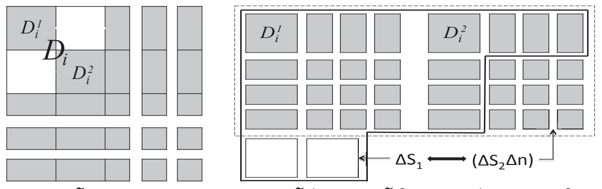
For an RBBDF matrix X with k diagonal blocks $D_1D_2\cdots D_k$ (e.g., the matrix in (5) has 3 diagonal blocks: $D_{11}D_{12}$ and D_2, and the original rating matrix is viewed as a single diagonal block), $\tilde{X} = \mathrm{diag}(\tilde{X}_1\tilde{X}_2\cdots\tilde{X}_k)$ is used to denote its corresponding BDF matrix (e.g., the matrix in (9)). Algorithm 1 shows the procedure, followed by more detailed explanations and analyses. RBBDF$(X, \hat{\rho}, 1)$ is called to start the procedure.

Algorithm 1 RBBDF$(X, \hat{\rho}, k)$

```
Require:
    User-item rating matrix: $X$
    Average density requirement: $\hat{\rho}$
    Current number of diagonal blocks in $X$: $k$
Ensure:
    Matrix $X$ be permuted into RBBDF structure
    BDF matrix $\tilde{X}$ which is constructed from $X$
 1: $\rho \leftarrow \bar{\rho}(\tilde{X}_1\tilde{X}_2\cdots\tilde{X}_k)$
 2: if $\rho \geq \hat{\rho}$ then
 3:     return $\tilde{X}$ ▷ Density requirement has been reached
 4: else
 5:     $[D_{s_1}D_{s_2}\cdots D_{s_k}] \leftarrow$ Sort$\big([D_1D_2\cdots D_k]\big)$ ▷ Sort diagonal blocks by size in decreasing order
 6:     for $i \leftarrow 1$ to $k$ do
 7:         $[D^1_{s_i}D^2_{s_i}] \leftarrow$ MetisNodeBisection$(D_{s_i})$ ▷ Partition $D_{s_i}$ into 2 diagonals using core routine of Metis
 8:         if $\bar{\rho}(\tilde{X}_{s_1}\cdots\tilde{X}_{s_{i-1}}\tilde{X}^1_{s_i}\tilde{X}^2_{s_i}\tilde{X}_{s_{i+1}}\cdots\tilde{X}_{s_k}) > \rho$ then
 9:             $X' \leftarrow$ Permute $D_{s_i}$ into $[D^1_{s_i}D^2_{s_i}]$ in $X$
10:             RBBDF$(X', \hat{\rho}, k+1)$ ▷ Recurse
11:             break ▷ No need to check the next diagonal
12:         end if
13:     end for
14:     return $\tilde{X}$ ▷ No diagonal improves average density
15: end if
```

Algorithm 1 requires a 'density requirement' $\hat{\rho}$ as input, which is the expected average density of submatrices $\tilde{X}_1\cdots\tilde{X}_k$ in the final BDF matrix $\tilde{X}$. In each recursion, the algorithm checks each diagonal block D_i of X in decreasing order of matrix areas. If the average density of extracted submatrices can be improved by partitioning a diagonal block, then the algorithm takes the partitioning and recurses, until $\hat{\rho}$ is reached or none of the diagonal blocks can improve the average density any more.

Note that, to gain a high efficiency, a fundamental heuristic is used in this algorithm, which is the area of diagonal blocks, and we explain its rationality here.



(a) $\tilde{X}_i$ for D_i (b) $\tilde{X}^1_i$ and $\tilde{X}^2_i$ for D^1_i and D^2_i

Figure 3: Partition a diagonal block, rearrange its corresponding borders, and extract new submatrices. The shaded areas represent non-zero blocks.

Figure 3(a) shows a diagonal block D_i along with its corresponding borders. Note that this figure is, in fact, the submatrix $\tilde{X}_i$ in $\tilde{X}$ corresponding to D_i. Two new submatrices $\tilde{X}^1_i$ and $\tilde{X}^2_i$ are constructed when D_i is partitioned into D^1_i and D^2_i, which are boxed by dotted lines in Figure 3(b). One can see that the area boxed by solid lines in Figure 3(b) constitutes the original submatrix $\tilde{X}_i$. As a result, transforming $\tilde{X}_i$ to $\tilde{X}^1_i$ and $\tilde{X}^2_i$ is essentially removing the two zero blocks and replacing them with some duplicated non-zero blocks.

Let $s = \sum_{t=1}^{k} s(\tilde{X}_t)$ and $n = \sum_{t=1}^{k} n(\tilde{X}_t)$; let Δs_1 be the total area of removed zero blocks, Δs_2 be the total area of duplicated non-zero blocks, and Δn be the number of nonzeros in Δs_2. The increment of average density after partitioning D_i is:

$$\Delta\rho = \rho' - \rho = \frac{n + \Delta n}{s - \Delta s_1 + \Delta s_2} - \frac{n}{s} = \frac{s\Delta n + n\Delta s}{s(s - \Delta s)} \quad (17)$$

where ρ and ρ' are the average densities of diagonal blocks in $\tilde{X}$ before and after partitioning D_i, and $\Delta s \triangleq \Delta s_1 - \Delta s_2$.

Because $s - \Delta s > 0$, we have the following:

$$\Delta\rho > 0 \leftrightarrow s\Delta n + n\Delta s = s\Delta n + n(\Delta s_1 - \Delta s_2) > 0 \quad (18)$$

If $\Delta s > 0$, then (18) holds naturally. Otherwise, the following is required:

$$\frac{n}{s} < \frac{\Delta n}{\Delta s_2 - \Delta s_1} \quad (19)$$

Although not guaranteed, (19) can usually be satisfied as the following property usually holds:

$$\frac{n}{s} < \frac{\Delta n}{\Delta s_2} < \frac{\Delta n}{\Delta s_2 - \Delta s_1} \quad (20)$$

Intuitionally, (20) means that the density of duplicated non-zero blocks after partitioning is usually greater than the original average density of k submatrices $\tilde{X}_1\tilde{X}_2\cdots\tilde{X}_k$, as the latter contains many zero blocks. Additionally, a large Δs tends to yield a large density increment $\Delta\rho$ according to (17), which leads to adopting areas of diagonal blocks as heuristics. It will be verified experimentally that this heuristic improves the average density at the first attempt nearly all the time.

The time complexity of Node-based bisection in Metis is $O(n)$, where n is the number of non-zeros in a matrix [14]. Suppose that matrix X is permuted into an RBBDF structure which has k diagonal blocks ($k \ll n$) in the end, and the algorithm chooses the largest diagonal block for partitioning in each recursion; then, the height of the recursion tree will be $O(\lg k)$ and the total computational cost in each level of the tree is $O(n)$. As a result, the time complexity of Algorithm 1 is $O(n \lg k)$.

5. EXPERIMENTS

5.1 Datasets Description

A series of experiments were conducted on four real-world datasets: MovieLens-100K, MovieLens-1M, DianPing and Yahoo!Music. The MovieLens dataset is from GroupLens Research. We also collected a year's data from a famous restaurant rating web site *DianPing*[2] from Jan. 1st to Dec. 31st 2011 and selected those users with 20 or more ratings. The ratings also range from 0 to 5. The Yahoo!Music dataset is from KDD Cup 2011, and its ratings range from 0 to 100. Statistics of these datasets are presented in Table 1.

Table 1: Statistics of four datasets

	ML-100K	ML-1M	DianPing	Yahoo!Music
#users	943	6,040	11,857	1,000,990
#items	1,682	3,952	22,365	624,961
#ratings	100,000	1,000,209	510,551	256,804,235
#ratings/user	106.045	165.598	43.059	256.550
#ratings/item	59.453	253.089	22.828	410.912
average density	0.0630	0.0419	0.00193	0.000411

These datasets are chosen as they have different sizes and densities. Besides, two of them have more users than items, and the others are the opposite. We expect to verify how our framework works on datasets of different sizes and densities.

5.2 Algorithms and Evaluation Metrics

Four popular and state-of-the-art matrix factorization algorithms are the subject of experimentation in this work:

SVD: The Alternating Least Squares (ALS) algorithm in [17] is used for SVD learning.

NMF: The NMF algorithm based on divergence cost in [19] is used. We also use F-norm regularizer, similar to SVD.

PMF: The Bayesian PMF by Markov Chain Monte Carlo method in [25] is used.

MMMF: The fast version of MMMF in [23] is used.

For easier comparison with previous proposed methods in the literature, we use *Root Mean Square Error (RMSE)* to measure the prediction accuracy in this work.

For N rating-prediction pairs $\langle r_i, \hat{r}_i \rangle$:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(r_i - \hat{r}_i)^2}{N}}$$

Five-fold cross validation is conducted to calculate the average RMSE for MovieLens and DianPing. The Yahoo! Music dataset itself is partitioned into training and validation sets, which are used for training and evaluation, respectively.

5.3 Analyses of RBBDF Algorithm

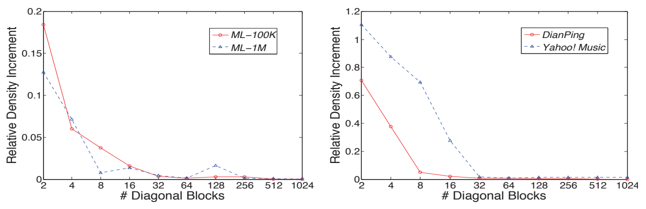
5.3.1 Number of diagonal blocks

The only parameter to be tuned in the RBBDF algorithm is the average density requirement $\hat{\rho}$. Intuitionally, a low density requirement gives less and larger diagonal blocks in $\tilde{X}$, and vice versa. The relationship between the number of diagonal blocks k and the density requirement $\hat{\rho}$ on four datasets is shown by red solid lines in Figure 4.

We see that the number of diagonal blocks increases faster and faster with an increasing density requirement. To investigate the underlying reason, a more straightforward view is given by the relative density increment in Figure 5. Suppose that the current number of diagonal blocks is k; the average density of $\tilde{X}_1\tilde{X}_2\cdots\tilde{X}_k$ is $\bar{\rho}_k$, and the average density goes to $\bar{\rho}_{k+1}$ after partitioning a diagonal block. Then, the relative density increment is $\Delta\rho/\rho_1 = (\bar{\rho}_{k+1} - \bar{\rho}_k)/\bar{\rho}_1$, where the constant $\bar{\rho}_1$ is the density of the whole original matrix.

[2]http://www.dianping.com

Experimental results show that the relative density increment becomes lower and lower as the number of diagonal blocks increases. As a result, it is relatively easy to improve the average density at the beginning, as partitioning a large diagonal block D_i gains a large density increment $\Delta\rho$. However, this process tends to be more and more difficult when diagonal blocks become small. The experimental result is in accordance with the analysis in (17)$\sim$(20). These results partially verify the heuristic used in Algorithm 1.



(a) ML-100K & ML-1M (b) DianPing & Yahoo!Music

Figure 5: Relationship between the Relative Density Increment $\Delta\rho/\rho_1$ and the Current number of diagonal blocks k on four datasets.

5.3.2 Verification of heuristic

The *First Choice Hit Rate (FCHR)* is used to verify the heuristic used in the RBBDF algorithm, as we expect the average density to be improved by partitioning the first diagonal block D_{s_1} in the sorted list $[D_{s_1}D_{s_2}\cdots D_{s_k}]$, in which case there is no need to check the remaining diagonal blocks.

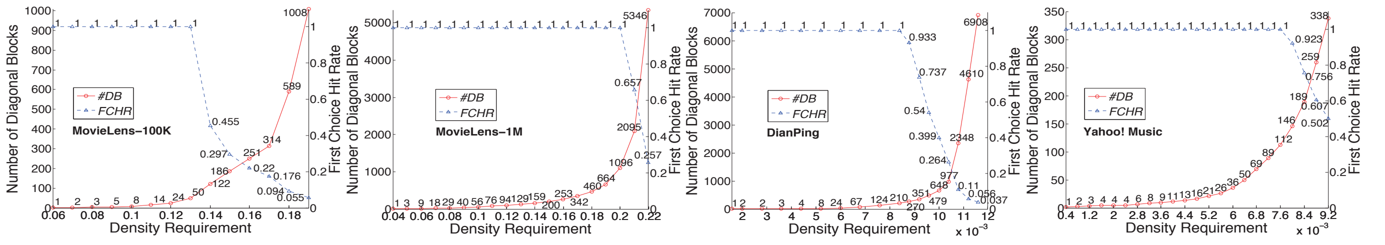
$$\text{FCHR} = \frac{\#\ \textit{recursions where } D_{s_1} \textit{ is chosen}}{\#\ \textit{recursions in total}}$$

One can see that FCHR = 1 means that average density can always be improved by partitioning the largest diagonal block directly. The relationships between FCHR and density requirement on four datasets are shown by blue dotted lines in Figure 4. On all of the four datasets, FCHR = 1 at the beginning and begins to drop when density requirement is high enough, which is also in accordance with the analysis in Section 4.3. As a result, by taking the areas of diagonal blocks as a heuristic, only one diagonal block is partitioned in each recursion, and there is no redundant computation when an appropriate density requirement is given.

When the density requirement is high, we have FCHR $<$ 1, and redundant computation will be introduced: we might partition a diagonal block without improving the average density. However, we would like to note that it does not matter very much in practice. First, when considering the $O(n)$ complexity of the Node-based Bisection, it will be faster and faster to partition diagonal blocks as they become smaller. Second, there is no need to split a matrix into hundreds or even thousands of diagonal blocks in practice. According to our experiments in the following sections, it is sufficient to gain both high prediction accuracy and computational efficiency by partitioning a matrix into a small number of diagonal blocks, in which case we have FCHR = 1.

5.3.3 Computational Time of RBBDF Algorithm

Experiments were conducted on an 8-core 3.1GHz linux server with 64G RAM. We tuned the density requirement $\hat{\rho}$



(a) ML-100K (b) ML-1M (c) DianPing (d) Yahoo! Music

Figure 4: Number of Diagonal Blocks (#DB, solid lines) and First Choice Hit Rate (FCHR, dotted lines) under different density requirements $\hat{\rho}$. The tuning steps of $\hat{\rho}$ are 0.01, 0.01, 0.0008 and 0.0004, respectively.

to achieve the expected number of diagonal blocks k. The run time of the RBBDF algorithm is shown in Table 2.

In the experiments, we see that the run time increases along with the number of diagonal blocks, and it takes less time to partition a submatrix as they become smaller. Moreover, the time used by the RBBDF algorithm is much less than that used for training an MF model on matrix X. We will show the results on model training time in Section 5.5.

Table 2: Computational time of the RBBDF algorithm with different numbers of diagonal blocks k.

k	5	10	15	20	50	100	150	200
ML-100K / ms	160	180	196	208	224	340	422	493
ML-1M / s	4.45	5.61	6.25	6.76	8.31	9.51	10.25	10.74
DianPing / s	6.01	9.69	11.61	12.84	14.64	15.06	16.18	16.95
Yahoo! / min	8.03	9.54	10.95	12.08	17.67	21.83	23.35	24.73

5.4 Prediction Accuracy

5.4.1 Number of latent factors

The number of latent factors r plays an important part in MF algorithms. It would be insufficient for approximating a matrix if r is too low, and would be computationally expensive if r is too high. As the diagonal blocks in $\tilde{X}$ and the original matrix X are of different sizes, it's important to investigate how to choose a proper r in practical applications.

We use MovieLens-1M to test the impact of r in the LMF framework. The density requirement is $\hat{\rho} = 0.055$, and matrix X is permuted into an RBBDF structure with 4 diagonal blocks; then, $\tilde{X} = \text{diag}(\tilde{X}_1, \tilde{X}_2, \tilde{X}_3, \tilde{X}_4)$ is constructed. Some statistical information about $\tilde{X}$ is shown in Table 3.

Table 3: Statistics of the four diagonal blocks

	$\tilde{X}_1$	$\tilde{X}_2$	$\tilde{X}_3$	$\tilde{X}_4$
#users	1,507	1,683	1,743	1,150
#items	2,491	3,108	3,616	3,304
#ratings	118,479	259,665	462,586	192,267
density	0.0316	0.0496	0.0734	0.0506

We tuned r from 5 to 100, with a tuning step of 5. It's necessary to note that r is required to be comparable with $\min(m, n)$ in MMMF, where m and n are the numbers of users and items in X. However, it would be time consuming to train a model using thousands or even millions of factors. Fortunately, according to [23], it's sufficient to use much smaller values of r to achieve satisfactory performance in practice ($r = 100$ is used in [23] for ML-1M). As a result, the tuning range of $5 \sim 100$ is also used for MMMF.

For each of the four MF algorithms, two sets of experiments were conducted. First, we approximate the original matrix X using r factors directly, and record the RMSE. Second, predictions are made by the LMF framework in Section 4.2 using the four diagonal blocks in $\tilde{X}$, each with r factors. Cross-validation is performed on X to find the best regularization parameters for each MF method. In SVD and NMF, λ is set to 0.065; in PMF, λ_U and λ_V are both 0.002; and in MMMF, the regularization constant C is set to 1.5. RMSE v.s. the number of latent factors r is shown in Figure 7.

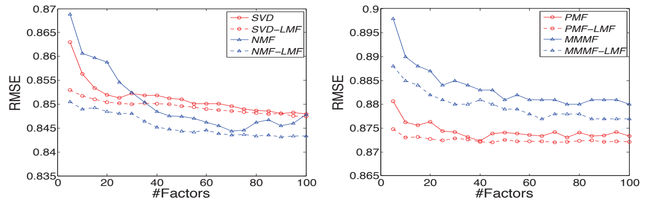
Experimental results show that better performance in terms of RMSE can be achieved in the LMF framework. Furthermore, the improvement tends to be more obvious when the number of latent factors r is relatively small. This result could arise because, in such cases, r is not sufficient to approximate the original matrix X, while it is sufficient to approximate relatively small submatrices in $\tilde{X}$. We view this as an advantage of the LMF framework, as better performance can be achieved with fewer factors, which benefits the model complexity and training time.

5.4.2 Different density requirements

The final number of diagonal blocks k in $\tilde{X}$ is different under different density requirements $\hat{\rho}$. We experimented RMSE with different density requirements. The number of latent factors r is set to 60, as we find it sufficient to smooth the performance improvement on the datasets. The regularization coefficients are the same: $\lambda = 0.065$ for SVD and NMF, $\lambda_U = \lambda_V = 0.002$ for PMF, and $C = 1.5$ for MMMF.

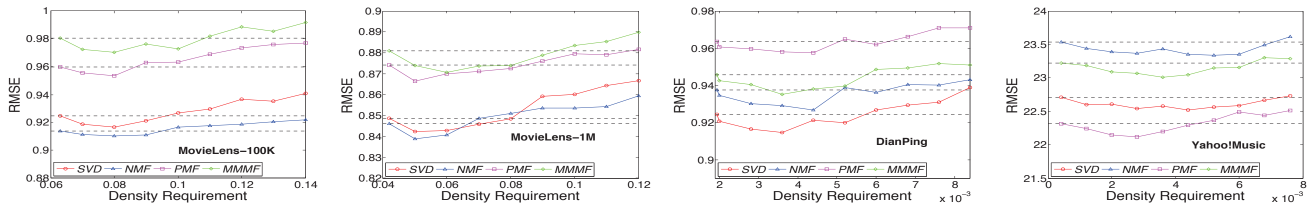
The RMSE versus different choices of $\hat{\rho}$ on all of the four datasets are plotted in Figure 6. In each subfigure, the four curves correspond to the four MF methods used, which are SVD, NMF, PMF and MMMF. The density requirement on the first point of each curve is the average density of the corresponding dataset; as a result, RMSE on this point is the baseline performance for the matrix factorization algorithm. Thus, points below the beginning point of a curve indicate an improvement on prediction accuracy, and vice versa.

Experimental results show that our LMF framework helps decomposable MF algorithms to gain better prediction accuracies if appropriate density requirements are given, but might bring negative effects if $\hat{\rho}$ is not appropriately set.



(a) SVD & NMF (b) PMF & MMMF

Figure 7: RMSE v.s. different numbers of latent factors. Solid/dotted lines are results of approximating X directly/through the LMF framework.



(a) MovieLens-100K (b) MovieLens-1M (c) DianPing (d) Yahoo! Music

Figure 6: RMSE on four datasets using the LMF framework under different density requirements. Dotted lines in each subfigure represent baseline performance of the corresponding matrix factorization algorithm.

Here, by 'appropriate', we mean that $\hat{\rho}$ is not too high. According to the experimental results on four datasets, better prediction accuracies are achieved along with an increasing $\hat{\rho}$ (and also the number of diagonal blocks k) at the beginning, but the performance tends to drop when $\hat{\rho}$ is set too high. In our view, this is not surprising because many small scattered diagonal blocks are extracted when a high density requirement is set, which would bring negative effects to MF algorithms. Table 4 presents the average number of users and items in the diagonal blocks of $\tilde{X}$ under different $\hat{\rho}$ on MovieLens-1M. We see that the average number of users goes to only a hundred or less when $\hat{\rho} \geq 0.1$.

Table 4: Average number of users and items in diagonal blocks of $\tilde{\mathbf{X}}$ under different $\hat{\rho}$ on MovieLens-1M.

$\hat{\rho}$	0.045	0.052	0.060	0.069	0.081	0.102	0.129	0.160
k	2	4	8	16	32	64	128	256
Avg #users	3020	1520	779	409	220	128	82	61
Avg #items	3170	3129	3055	3064	3015	3007	3015	3030

However, better performance is achieved given appropriate density requirements. By combining this observation with the experimental results in Section 5.3, it is neither reasonable nor necessary to use high density requirements that result in hundreds or even thousands of diagonal blocks.

Table 5 shows the best RMSE achieved on each dataset for each MF method. To calculate the average RMSE on each dataset, five-fold cross-validation was conducted on MovieLens and DianPing, and experiments were conducted five times on Yahoo! Music. The standard deviations were $\leq$ 0.002 on MovieLens and DianPing, and were ≤ 0.01 on Yahoo! Music. We see that, in the best cases, MF algorithms benefit from the LMF framework in terms of RMSE on all of the four datasets. Specifically, the sparser a matrix is, the higher RMSE increment LMF tends to gain.

5.5 Speedup by Parallelization

An important advantage of LMF is that, once the BDF matrix $\tilde{X} = \text{diag}(\tilde{X}_i)$ is constructed, diagonal blocks $\tilde{X}_i$ can be trained in parallel. According to the decomposable property in Theorem 1, sub-problems of learning different (U_i, V_i) are not coupled; as a result, there is no need to implement rather complicated parallel computing algorithms. In fact, simple multi-threading technique is adequate for the task, which contributes to the scalability of recommender systems while, at the same time, keeps system simplicity.

The experiment comprises three stages. In the first stage, X is permuted into an RBBDF structure, and a BDF matrix $\tilde{X} = \text{diag}(\tilde{X}_i)(1 \leq i \leq k)$ is constructed. As we have 8 cores, the density requirement is tuned on each dataset to construct $\tilde{X}$ with 8 diagonal blocks. In the second stage, each diagonal block is factorized independently with a thread, and (U_i, V_i) is achieved. In the last stage, (U_i, V_i) from all of the diagonal blocks are used to approximate the original matrix X by LMF. The computational time consumed in each stage is recorded (in the second stage, the time recorded is the longest among all of the diagonal blocks). Finally, the total time of the three stages is adopted to evaluate the overall efficiency. The number of factors and the regularization coefficients are the same as those in Section 5.4.2. The results are shown in Table 6, where 'Base' represents the computational time of factorizing X directly, 'LMF' is the time used by the LMF framework, and 'Speedup' is 'Base/LMF'.

Table 6: Computational time and speedup by multithreading with 8 diagonal blocks.

Method	MovieLens-100K			MovieLens-1M		
	Base	LMF	Speedup	Base	LMF	Speedup
SVD	23.9s	7.7s	3.10	184.9s	43.4s	4.26
NMF	8.7s	3.9s	2.23	86.6s	22.1s	3.92
PMF	43.8s	11.6s	3.78	265.1s	60.1s	4.41
MMMF	19.6min	4.71min	4.16	1.73h	21.5min	4.83

Method	DianPing			Yahoo!Music		
	Base	LMF	Speedup	Base	LMF	Speedup
SVD	143.7s	35.7	4.03	6.22h	1.21h	5.14
NMF	64.4s	16.6s	3.88	4.87h	1.05h	4.64
PMF	190.5s	44.1s	4.32	7.91h	1.48h	5.34
MMMF	48.5min	10.2min	4.75	38.8h	6.22h	6.24

Experimental results show that the LMF framework helps to save a substantial amount of model training time through very simple multithreading parallelization techniques. This is especially helpful when learning large magnitude datasets, which is important in real-world recommender systems.

6. DISCUSSIONS

Unlike benchmark datasets, rating matrices in real-world recommender systems usually change dynamically as new ratings are made by users continuously. A typical way to settle this problem in practice is to retrain MF models periodically or when a predefined prediction accuracy threshold (say RMSE) is reached. However, it would be time consuming to refactorize large rating matrices as a whole and to do so frequently. In the LMF framework, however, it is possible to only refactorize those submatrices whose prediction accuracies have reached a predefined threshold, rather than refactorize the whole matrix, which further benefits system scalability. This potential advantage that LMF might bring about will be investigated both by simulation and in practical real-world recommender systems in future work.

7. CONCLUSIONS

In this paper, we explored the BDF, BBDF and RBBDF structures of sparse matrices and their properties in terms of matrix factorization. The LMF framework is proposed, and to explicitly indicate the scope of matrix factorizations

Table 5: Best performance achieved in LMF with corresponding density requirement $\hat{\rho}$ and number of diagonal blocks k. Bold numbers indicate improvements that are ≥ 0.01 on MovieLens and DianPing or ≥ 0.2 on Yahoo!Music. The standard deviations are ≤ 0.002 on MovieLens and DianPing and ≤ 0.01 on Yahoo!Music.

Method	MovieLens-100K				MovieLens-1M				DianPing				Yahoo!Music			
	baseline	$\hat{\rho}$	k	RMSE	baseline	$\hat{\rho}$	k	RMSE	baseline	$\hat{\rho}$	k	RMSE	baseline	$\hat{\rho}$	k	RMSE
SVD	0.9249	0.08	3	0.9165	0.8487	0.05	3	0.8423	0.9244	0.0036	3	0.9145	22.713	0.0044	13	22.519
NMF	0.9138	0.08	3	0.9102	0.8461	0.05	3	0.8388	0.9376	0.0044	4	**0.9267**	23.538	0.0052	21	**23.335**
PMF	0.9598	0.08	3	0.9534	0.8741	0.05	3	0.8664	0.9636	0.0044	4	0.9575	22.312	0.0028	6	22.121
MMMF	0.9807	0.08	3	**0.9703**	0.8810	0.06	9	0.8740	0.9457	0.0036	3	**0.9352**	23.218	0.0036	9	**23.007**

that the framework can handle, decomposable properties of matrix factorization algorithms were investigated in detail. Based on graph partitioning theories, we designed a density-based algorithm to permute sparse matrices into RBBDF structures, and studied its algorithmic properties both formally and experimentally. Experimental results show that LMF helps the matrix factorization algorithms we studied to gain better performance and, at the same time, contributes to system scalability by simple parallelization techniques.

8. ACKNOWLEDGEMENT

The authors thank Jun Zhu for the fruitful discussions and the reviewers for their constructive suggestions. This work was supported by Natural Science Foundation (60903107, 61073071) and National High Technology Research and Development (863) Program (2011AA01A205) of China.

9. REFERENCES

[1] H. Abdi and L. J. Williams. Principal component analysis. *WIREs Comp Stat*, 2:433–459, 2010.

[2] C. Alzate and J. A. Suykens. Multiway spectral clustering with out-of-sample extensions through weighted kernel pca. *PAMI*, 32:335–347, 2010.

[3] B. Arsic et al. Graph spectral techniques in computer sciences. *Appl. Anal. Discrete Math*, 6:1–30, 2012.

[4] C. Aykanat et al. Permuting sparse rectangular matrices into block-diagonal form. *SISC*, 2004.

[5] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *JMLR*, 2005.

[6] E. Boman and M. Wolf. A nested dissection approach to sparse matrix partitioning for parallel computations. *AMM*, 2007.

[7] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is np-hard. *IPL*, 1992.

[8] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *TKDE*, 2012.

[9] I. S. Dhilon et al. Concept decompositions for large sparse text data using clustering. *JMLR*, 2001.

[10] E. Gallopoulos and D. Zeimpekis. Clsi: A flexible approximation scheme from clustered term-document matrices. *SDM*, 2005.

[11] R. Gemulla et al. Large-scale matrix factorization with distributed stochastic gradient descent. *KDD*, 2011.

[12] J. Herlocker et al. An algorithmic framework for performing collaborative filtering. *SIGIR*, 1999.

[13] G. Karypis. Metis-a software package for partitioning unstructured graphs, meshes, and computing fill reducing orderings of sparse matrices-v5.0. 2011.

[14] G. Karypis et al. A fast and high quality multilevel scheme for partitioning irregular graphs. *SISC*, 1999.

[15] J. Kim, I. Hwang, Y. Kim, et al. Genetic approaches for graph partitioning: A survey. *CECCO*, 2011.

[16] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. *KDD*, 2008.

[17] Y. Koren, R. Bell, et al. Matrix factorization techniques for recommender systems. *Computer*, 2009.

[18] D. Lee and H. Seung. Learning the parts of objects with nonnegative matrix factorization. *Nature*, 1999.

[19] D. Lee and H. Seung. Algorithms for non-negative matrix factorization. *NIPS*, 2001.

[20] C. Liu, H. Yang, J. Fan, L. He, et al. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. *WWW*, 2010.

[21] J. Liu, M. Chen, J. Chen, et al. Recent advances in personal recommender systems. *JISS*, 2009.

[22] M. J. Pazzani and D. Billsus. Content-based recommendation systems. *Adaptive Web LNCS*, 2007.

[23] J. Rennie et al. Fast maximum margin matrix factorization for collaborative prediction. *ICML*, 2005.

[24] P. Resnick et al. Grouplens: An open architecture for collaborative filtering of netnews. *CSCW*, 1994.

[25] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. *ICML*, 2008.

[26] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. *NIPS*, 2008.

[27] P. Sanders and C. Schulz. Engineering multilevel graph partitioning algorithms. *ESA*, 2011.

[28] B. Sarwar et al. Application of dimension reduction in recommender systems - a case study. *WebKDD*, 2000.

[29] B. Sarwar, G. Karypis, et al. Item-based collaborative filtering recommendation algorithms. *WWW*, 2001.

[30] B. Sarwar, G. Karypis, et al. Incremental singular value decomposition algorithms for highly scalable recommender systems. *ICCIT*, 2002.

[31] B. Savas et al. Clustered low rank approximation of graphs in information science applications. *SDM*, 2011.

[32] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. *KDD*, 2008.

[33] N. Srebro and T. Jaakkola. Weighted low-rank approximations. *ICML*, 2003.

[34] N. Srebro, J. Rennie, and T. S. Jaakkola. Maximum-margin matrix factorization. *NIPS*, 2005.

[35] X. Su and T. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in AI.*, 2009.

[36] G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. *ICDM*, 2008.

[37] B. Xu, J. Bu, and C. Chen. An exploration of improving collaborative recommender systems via user-item subgroups. *WWW*, 2012.

[38] G. Xue, C. Lin, Q. Yang, et al. Scalable collaborative filtering using cluster-based smoothing. *SIGIR*, 2005.