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), $\tilde{X} = \text{diag}(\tilde{X}_1\tilde{X}_2\cdots\tilde{X}_k)$ 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.

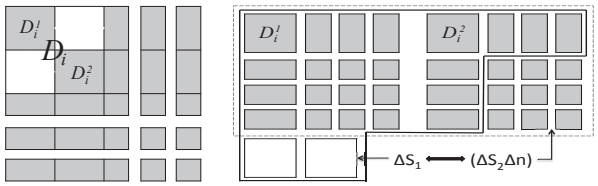
Algorithm 1 RBBDF$(X, \hat{\rho}, k)$

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: ρ ← ρ̄(X̃_1 X̃_2 ··· X̃_k)
2: if ρ ≥ ρ̂ then
3:    return X̃ ▷ Density requirement has been reached
4: else
5:    [D_{s_1} D_{s_2} ··· D_{s_k}] ← Sort([D_1 D_2 ··· D_k]) ▷ Sort diagonal blocks by size in decreasing order
6:    for i ← 1 to k do
7:       [D^1_{s_i} D^2_{s_i}] ← MetisNodeBisection(D_{s_i}) ▷ Partition D_{s_i} into 2 diagonals using core routine of Metis
8:       if ρ̄(X̃_{s_1} ··· X̃_{s_{i-1}} X̃^1_{s_i} X̃^2_{s_i} X̃_{s_{i+1}} ··· X̃_{s_k}) > ρ then
9:          X' ← Permute D_{s_i} into [D^1_{s_i} D^2_{s_i}] in X
10:         RBBDF(X', ρ̂, k + 1) ▷ Recurse
11:         break ▷ No need to check the next diagonal
12:      end if
13:   end for
14:   return X̃ ▷ 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) $\tilde{X}_i^1$ and $\tilde{X}_i^2$ for D_i^1 and D_i^2

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 $\tilde{X}_i^1$ and $\tilde{X}_i^2$ are constructed when D_i is partitioned into D_i^1 and D_i^2, 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 $\tilde{X}_i^1$ and $\tilde{X}_i^2$ 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. 1^{st} to Dec. 31^{st} 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.

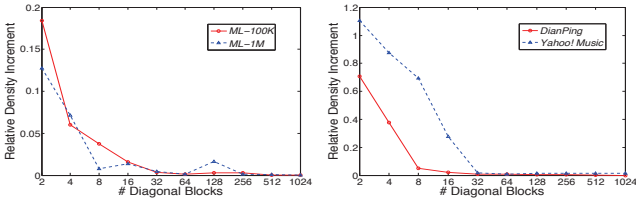
5.3 Analyses of RBBDF Algorithm

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.

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)∼(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.

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

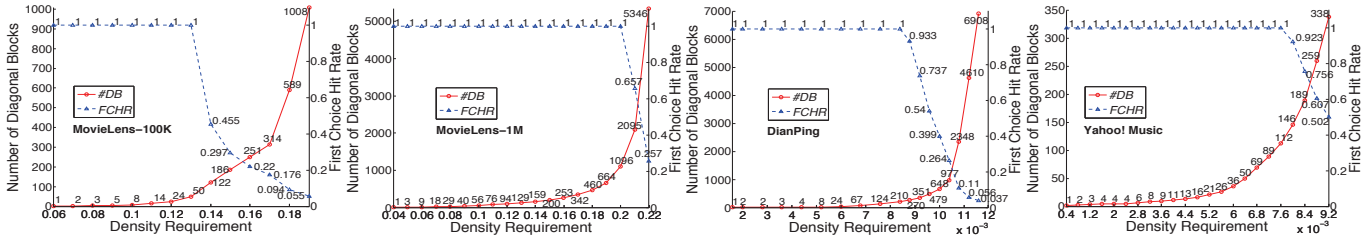
One can see that FCHR = 1 means that average density can always be improved by partitioning the largest diagonal block directly. The relationships between FCHR and density requirement on four datasets are shown by blue dotted lines in Figure 4. On all of the four datasets, FCHR = 1 at the beginning and 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 $\hat{\rho}$

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



(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.

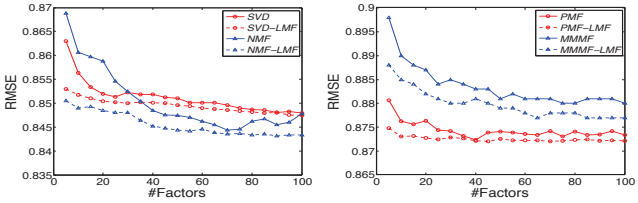
Experimental results show that better performance in terms of RMSE can be achieved in the LMF framework. Furthermore, the improvement tends to be more obvious when the number of latent factors r is relatively small. This result could arise because, in such cases, r is not sufficient to approximate the 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.

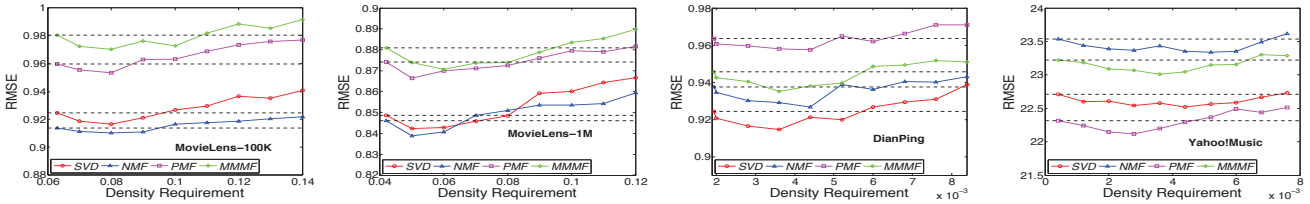
The RMSE versus different choices of $\hat{\rho}$ on all of the four datasets are plotted in Figure 6. In each subfigure, the four curves correspond to the four MF methods used, which are SVD, NMF, PMF and MMMF. The density requirement on the first point of each curve is the average density of the corresponding 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 $\tilde{X}$ 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 $\leq$ 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.