International Journal of u- and e- Service, Science and Technology
Vol.9, No. 7 (2016), pp.131-138
http://dx.doi.org/10.14257/ijunesst.2016.9.7.13

A Parallel Personalized Recommendation Algorithm using Bipartite Graphs

Hao Huang[1], Sotirios G. Ziavras[2] and Yaojie Lu[2]

[1]School of Information Technology, University of International Business and Economics, Beijing 100029, PR China
[2]Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, U.S.A.
[1]hvictory10@163.com

Abstract

BDM-NBI algorithm is proposed in this paper. It focuses on the analysis of a personalized recommendation algorithm that utilizes a weighted bipartite graph suitable for processing big data. To improve the performance of this recommendation algorithm through parallel processing techniques, a sparse matrix partitioning algorithm is then developed that uses the bipartite graph as input. Our algorithm adopts bipartite graph partitioning using a vertex separator method that partitions a high-dimensional sparse matrix into a pseudo-block based diagonal matrix. Then, the recommendation algorithm analyzes all weighted sub-matrices in parallel. We produce the global recommendation weighted matrix by merging all of the sub-matrices in parallel. Experiments with Hadoop show that our algorithm has good approximation for small matrices and excellent scalability.

Keywords: *Personalized Recommendation; Parallelization; Hadoop; Sparse Matrix Partition*

1. Introduction

Information overloading in the Internet requires the introduction of automatic filtering techniques to make sense out of vast amounts of data [1], the research area of personalized recommendation systems is very promising in this regard [2]. Recommender systems apply statistical and knowledge discovery techniques to the problem of making product recommendations based on patterns of previously recorded data [10]. The central problem in the design of recommender systems is how to improve the accuracy [3] and diversity [4] of the recommendations. In this paper, we focus on producing good recommendations when processing large data sets. Recommendation algorithms normally handle high dimensional sparse matrices. If the dimension of the matrix is in the tens of thousands, the performance of the algorithm becomes a major issue [12]. As the basis of our work, we adopt the network-based inference (NBI) recommendation method that works with weighted bipartite graphs containing user and object nodes [4]. This method has remarkably higher accuracy and personalization effectiveness than the widely applied global ranking method [5] and collaborative filtering [9].

Sparse matrices are often used to represent data for this problem with the rows and columns represent objects and users, respectively. We first apply bipartite graph partitioning with a vertex separator method that divides the sparse matrix into several disconnected parts of comparable size. Then, the NBI algorithm is run using coarse-grain parallelization [8]. The dissection method improves the big data processing performance of NBI very dramatically. In the matrix partitioning procedure, we introduce dummy nodes that produce small inaccuracy but help partition the matrix into a singly bordered block-diagonal form (SB) that facilitates parallel processing [11].

More specifically, our target problem is permuting rows and columns of an $m \times n$ adjacent matrix A of objects and users, with elements a_{ij} that show connections between users $j \in [1, \dots, n]$ and objects $i \in [1, \dots, m]$, into a k-way SB form, as per Equation 1 [7].

$$A = PA'Q = \begin{bmatrix} A_{11} & \cdots & A_{1k} \\ \vdots & \ddots & \vdots \\ A_{k1} & \cdots & A_{kk} \\ A_{(k+1)1} & \cdots & A_{(k+1)k} \end{bmatrix} = \begin{bmatrix} B_1 & & \\ & \ddots & \\ & & B_k \\ R_1 & \cdots & R_k \end{bmatrix} = A_{SB} \tag{1}$$

P and Q denote row and column permutation matrices, respectively, which must be computed. The lower border sub-matrix $R = (R_1 \dots R_k)$ represents a coupling row containing submatrices R_i, for $i=1,\dots, k$, created such that nonzero elements exist in the columns of at least two diagonal blocks. The NBI algorithm is then run in parallel on $\{B_1, \dots, B_k, R\}$.

2. NBI Algorithm

Denoting the object set by O = {o1, o2, … , om} and the user set by U ={u1, u2, … , un}, the input to the recommendation system can be completely represented by a bipartite network having m + n nodes where an object is connected with a user if and only if this object has been chosen by this user. Based on this bipartite object-user network, an object-object network can be constructed where each node represents an object and two nodes are connected if and only if both have been collected by at least one user.

In the NBI algorithm [4], the input is the user-object sparse matrix and the output is a weighted matrix W. W is a $m \times m$ square matrix that describes the object-object network and its dimensionality is the number of objects. An element in W is produced by applying Equation 2.

$$w_{ij} = \frac{1}{d(o_j)} \sum_{l=1}^{n} \frac{a_{il} a_{jl}}{d(u_l)} \tag{2}$$

where $1 \le i, j \le m$, $d(o_j) = \sum_{i=1}^{n} a_{ji}$ and $d(u_l) = \sum_{i=1}^{m} a_{il}$ denote the degree of object o_j (involving all users) and the degree of user u_l (involving all objects), respectively. $A = \{a_{il}\}$ is the $m \times n$ adjacent matrix of Equation 1 where

$$a_{il} = \begin{cases} 1, & object\ o_i\ is\ chosen\ by\ u_l \\ 0, & otherwise \end{cases} \tag{3}$$

and $1 \le i \le m$ and $1 \le l \le$ n. From Equation 2, the computational complexity for calculating each weight w_{ij} can be deduced as $O(n)$ and $O(m^2 \times n)$ for calculating all the weights. This complexity is very substantial and has an adverse impact for large object-user spaces. Appropriately partitioning the sparse user-object matrix in a way that can support parallel operations in determining the recommendations can greatly decrease the overall computational complexity and make this problem more tractable in practice. This is the objective of our work that can reach fruition by taking advantage of the sparsity of matrix.

3. BDM-NBI Bipartite-Graph Partitioning Algorithm

Our BDM-NBI algorithm starts with operations on the object-user matrix, applies bipartite graph partitioning, and involves two phases. BDM stands for Bordered block-Diagonal Matrix. In the first phase the matrix is partitioned into the double bordered block-diagonal (DB) form [8]. The second phase converts it into the SB form. Equation 4 shows the k-way DB form, where the missing values are all zeros except for the submatrices in the diagonal. Each row and column in the submatrices of matrices

$\mathrm{R} = (\mathrm{R}_1 \ldots \mathrm{R}_\mathrm{k}, \mathrm{D})$ and $\mathrm{C} = (\mathrm{C}_1^\mathrm{T} \ldots \mathrm{C}_1^\mathrm{T}, \mathrm{D}^\mathrm{T})^\mathrm{T}$ is a coupling row and column. Each coupling row has nonzero elements in the columns of at least two diagonal blocks. Each coupling column has nonzero elements in the rows of at least two diagonal blocks.

$$A = PA'Q = \begin{bmatrix} A_{11} & \cdots & A_{1k} \\ \vdots & \ddots & \vdots \\ A_{k1} & \cdots & A_{kk} \\ A_{(k+1)1} & \cdots & A_{(k+1)k} \end{bmatrix} = \begin{bmatrix} B_1 & & & C_1 \\ & \ddots & & \vdots \\ & & B_k & C_k \\ R_1 & \cdots & R_k & D \end{bmatrix} = A_{DB} \quad (4)$$

Details follow of the two-phase process that produces the successive transformations $A \rightarrow A_{DB}$ and $A_{DB} \rightarrow A_{SB}$.

Phase 1. *Bipartite graph transformation $A \rightarrow A_{DB}$.*

This phase finds a small separator to dissect a graph into components of approximately equal size. The key point is to generate a long level structure1 $\mathrm{L(r)}$ of the graph and then choose a small "middle" level separator. Let $\mathrm{G} = (\mathrm{X}, \mathrm{E})$ be a graph that represents the relationship of users and objects, where X and E are the sets of vertices and edges, respectively. This graph is created from matrix A using the following steps in Algorithm 1 [13].

Algorithm 1.

```
    Step 1. Set $\mathfrak{R} = X$ and $p = |X|$; p is the population in X.
    Step 2. Find a connected component G(C) (C is a cut set of nodes) in $G(\mathfrak{R})=(X,E)$ and construct a level structure[1] [13] of the component G(C) rooted at a pseudo-peripheral node[2] r, where $L(r) = \{L_0(r), L_1(r), \ldots, L_{l(r)}(r)\}$ and $l(r)$ is the length of the level structure.
    Step 3. If $l(r) \leq 2$, then set S = C and go to Step 4. Otherwise let $j = \lfloor (l(r)+1)/2 \rfloor$, and determine the set $S \subset L_{j(r)}$, where $S = \{y \in L_{j(r)} | Adj(y) \cap L_{j+1(r)} \neq \emptyset\}$. Adj(y) represents the set of nodes adjacent to y.
    Step 4. Number the nodes in the separator S starting with $p - |S| + 1$ and reaching p. Reset $\mathfrak{R} \leftarrow \mathfrak{R} - S$ and $p \leftarrow p - |S|$. If $\mathfrak{R} \neq \emptyset$, go to Step 2.
```

This algorithm transforms matrix A to the A_{DB} form of Equation 4. The number of coupling rows and columns in A_{DB} is equal to the number of row and column vertices, respectively, in separator S. The total number of coupling rows and columns in A_{DB} must be minimized. The algorithm's computational complexity is [13]:

$$O(\textstyle\sum_{x \in X} |Deg(x)|^2) \leq O(m \sum_{x \in X} |Deg(x)|) = O(m|E|) \quad (5)$$

where $|E|$ is the number of edges in *G*, *Deg(x)* is the degree of node x and m is the max node degree in graph *G(X,E)*.

A very important step in the above algorithm is choosing a good pseudo-peripheral node. The procedure to generate this node for graph $\mathrm{G} = (\mathrm{X}, \mathrm{E})$ is shown in Algorithm 2 that follows [13].

Algorithm 2.

```
    Step 1. Choose an arbitrary node r in X.
    Step 2. Construct the level structure rooted at r: $L(r) = \{L_0(r), L_1(r), \ldots, L_{l(r)}(r)\}$.
    Step 3. Choose a node x of minimum degree in $L_{l(r)}(r)$.
    Step 4. a) Construct the level structure rooted at x,
            b) If $l(x) > l(r)$, then set $r \leftarrow x$ and go to Step 3.
    Step 5. Choose node x as the pseudo-peripheral node.
```

[1] A general level structure is a partitioning $\mathrm{L} = \{L_0, L_1, \ldots, L_l\}$ where $\mathrm{Adj}(L_0) \subset L_1$, $\mathrm{Adj}(L_l) \subset L_{l-1}$ and $\mathrm{Adj}(L_i) \subset L_{i-1} \cup L_{i+1}$, $i = 2,3, \ldots, l-1$ [13].

[2] A node $\mathrm{x} \in \mathrm{X}$ is said to be a peripheral node if its eccentricity is equal to the graph's diameter.

Phase 2. *Column-splitting for* $A_{DB} \rightarrow A_{SB}$ *transformation.*

The second phase of the algorithm transforms the matrix from the DB into the SB form by considering the coupling columns in A_{DB}. Each column j is represented by multiple copies, with one copy for each block C_i that has at least one nonzero in column j. These multiple copies are used to decouple the corresponding C_i's. This splitting process for each such column j increases both the row and column dimensions of matrix A_{SB} [6].

With the NBI algorithm, we will discard the dummy rows after computing the weighted matrix W. The rows indicate the objects and the columns indicate the users. These dummy rows do not really exist to connect objects with users so the recommendation result is not affected by discarding these dummy rows. The NBI algorithm will run on B_1, …, B_k, R in parallel. So this phase decreases the storage requirements for the matrix and decomposes a high-dimensional sparse matrix into several lower-dimensional matrices. The algorithmic steps of this phase are shown in Algorithm 3 that follows.

Algorithm 3.

```
Step 1. Choose an arbitrary node $s \in S$, where S is the graph separator of Algorithm 1.
Step 2. Get all the coupling elements of node s.
Step 3. Count the number N of blocks corresponding to node s.
Step 4. Insert new columns into the corresponding blocks and duplicate the corresponding elements.
Step 5. Add N rows at the end of the matrix and mark the crossover point of these new rows and new columns.
Step 6. $S \leftarrow S - s$. If $S \neq \emptyset$, go to Step 1.
```

4. Experimental Results

Our experimental data sets are MovieLens [14] and Netflix [15]. Movielens has three different sizes of data sets. The smallest data set contains 100,000 ratings of 1,682 movies and 943 users, and users rate movies on a scale from 1 to 5 (integral) stars. A movie has been collected by a user if and only if the giving rating is at least 3. The original data contains 105 ratings, 85.25% of which are no less than 3. The data contains 85250 object-user pairs. The biggest data set contains 10,000,054 anonymous ratings of 10,681 movies made by 71,567 MovieLens users. We just retain the data with a grade of not less than 3. The Netflix training data set contains 100,480,507 ratings that 480,189 users gave to 17,770 movies. Each training

Table 1. Properties of Test Matrices

Name	Rows	Columns	Nonzero	Sparseness
MLens-100K	1574	943	82520	0.055596
MLens-10MB	10598	69863	8242124	0.011132
MLens-10MS1	10561	69859	6593672	0.008937
MLens-10MS2	10546	63769	6000000	0.008922
Netflix-B	17770	479760	85730437	0.010056
Netflix-S1	17770	479244	68580829	0.008053
Netflix-S2	15643	478680	60000000	0.008013

rating is a quadruplet of the form <user, movie, date of grade, grade>. The user and movie fields are integer IDs, while grades are from 1 to 5 (integral) stars. We delete data with a grade less than 3. All the experiments are based on three data sets. Our experiments were performed with Hadoop on 6 computers with Intel i7 and 16GB of memory. One name node and one standby name node and 4 data nodes.

Table 1 shows the details for these data sets. MLens-10MS1 and MLens-10MS2 are sampled from MLens-10MB. Netflix-S1 and Netflix-S2 are sampled from Netflix-B. The MLens-100K sparsity is the highest in the group because this data set contains information when users grade movies with a number no less than 20. This data set's maximum grade is 509.

Table 2 presents the results of our experiments with bipartite graph partitioning for both the $A \rightarrow A_{DB}$ and $A_{DB} \rightarrow A_{SB}$ transformations. K is a parameter used by the bipartite graph partitioning algorithm to decide how many blocks to produce. $\%C_R$ and $\%C_C$ increase when K increases, where $\%C_R = C_R/\text{Rows}$, $\%C_C = \frac{C_C}{\text{Columns}}$, and C_R and C_C are the

Table 2. Matrix Partitioning Results

Name	K	A_{DB}		A_{SB}
		$\%C_R$	$\%C_C$	$\%C_R$
MLens-100K	2	18.16	3.21	18.89
	4	22.68	3.64	22.73
	8	25.71	4.11	25.12
	16	28.27	4.67	28.31
MLens-10MB	2	9.47	0.65	9.64
	4	12.88	1.03	13.15
	8	15.92	1.46	16.23
	16	18.19	1.94	18.98
MLens-10MS1	2	8.23	0.51	8.47
	4	11.36	0.87	11.72
	8	14.43	1.26	14.87
	16	17.51	1.64	17.92
MLens-10MS2	2	8.21	0.51	8.45
	4	11.29	0.82	11.66
	8	14.38	1.23	14.80
	16	17.48	1.61	17.89
Netflix-B	2	9.44	0.66	9.61
	4	12.65	0.94	13.07
	8	15.89	1.44	16.20
	16	18.02	1.90	18.83
Netflix-S1	2	8.11	0.47	8.41
	4	11.16	0.79	11.64
	8	14.33	1.17	14.76
	16	17.44	1.59	17.86
Netflix-S2	2	8.12	0.47	8.43
	4	11.15	0.80	11.60
	8	14.35	1.17	14.68
	16	17.39	1.60	17.82

numbers of coupling rows and coupling columns, respectively. It is true that $\%C_C < \%C_R$. Among all the data sets, obviously the coupling rows and columns in MLens-100K are larger. Also, due to its very sparse for the $A_{DB} \rightarrow A_{SB}$ transformation the algorithm needs to do much more column-splitting than for the other data sets. This operation reduces the accuracy of the recommendation.

5. BDM-NBI Algorithm

For the original NBI algorithm to run on the Hadoop platform for processing massive data sets in parallel, we combine the bipartite-graph partitioning and NBI algorithms. Firstly, the bipartite-graph partitioning algorithm partitions the massive data into blocks of comparable size. Then these data blocks are distributed for concurrent processing.

5.1. Experimental Results

Our experiments include two phases. The first phase tests the accuracy of the algorithm. We compare differences in the recommendation results generated by the original NBI algorithm and our singly bordered block-diagonal matrix NBI algorithm (BDM-NBI). We restrict the dimensionality of the matrix to no more than 4,000 since we test the NBI algorithm on the whole data set. This limit guarantees that the NBI algorithm can finish in a reasonable execution time. If the matrix dimensionality is too big, the algorithm runs a very long time. The second phase tests the BDM-NBI algorithm on massive data sets. The accuracy and diversity of the recommendations are evaluated as per [5].

Table 3. Features of the Data Set

Name	Rows	Columns	Nonzero	Sparsity
MLens-100K	1574	943	82520	0.055596
MLens-10MS3	1730	1795	4000	0.001288
MLens-10MS4	1622	1509	3500	0.001430
MLens-10MS5	3966	2707	9000	0.000838
Netflix-S5	3938	119	4000	0.008536

For the sampling data sets MLens-10MS3, MLens-10MS4, MLens-10MS5 and Netflix-S5, the numbers of rows and columns are the numbers of movies and users, respectively. Since these five data sets are rather small, the NBI algorithm runs in a reasonable time.

Table 4. Matrices Partition Results

Name	K	A_{DB}		A_{SB}
		$\%C_R$	$\%C_C$	$\%C_R$
MLens-100K	2	18.16	3.21	18.89
MLens-10MS3	2	6.78	0.29	6.99
MLens-10MS4	2	6.56	0.28	6.83
MLens-10MS5	4	9.21	0.59	9.35
Netflix-S5	4	11.15	0.79	11.62

We partition the first three data sets into two equal parts and the last two data sets into four equal parts. Then the BDM-NBI algorithm processes them concurrently. We compare the different recommendations by Ranking[3], Precision[4] and the Hamming distance. Excluding MLens-100K, the results for the other data sets are quite similar for NBI and BDM-NBI. The experiments show that the BDM-NBI algorithm produces good approximation when the data set is very sparse. If the data set is not very sparse, the results of the two algorithms differ substantially. This means that the BDM-NBI algorithm is not suitable for data sets are not very sparse. From Table 4, we can see that the results for MLens-100K are very different than those for MLens-10MS3 and MLens-10MS4. Fortunately, in most cases the data sets assumed for recommendation systems are very sparse. In relation to the Precision metric, L=10 and L=50 is the length of the top-L recommendation list. The bigger the value of L, the lower the precision is.

Table 5. Recommendation Results for *K*=2, 4

Name	Ranking		Precision				Hamming	
	NBI	**BDM-NBI**	**NBI**		**BDM-NBI**		NBI	BDM-NBI
			L=10	**L=50**	**L=10**	**L=50**		
MLens-100K	0.104	0.221	0.162	0.073	0.138	0.044	0.628	0.347
MLens-10MS3	0.142	0.144	0.147	0.053	0.145	0.051	0.396	0.391
MLens-10MS4	0.140	0.142	0.148	0.054	0.146	0.051	0.402	0.394
MLens-10MS5	0.136	0.139	0.150	0.056	0.147	0.053	0.403	0.392
Netflix-S5	0.134	0.137	0.132	0.057	0.126	0.046	0.406	0.393

Then we test the BDM-NBI algorithm on the massive data sets of Table 1. Except for MLens-100K, the numbers of movies in the other data sets are more than 10,000. We

[3] For an arbitrary user u_i, if the edge $u_i - o_j$ is in the test set, the position of o_j is found in the ordered queue. Test entries are collected by users so a good algorithm normally provides a high recommendation to them; this produces a small Ranking value [13].

[4] For an arbitrary target user u_i, the precision $P_i(L)$, of u_i is the ratio of the number of u_i's removed links, $R_i(L)$, contained in the top-L recommendations to L, say $P_i(L) = R_i(L)/L$. $\mathrm{P(L)} = \frac{1}{m}\sum_{i=1}^{m} P_i(L)$ [13].

discard MLens-100K in this experiment. The parameter K of the BDM-NBI algorithm is set to 16. Choosing wisely this parameter can guarantee that the algorithm will finish in reasonable time. Because the dimensionality of the matrices is very high, we cannot compare directly the results of NBI and BDM-NBI. The results are quite similar to those in Table 5. This experiment shows that the BDM-NBI algorithm can ensure credible recommendation results on big data sets which are very sparse.

Table 6. Recommendation Results for *K*=16

Name	Ranking	Precision		Hamming
		L=10	L=50	
MLens-10MS1	0.124	0.158	0.064	0.557
MLens-10MS2	0.128	0.157	0.062	0.553
Netflix-S1	0.142	0.142	0.053	0.544
Netflix-S2	0.147	0.142	0.052	0.546

6. Conclusions

Our experiments show that the BDM-NBI algorithm produces quality recommendation results. Except for the MLens-100K data set, we can compare from Table 5 the experimental results of these two algorithms. They produce very similar results. From Equation 2, we can evaluate the computation complexity of the NBI and BDM-NBI algorithms. The computational complexities of NBI and BDM-NBI are $O(m * m * n)$ and $O(\sum_{i=1}^{r}(m_i * m_i * n_i))$, respectively, where r is the number of matrix blocks, and m_i and n_i are the dimensions of block i. On the Hadoop platform with the multicore processor, if the dimensionality of the matrix is about 1,500, the processing time of the NBI algorithm is about 35 minutes. If the sampling data set's dimensionality is about 4,000, the processing time is about seven hours. But even for Netflix-B with 4.79 million users, the processing time of the BDM-NBI algorithm is about just six hours. So we can conclude that the BDM-NBI algorithm can analyze large data sets in reasonable time while producing reliable recommendation results.

Acknowledgments

This work has been supported by NSFC Projects (No. 2013CB834205).

References

[1] U. Hanani, B. Shapira and P. Shoval, "Information Filtering: Overview of Issues", Research and Systems. User Modeling and User Adapted Interaction, vol. 11, no. 3, **(2001)**, pp. 203–259.
[2] M. J. Pazzani, "A Framework for Collaborative", Content-Based and Demographic Filtering. Artificial Intelligence Review, vol. 13, no. 5-, **(1999)**6, pp. 393-408.
[3] J. L. Herlocker, J. A. Konstan, L. G. Terveen and J. T. Riedl, "Evaluating Collaborative Filtering Recommender Systems", ACM Trans. Information Systems, vol. 22, no. 1, **(2004)**, pp. 5–53.
[4] T. Zhou, L. L. Jiang, R. Q. Su and Y. C. Zhang, "Effect of Initial Configuration on Network-based Recommendation", Physics and Society, vol. 81, no. 58004, **(2008)**.
[5] T. Zhou, R. Q. Su, R. R. Liu, L. L. Jiang, B. H. Wang and Y. C. Zhang, "Ultra Accurate Personal Recommendation via Eliminating Redundant Correlations", Data Analysis, Statistics and Probability, **(2009)**.
[6] M. C. Ferris and J. D. Horn, "Partitioning Mathematical Programs for Parallel Solution", Mathematical Programming, no. 80, **(1998)**, pp. 35–61.
[7] C. Aykanat, A. Pinar and U. V. Çatalyürek, "Permuting Sparse Rectangular Matrices into Block-Diagonal Form", SIAM Journal on Scientific Computing, vol. 25, no. 6, **(2002)**, pp. 1860–1879.
[8] X. Wang and S. G. Ziavras, "Parallel Solution of Newton's Power Flow Equations on Configurable Chips", Electrical Power and Energy Systems, vol. 29, no. 5, **(2007)**, pp. 422-431.
[9] J. L. Herlocker, J. A. Konstan, L. G. Terveen and J. T. Riedl, "Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems, vol. 22, **(2004)**, pp. 5-53.
[10] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Analysis of Recommendation Algorithms for e-Commerce", 2nd ACM Conference on Electronic Commerce, **(2000)**, pp. 158-167.
[11] T. Rashid and T. A. Davis, "An Approach for Parallelizing Any General Unsymmetric Sparse Matrix

Algorithm", 7th SIAM Conference on Parallel Processing for Scientific Computing, **(1995)**, pp. 413–417.

[12] A. Shepitsen, J. Gemmell, B. Mobasher and R. Burke, "Personalized Recommendation in Social Tagging Systems Using Hierarchical Clustering", ACM Conference on Recommender Systems, **(2008)**, pp. 259 – 266.

[13] G. Alan, L. Joseph and N. Esmond, "Computer Solution of Sparse Linear Systems", **(1994)**, http://web.engr.illinois.edu/~heath/courses/cs598mh/george_liu.pdf.

[14] http://www.grouplens.org/

[15] http://www.netflixprize.com/

Authors

Hao Huang is a Lecturer in School of Information Technology & Management at University of International Business and Economics. He received his PhD from School of Computer Science & Technology at Beijing Institute of Technology in 2006. His research interests are machine learning, data mining and personalization recommendation.

Sotirios G. Ziavras is the Associate Provost for Graduate Studies at NJIT, a Professor of ECE and the Director of CAPPL (Computer Architecture and Parallel Processing Laboratory). He received the Diploma in EE from the National Technical University of Athens, Greece and the D.Sc. in Computer Science from George Washington University. His main research interests are multicore processors, reconfigurable computing, accelerators, parallel processing and embedded computing.

Yaojie Lu received the B.S. in Physics from Fudan University, China and M.Sc in Electrical Engineering from NJIT. He is currently a Ph.D candidate and teaching assistant at ECE department of NJIT, and works in the CAPPL (Computer Architecture and Parallel Processing Laboratory). His main research interests are parallel and distributed computer architectures, vector processing and embedded computing.