Chapter 1
Bounded Matrix Low Rank Approximation

Ramakrishnan Kannan, Mariya Ishteva, Barry Drake, and Haesun Park

Abstract Low Rank Approximation is the problem of finding two matrices $\mathbf{P} \in \mathbb{R}^{m \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times n}$ for input matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$, such that $\mathbf{R} \approx \mathbf{PQ}$. It is common in recommender systems rating matrix, where the input matrix $\mathbf{R}$ is bounded in between $[r_{min}, r_{max}]$ such as $[1, 5]$. In this chapter, we propose a new improved scalable low rank approximation algorithm for such bounded matrices called Bounded Matrix Low Rank Approximation(BMA) that bounds every element of the approximation $\mathbf{PQ}$. We also present an alternate formulation to bound existing recommender system algorithms called BALS and discuss its convergence. Our experiments on real world datasets illustrate that the proposed method BMA outperforms the state of the art algorithms for recommender system such as Stochastic Gradient Descent, Alternating Least Squares with regularization, SVD++ and Bias-SVD on real world data sets such as Jester, Movielens, Book crossing, Online dating and Netflix.

1.1 Introduction

Matrix low rank approximation of a matrix $\mathbf{R}$ finds matrices $\mathbf{P} \in \mathbb{R}^{n \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times m}$ such that $\mathbf{R}$ is well approximated by $\mathbf{PQ}$ i.e., $\mathbf{R} \approx \mathbf{PQ} \in \mathbb{R}^{n \times m}$,

Ramakrishnan Kannan
Georgia Institute of Technology, USA, e-mail: rkannan@gatech.edu

Mariya Ishteva
Vrije Universiteit Brussel (VUB), Belgium, e-mail: mariya.ishteva@vub.ac.be

Barry Drake
Georgia Tech Research Institute, USA, e-mail: Barry.Drake@gtri.gatech.edu

Haesun Park
Georgia Institute of Technology, USA, e-mail: hpark@cc.gatech.edu

where $k < rank(\mathbf{R})$. Low rank approximations vary depending on the constraints imposed on the factors as well as the measure for the difference between $\mathbf{R}$ and $\mathbf{PQ}$. Low rank approximations have produced a huge amount of interest in the data mining and machine learning communities due to its effectiveness for addressing many foundational challenges in these application areas. A few prominent techniques of machine learning that use low rank approximation are principal component analysis, factor analysis, latent semantic analysis, and non-negative matrix factorization (NMF), to name a few.

One of the most important low rank approximations is based on the singular value decompositions (SVD) [9]. Low rank approximation using SVD has many applications over a wide spectrum of disciplines. For example, an image can be compressed by taking the low row rank approximation of its matrix representation using SVD. Similarly, in text data – latent semantic indexing, is a dimensionality reduction technique of a term-document matrix using SVD [6]. The other applications include event detection in streaming data, visualization of a document corpus and many more.

Over the last decade, NMF has emerged as another important low rank approximation technique, where the low-rank factor matrices are constrained to have only non-negative elements. It has received enormous attention and has been successfully applied to a broad range of important problems in areas including, but not limited to, text mining, computer vision, community detection in social networks, visualization, and bioinformatics. [21][14].

In this chapter, we propose a new type of low rank approximation where the elements of the approximation are non-negative, or more generally, bounded – that is, its elements are within a given range. We call this new low rank approximation Bounded Matrix Low Rank Approximation (BMA). BMA is different from NMF in that it imposes both upper and lower bounds *on its product* $\mathbf{PQ}$ rather than non-negativity in each of the low rank factors $\mathbf{P} \geq 0$ and $\mathbf{Q} \geq 0$. Thus, the goal is to obtain a lower rank approximation $\mathbf{PQ}$ of a given input matrix $\mathbf{R}$, where the elements of $\mathbf{PQ}$ and $\mathbf{R}$ are bounded.

Let us consider a numerical example to appreciate the difference between NMF and BMA. Consider the 4x6 matrix with all entries between 1 and 10. In the Figure 1.1, the output low rank approximation is shown between BMA and NMF by running only one iteration for low rank 3. It is important to observe the following.

- All the entries in the BMA are bounded between 1 and 10, whereas, approximation generated out of NMF is not bounded in the same range of input matrix. This difference is hugely pronounced in the case of very large input matrix and the current practice is when an entry in the low rank approximation is beyond the bounds, it is artificially truncated.
- In the case of BMA, as opposed to NMF, every individual low rank factors are unconstrained and takes even negative values.

Also, from this example, it is easy to understand that for certain applications, enforcing constraints on the product of the low rank factors like BMA result in a better approximation error than constraints on individual low rank factors – like NMF.

$$\mathbf{R} = \begin{pmatrix} 4 & 2 & 6 & 6 & 1 & 6 \\ 9 & 5 & 9 & 8 & 2 & 9 \\ 2 & 9 & 1 & 6 & 4 & 1 \\ 10 & 8 & 10 & 2 & 9 & 1 \end{pmatrix}$$

Input Bounded Matrix $\mathbf{R} \in [1, 10]$

$$\mathbf{R'}_{BMA} = \begin{pmatrix} 4.832 & 2.233 & 5.118 & 5.824 & 1.000 & 5.878 \\ 8.719 & 4.839 & 9.092 & 8.104 & 2.490 & 9.012 \\ 1.983 & 9.017 & 1.661 & 5.957 & 3.861 & 1.000 \\ 10.000 & 7.896 & 10.000 & 2.106 & 4.871 & 5.599 \end{pmatrix}$$

BMA Output $\mathbf{R}_{BMA}$. The error $\|\mathbf{R} - \mathbf{R'}_{BMA}\|_F^2$ is 40.621.

$$\mathbf{R'}_{NMF} = \begin{pmatrix} 4.939 & 3.378 & 5.312 & 5.021 & 3.131 & 4.736 \\ 8.066 & 5.794 & 8.696 & 8.495 & 4.649 & 8.036 \\ 5.850 & 4.600 & 6.241 & 4.548 & 4.036 & 4.343 \\ 9.693 & 7.549 & \mathbf{10.225} & 5.165 & 8.303 & 4.958 \end{pmatrix}$$

NMF Output $\mathbf{R}_{NMF}$. The error $\|\mathbf{R} - \mathbf{R'}_{NMF}\|_F^2$ is 121.59.

$$\begin{pmatrix} 4.138 & -0.002 & 7.064 \\ 7.037 & -0.008 & 9.429 \\ 3.730 & 0.031 & -4.844 \\ 6.776 & -0.020 & 1.000 \end{pmatrix} \qquad \begin{pmatrix} 1.214 & 1.384 & 1.202 & 0.753 & 0.734 & 0.768 \\ -90.835 & 49.650 & -92.455 & 170.147 & -9.311 & -0.525 \\ -0.058 & -0.478 & -0.011 & 0.441 & -0.292 & 0.382 \end{pmatrix}$$

BMA's Left and Right Low Rank Factors $\mathbf{P}_{BMA}$ and $\mathbf{Q}_{BMA}$

$$\begin{pmatrix} 4.021 & 2.806 & 0.153 \\ 8.349 & 4.179 & 0.000 \\ 7.249 & 1.216 & 0.000 \\ 10.015 & 0.725 & 0.467 \end{pmatrix} \qquad \begin{pmatrix} 0.727 & 0.605 & 0.770 & 0.431 & 0.557 & 0.416 \\ 0.478 & 0.178 & 0.543 & 1.172 & 0.000 & 1.092 \\ 4.421 & 2.920 & 4.538 & 0.000 & 5.835 & 0.000 \end{pmatrix}$$

NMF's Non-negative Left and Right Low Rank Factors $\mathbf{P}_{NMF}$ and $\mathbf{Q}_{NMF}$

Fig. 1.1 Numerical Motivation for BMA

In order to address the problem of an input matrix with missing elements, we will formulate a BMA that imposes bounds on a low rank matrix that is the best approximate for such matrices. The algorithm design considerations are – (1) Simple implementation (2) Scalable to large data and (3) Easy parameter tuning with no hyper parameters.

Formally, the BMA problem for an input matrix $\mathbf{R}$ is defined as

$$\begin{aligned} \min_{\mathbf{P},\mathbf{Q}} \quad & \|\mathbf{M} \cdot *(\mathbf{R} - \mathbf{PQ})\|_F^2 \\ \text{subject to} \quad & \\ & r_{min} \leq \mathbf{PQ} \leq r_{max}, \end{aligned} \tag{1.1}$$

where r_{min} and r_{max} are the bounds and $\|\cdot\|_F$ stands for the Frobenius norm. In the case of an input matrix with missing elements, the low rank matrix is approximated only against the known elements of the input matrix. Hence, during error computation the filter matrix $\mathbf{M}$, includes only the corresponding elements of the low rank $\mathbf{PQ}$ for which the values are known. Thus, $\mathbf{M}$ has '1' everywhere for input matrix $\mathbf{R}$ with all known elements. However, in the case of a recommender system, the matrix $\mathbf{M}$ has zero for each of the missing elements of $\mathbf{R}$. In fact, for recommender systems, typically only 1 or 2% of all matrix elements are known.

It should be pointed out that an important application for the above formulation is recommender systems, where the community refers to it as a matrix factorization. The unconstrained version of the above formulation (1.1), was first solved using Stochastic Gradient Descent (SGD) [7] and Alternating Least Squares with Regularization (ALSWR) [31]. However, we have observed that previous research has not leveraged the fact that all the ratings $r_{ij} \in \mathbf{R}$ are bounded within $[r_{min}, r_{max}]$. All existing algorithms *artificially truncate* their final solution to fit within the bounds.

Recently, there has been many innovations introduced into the the naive low rank approximation technique such as considering only neighboring entries during the factorization process, time of the ratings, and implicit ratings such as "user watched but did not rate". Hence, it is important to design a bounding framework that seamlessly integrates into the existing sophisticated recommender systems algorithms.

Let $f(\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q})$ be an existing recommender system algorithm that can predict all the (u, i) ratings, where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_l\}$ is the set of parameters apart from the low rank factors $\mathbf{P}, \mathbf{Q}$. For example,., in the recommender system context, certain implicit signals are combined with the explicit ratings such as user watched a movie till the end but didn't rate it. We have learn weights for such implicit signals to predict a user's rating. Such weights are represented as parameter $\boldsymbol{\Theta}$. For simplicity, we are slightly abusing the notation here. The $f(\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q})$ either represents estimating a particular value of (u, i) pair or it represents the complete estimated low rank k matrix $\hat{\mathbf{R}} \in \mathbb{R}^{n \times m}$. The ratings from such recommender system algorithms can be scientifically bounded by the following optimization problem based on low rank approximation to determine the unknown ratings.

$$\begin{aligned} \min_{\boldsymbol{\Theta},\mathbf{P},\mathbf{Q}} \quad & \|\mathbf{M} \cdot *(\mathbf{R} - f(\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}))\|_F^2 \\ \text{subject to} \quad & \\ & r_{min} \leq f(\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}) \leq r_{max}. \end{aligned} \tag{1.2}$$

Traditionally, regularization is used to control the low rank factors $\mathbf{P}$ and $\mathbf{Q}$ from taking larger values. However, this does not guarantee that the value of the product $\mathbf{PQ}$ is in the given range. We also experimentally show that introducing the bounds on the product of $\mathbf{PQ}$ outperforms the low rank approximation algorithms with regularization.

In this chapter, we present a survey of current state-of-the-art and foundational material. An explanation of the Block Coordinate Descent (BCD) framework [2] that was used to solve the NMF problem and how it can be extended to solve the Problems (1.1) and (1.2) will be presented. Also, described in this chapter are implementable algorithms and scalable techniques for solving large scale problems in multi core system with low memory. Finally, we present substantial experimental results illustrating that the proposed methods outperform the state-of-the-art algorithms for recommender systems such as Stochastic Gradient Descent, Alternating Least Squares with regularization, SVD++, Bias-SVD on real world data sets such as Jester, Movielens, Book crossing, Online dating and Netflix. This chapter is based primarily on our earlier work [12, 13]. Notations that are consistent with those in the machine learning literature are used throughout this chapter. A lowercase/uppercase letter such as x or X, is used to denote a scalar; a boldface lowercase letter, such as $\mathbf{x}$, is used to denote a vector; a boldface uppercase letter, such as $\mathbf{X}$, is used to denote a matrix. Indices typically start from 1. When a matrix $\mathbf{X}$ is given, $\mathbf{x}_i$ denotes its i^{th} column, $\mathbf{x}_j^\intercal$ denotes its j^{th} row and x_{ij} or $X(i,j)$ denote its $(i,j)^{th}$ element. For a vector $\mathbf{i}$, $\mathbf{x}(\mathbf{i})$ means that vector $\mathbf{i}$ indexes into the elements of vector $\mathbf{x}$. That is, for $\mathbf{x} = [1, 4, 7, 8, 10]$ and $\mathbf{i} = [1, 3, 5]$, $\mathbf{x}(\mathbf{i}) = [1, 7, 10]$. We have also borrowed certain notations from matrix manipulation scripts such as Matlab/Octave. For example, the $max(\mathbf{x})$ is the maximal element $x \in \mathbf{x}$ and $max(\mathbf{X})$ is a vector of maximal elements from each column $\mathbf{x} \in \mathbf{X}$.

For the reader's convenience, the notations used in this chapter are summarized in Table 1.1.

1.2 Related Work

This section introduces the BMA and its application to recommender systems and reviews some of the prior research in this area. Following this section is a brief overview of our contributions.

The important milestones in matrix factorization for recommender systems have been achieved due to the Netflix competition (http://www.netflixprize.com/) where the winners were awarded 1 million US Dollars as grand prize.

Funk [7] first proposed matrix factorization for recommender system based on SVD, commonly called the Stochastic Gradient Descent (SGD) algorithm. Paterek [27] improved SGD by combining matrix factorization with baseline estimates. Koren, a member of the winning team of the Netflix prize, im-

$\mathbf{R} \in \mathbb{R}^{n \times m}$	Ratings matrix. The missing ratings are indicated by 0, and the given ratings are bounded within $[r_{min}, r_{max}]$.
$\mathbf{M} \in \{0,1\}^{n \times m}$	Indicator matrix. The positions of the missing ratings are indicated by 0, and the positions of the given ratings are indicated by 1.
n	Number of users
m	Number of items
k	Value of the reduced rank
$\mathbf{P} \in \mathbb{R}^{n \times k}$	User-feature matrix. Also called as a low rank factor.
$\mathbf{Q} \in \mathbb{R}^{k \times m}$	Feature-item matrix. Also called as a low rank factor.
$\mathbf{p}_x \in \mathbb{R}^{n \times 1}$	x-th column vector of $\mathbf{P} = [\mathbf{p_1}, \cdots, \mathbf{p_k}]$
$\mathbf{q}_x^\intercal \in \mathbb{R}^{1 \times m}$	x-th row vector of $\mathbf{Q} = [\mathbf{q_1}, \cdots, \mathbf{q_k}]^\intercal$
$r_{max} > 1$	Maximal rating/upper bound
r_{min}	Minimal rating/lower bound
u	A user
i	An item
$\cdot*$	Element wise matrix multiplication
$\cdot/$	Element wise matrix division
$\mathbf{A}(:, i)$	$i-th$ column of the matrix $\mathbf{A}$
$\mathbf{A}(i, :)$	$i-th$ row of the matrix $\mathbf{A}$
β	Data structure in memory factor
$memsize(v)$	The approximate memory of a variable v (the product of the number of elements in v, the size of each element, and β)
μ	Mean of all known ratings in $\mathbf{R}$
$\mathbf{g} \in \mathbb{R}^n$	Bias of all users u
$\mathbf{h} \in \mathbb{R}^m$	Bias of all items i

Table 1.1 Notations

proved the results with his remarkable contributions in this area. Koren [18] proposed a baseline estimate based on mean rating, user–movie bias, combined with matrix factorization and called it Bias-SVD. In SVD++ [18], he extended this Bias-SVD with implicit ratings and considered only the relevant neighborhood items during matrix factorization. The Netflix dataset also provided the time of rating. However most of the techniques did not include time in their model. Koren [19] proposed time-svd++, where he extended his previous SVD++ model to include the time information. So far, all matrix factorization techniques discussed here are based on SVD and used gradient descent to solve the problem. Alternatively, Zhou et al. [31] used alternating least squares with regularization (ALSWR). Apart from these directions, there had been other approaches such as Bayesian tensor factorization [29], Bayesian probabilistic modelling [28], graphical modelling of the recommender system problem [25] and weighted low-rank approximation with zero weights for the missing values [26]. One of the recent works by Yu et al. [30] also uses coordinate descent to matrix factorization for recommender system. However, they study the tuning of coordinate descent optimization techniques for a parallel scalable implementation of matrix factorization for recommender system. A detailed survey and overview of matrix factorization for recommender systems is given in [20].

1.2.1 Our Contributions

Given the above background, we highlight our contributions. We propose a novel matrix factorization called Bounded Matrix Low Rank Approximation (BMA) which imposes a lower and an upper bound for the estimated values of the missing elements in the given matrix. We solve the BMA using block coordinate descent method. From this perspective, this is the first work that uses the block coordinate descent method and experiment BMA for recommender systems. We present the details of the algorithm with supporting technical details and a scalable version of the naive algorithm. It is also important to study imposing bounds for existing recommender systems algorithms. We also propose a novel framework for Bounding existing ALS algorithms (called BALS). Also, we test our BMA algorithm, BALS framework on real world datasets and compare against state of the art algorithms SGD, SVD++, ALSWR and Bias-SVD.

1.3 Foundations

In the case of low rank approximation using NMF, the low rank factor matrices are constrained to have only non-negative elements. However, in the case of BMA, we constrain the elements of their product with an upper and lower bound rather than each of the two low rank factor matrices. In this section, we explain our BCD framework for NMF and subsequently explain using BCD to solve BMA.

1.3.1 NMF and Block Coordinate Descent

Consider a constrained non-linear optimization problem as follows:

$$\min f(x) \text{ subject to } x \in \mathcal{X}, \tag{1.3}$$

where $\mathcal{X}$ is a closed convex subset of $\mathbb{R}^n$. An important assumption to be exploited in the BCD method is that the set $\mathcal{X}$ is represented by a Cartesian product:

$$\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_l, \tag{1.4}$$

where $\mathcal{X}_j$, $j = 1, \cdots, l$, is a closed convex subset of $\mathbb{R}^{N_j}$, satisfying $n = \sum_{j=1}^{l} N_j$. Accordingly, vector $\mathbf{x}$ is partitioned as $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_l)$ so that $\mathbf{x}_j \in \mathcal{X}_j$ for $j = 1, \cdots, l$. The BCD method solves for $\mathbf{x}_j$ fixing all other subvectors of $\mathbf{x}$ in a cyclic manner. That is, if $\mathbf{x}^{(i)} = (\mathbf{x}_1^{(i)}, \cdots, \mathbf{x}_l^{(i)})$ is given as the current iterate at the i^{th} step, the algorithm generates the next iterate

$\mathbf{x}^{(i+1)} = (\mathbf{x}_1^{(i+1)}, \cdots, \mathbf{x}_l^{(i+1)})$ block by block, according to the solution of the following subproblem:

$$\mathbf{x}_j^{(i+1)} \leftarrow \underset{\xi \in \mathcal{X}_j}{\operatorname{argmin}} f(\mathbf{x}_1^{(i+1)}, \cdots, \mathbf{x}_{j-1}^{(i+1)}, \xi, \mathbf{x}_{j+1}^{(i)}, \cdots, \mathbf{x}_l^{(i)}). \tag{1.5}$$

Also known as a *non-linear Gauss-Seidel* method [2], this algorithm updates one block each time, always using the most recently updated values of other blocks $\mathbf{x}_{\tilde{j}}, \tilde{j} \neq j$. This is important since it ensures that after each update the objective function value does not increase. For a sequence $\left\{\mathbf{x}^{(i)}\right\}$ where each $\mathbf{x}^{(i)}$ is generated by the BCD method, the following property holds.

Theorem 1. *Suppose f is continuously differentiable in $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_l$, where $\mathcal{X}_j$, $j = 1, \cdots, l$, are closed convex sets. Furthermore, suppose that for all j and i, the minimum of*

$$\min_{\xi \in \mathcal{X}_j} f(\mathbf{x}_1^{(i+1)}, \cdots, \mathbf{x}_{j-1}^{(i+1)}, \xi, \mathbf{x}_{j+1}^{(i)}, \cdots, \mathbf{x}_l^{(i)})$$

is uniquely attained. Let $\left\{\mathbf{x}^{(i)}\right\}$ be the sequence generated by the block coordinate descent method as in Eq. (1.5)*. Then, every limit point of $\left\{\mathbf{x}^{(i)}\right\}$ is a stationary point. The uniqueness of the minimum is not required when $l = 2$ [10].*

The proof of this theorem for an arbitrary number of blocks is shown in Bertsekas [2]. For a non-convex optimization problem, often we can expect the stationarity of a limit point [23] from a good algorithm.

When applying the BCD method to a constrained non-linear programming problem, it is critical to wisely choose a partition of $\mathcal{X}$, whose Cartesian product constitutes $\mathcal{X}$. An important criterion is whether the subproblems in Eq. (1.5) are efficiently solvable: for example, if the solutions of subproblems appear in closed form, each update can be computed fast. In addition, it is worth checking how the solutions of subproblems depend on each other. The BCD method requires that the most recent values be used for each subproblem in Eq. (1.5). When the solutions of subproblems depend on each other, they have to be computed sequentially to make use of the most recent values; if solutions for some blocks are independent from each other, however, they can be computed simultaneously. We discuss how different choices of partitions lead to different NMF algorithms. Three cases of partitions are discussed below.

1.3.1.1 BCD with Two Matrix Blocks - ANLS Method

For convenience, we first assume all the elements of the input matrix are known and hence we ignore $\mathbf{M}$ from the discussion. The most natural partitioning of the variables is to have two big blocks, $\mathbf{P}$ and $\mathbf{Q}$. In this case,

following the BCD method in Eq. (1.5), we take turns solving

$$\mathbf{P} \leftarrow \arg\min_{\mathbf{P}\geq 0} f(\mathbf{P},\mathbf{Q}) \;\text{ and }\; \mathbf{Q} \leftarrow \arg\min_{\mathbf{Q}\geq 0} f(\mathbf{P},\mathbf{Q}). \tag{1.6}$$

Since the subproblems are non-negativity constrained least squares (NLS) problems, the two-block BCD method has been called the alternating non-negative least square (ANLS) framework [23, 15, 17].

1.3.1.2 BCD with 2k Vector Blocks - HALS/RRI Method

Let us now partition the unknowns into 2k blocks in which each block is a column of $\mathbf{P}$ or a row of $\mathbf{Q}$, as explained in Figure 1.2. In this case, it is easier to consider the objective function in the following form:

$$f(\mathbf{p}_1,\cdots,\mathbf{p}_k,\mathbf{q}_1^\intercal,\cdots,\mathbf{q}_k^\intercal) = \|\mathbf{R} - \sum_{j=1}^{k} \mathbf{p}_j\mathbf{q}_j^T\|_F^2, \tag{1.7}$$

where $\mathbf{P} = [\mathbf{p}_1,\cdots\mathbf{p}_k] \in \mathbb{R}_+^{n\times k}$ and $\mathbf{Q} = [\mathbf{q}_1,\cdots,\mathbf{q}_k]^\intercal \in \mathbb{R}_+^{k\times m}$. The form in Eq. (1.7) represents that $\mathbf{R}$ is approximated by the sum of k rank-one matrices.

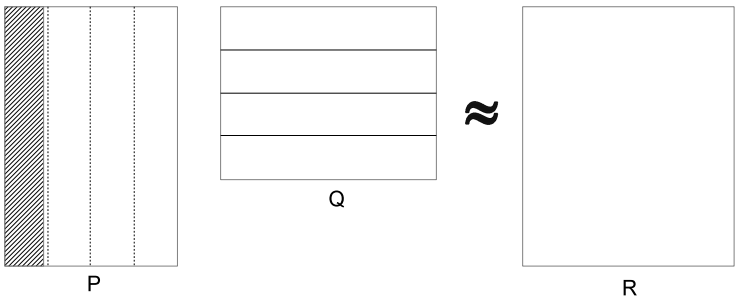

Fig. 1.2 BCD with 2k Vector Blocks

Following the BCD scheme, we can minimize f by iteratively solving

$$\mathbf{p}_i \leftarrow \arg\min_{\mathbf{p}_i\geq 0} f(\mathbf{p}_1,\cdots,\mathbf{p}_k,\mathbf{q}_1^\intercal,\cdots,\mathbf{q}_k^\intercal)$$

for $i = 1,\cdots,k$, and

$$\mathbf{q}_i^\intercal \leftarrow \arg\min_{\mathbf{q}_i^\intercal\geq 0} f(\mathbf{p}_1,\cdots,\mathbf{p}_k,\mathbf{q}_1^\intercal,\cdots,\mathbf{q}_k^\intercal)$$

for $i = 1,\cdots,k$.

The $2k$-block BCD algorithm has been studied as Hierarchical Alternating Least Squares (HALS) proposed by Cichocki et al. [5, 4] and independently by Ho et al. [11] as rank-one residue iteration (RRI).

1.3.1.3 BCD with k(n + m) Scalar Blocks

We can also partition the variables with the smallest $k(n+m)$ element blocks of scalars as in Figure 1.3, where every element of $\mathbf{P}$ and $\mathbf{Q}$ is considered as a block in the context of Theorem 1. To this end, it helps to write the objective function as a quadratic function of scalar p_{ij} or q_{ij} assuming all other elements in $\mathbf{P}$ and $\mathbf{Q}$ are fixed:

$$f(p_{ij}) = \|(\mathbf{r}_i^\intercal - \sum_{\tilde{k}\neq j} p_{i\tilde{k}}\mathbf{q}_{\tilde{k}}^\intercal) - p_{ij}\mathbf{q}_j^\intercal\|_2^2 + \text{const}, \tag{1.8a}$$

$$f(q_{ij}) = \|(\mathbf{r}_j - \sum_{\tilde{k}\neq i} \mathbf{p}_{\tilde{k}}q_{\tilde{k}j}) - \mathbf{p}_i q_{ij}\|_2^2 + \text{const}, \tag{1.8b}$$

where $\mathbf{r}_i^\intercal$ and $\mathbf{r}_j$ denote the i^{th} row and the j^{th} column of $\mathbf{R}$, respectively.

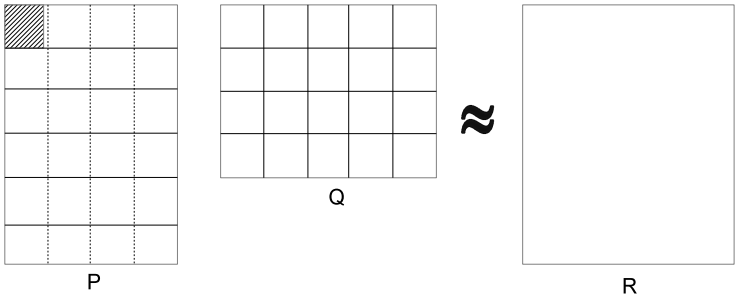

Fig. 1.3 BCD with k(n + m) Scalar Blocks

Kim et al. [16] discuss about NMF using BCD method.

1.3.2 Bounded Matrix Low Rank Approximation

The building blocks of BMA are column vectors $\mathbf{p}_x$ and row vectors $\mathbf{q}_x^\intercal$ of the matrix $\mathbf{P}$ and $\mathbf{Q}$ respectively. In this section, we discuss the idea behind finding these vectors $\mathbf{p}_x$ and $\mathbf{q}_x^\intercal$ such that all the elements in $\mathbf{T} + \mathbf{p}_x\mathbf{q}_x^\intercal \in [r_{min}, r_{max}]$ and the error $\|\mathbf{M} \cdot *(\mathbf{R} - \mathbf{PQ})\|_F^2$ is reduced. Here, $\mathbf{T} = \sum_{j=1, j\neq x}^{k} \mathbf{p}_j\mathbf{q}_j^\intercal$.

Problem (1.1) can be equivalently represented with a set of rank-one matrices $\mathbf{p}_x\mathbf{q}_x^\intercal$ as

$$\begin{aligned} \min_{\mathbf{p}_x,\mathbf{q}_x} \quad & \|\mathbf{M}\cdot*(\mathbf{R}-\mathbf{T}-\mathbf{p}_x\mathbf{q}_x^\intercal)\|_F^2 \\ & \forall x=[1,k] \\ \text{subject to} \quad & \\ & \mathbf{T}+\mathbf{p}_x\mathbf{q}_x^\intercal \le r_{max} \\ & \mathbf{T}+\mathbf{p}_x\mathbf{q}_x^\intercal \ge r_{min} \end{aligned} \tag{1.9}$$

Thus, we take turns solving for $\mathbf{p}_x$ and $\mathbf{q}_x^\intercal$. That is, assume we know $\mathbf{p}_x$ and find $\mathbf{q}_x^\intercal$ and vice versa. In the entire section we assume fixing column $\mathbf{p}_x$ and finding row $\mathbf{q}_x^\intercal$. Without loss of generality, all the discussion pertaining to finding $\mathbf{q}_x^\intercal$ with fixed $\mathbf{p}_x$ hold for the other scenario of finding $\mathbf{p}_x$ with fixed $\mathbf{q}_x^\intercal$.

There are different orders of updates of vector blocks when solving Problem (1.9). For example,

$$\mathbf{p}_1 \to \mathbf{q}_1^\intercal \to \cdots \to \mathbf{p}_k \to \mathbf{q}_k^\intercal \tag{1.10}$$

and

$$\mathbf{p}_1 \to \cdots \to \mathbf{p}_k \to \mathbf{q}_1^\intercal \to \cdots \to \mathbf{q}_k^\intercal. \tag{1.11}$$

Kim et al. [16] prove that Eq. (1.7) satisfies the formulation of BCD method. Eq. (1.7) when extended with the matrix $\mathbf{M}$ becomes Eq. (1.9). Here, the matrix $\mathbf{M}$ is like a filter matrix that defines the elements of $(\mathbf{R}-\mathbf{T}-\mathbf{p}_x\mathbf{q}_x^\intercal)$ to be included for the norm computation. Thus, Problem (1.9) is similar to Problem (1.7) and we can solve by applying $2k$ block BCD to update $\mathbf{p}_x$ and $\mathbf{q}_x^\intercal$ iteratively, although equation (1.9) appears not to satisfy the BCD requirements directly. We focus on the scalar block case, as it is convenient to explain regarding imposing bounds on the product of the low rank factors $\mathbf{PQ}$.

Also, according to BCD, the independent elements in a block can be computed simultaneously. Here, the computations of the elements $q_{xi}, q_{xj} \in \mathbf{q}_x^\intercal, i \ne j$, are independent of each other. Hence, the problem of finding row $\mathbf{q}_x^\intercal$ fixing column $\mathbf{p}_x$ is equivalent to solving the following problem

$$\begin{aligned} & \min_{q_{xi}} \|\mathbf{M}(:,i)\cdot*((\mathbf{R}-\mathbf{T})(:,i)-\mathbf{p}_x q_{xi})\|_F^2 \\ & \forall i=[1,m],\ \forall x=[1,k] \\ \text{subject to} \quad & \\ & \mathbf{T}(:,i)+\mathbf{p}_x q_{xi} \le r_{max} \\ & \mathbf{T}(:,i)+\mathbf{p}_x q_{xi} \ge r_{min} \end{aligned} \tag{1.12}$$

To construct the row vector $\mathbf{q}_x^\intercal$, we use $k(n+m)$ scalar blocks based on problem formulation (1.12). Theorem 3 identifies these best elements that construct $\mathbf{q}_x^\intercal$. As shown in Figure 1.4, given the **bold** blocks, $\mathbf{T}$, $\mathbf{R}$ and $\mathbf{p}_x$, we find the row vector $\mathbf{q}_x^\intercal = [q_{x1}, q_{x2}, \cdots, q_{xm}]$ for Problem (1.12). For this, let

us understand the boundary values of q_{xi} by defining two vectors, $\mathbf{l}$ bounding q_{xi} from below, and $\mathbf{u}$ bounding q_{xi} from above, i.e., $max(\mathbf{l}) \leq q_{xi} \leq min(\mathbf{u})$.

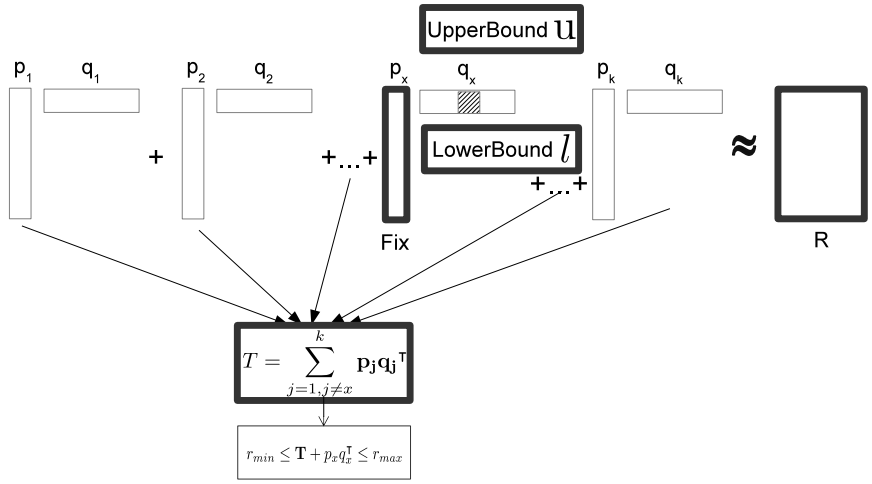

Fig. 1.4 Bounded Matrix Low Rank Approximation Solution Overview

Definition 1. The lower bound vector $\mathbf{l} = [l_1, \ldots, l_n] \in \mathbb{R}^n$ and the upper bound vector $\mathbf{u} = [u_1, \ldots, u_n] \in \mathbb{R}^n$ for a given $\mathbf{p}_x$ and $\mathbf{T}$ that bound q_{xi} are defined $\forall j \in [1, n]$ as

$$l_j = \begin{cases} \dfrac{r_{min} - \mathbf{T}(j,i)}{p_{jx}}, & p_{jx} > 0 \\ \dfrac{r_{max} - \mathbf{T}(j,i)}{p_{jx}}, & p_{jx} < 0 \\ -\infty, & otherwise \end{cases}$$

and

$$u_j = \begin{cases} \dfrac{r_{max} - \mathbf{T}(j,i)}{p_{jx}}, & p_{jx} > 0 \\ \dfrac{r_{min} - \mathbf{T}(j,i)}{p_{jx}}, & p_{jx} < 0 \\ \infty, & otherwise. \end{cases}$$

It is important to observe that the defined $\mathbf{l}$ and $\mathbf{u}$ – referred as `LowerBounds` and `UpperBounds` in Algorithm 1, are for a given $\mathbf{p}_x$ and $\mathbf{T}$ to bound q_{xi}. Alternatively, if we are solving $\mathbf{p}_x$ for a given $\mathbf{T}$ and $\mathbf{q}_x$, the above function correspondingly represents the possible lower and upper bounds for p_{ix}, where $\mathbf{l}, \mathbf{u} \in \mathbb{R}^m$.

Theorem 2. *Given* $\mathbf{R}$, $\mathbf{T}$, $\mathbf{p}_x$, *the* q_{xi} *is always bounded as* $max(\mathbf{l}) \leq q_{xi} \leq min(\mathbf{u})$.

Proof. It is easy to see that if $q_{xi} < max(\mathbf{l})$ or $q_{xi} > min(\mathbf{u})$, then $\mathbf{T}(:,i) + \mathbf{p}_x q_{xi}^\intercal \notin [r_{min}, r_{max}]$. □

Here, it is imperative to note that if q_{xi}, results in $\mathbf{T}(:,i) + \mathbf{p}_x q_{xi}^\intercal \notin [r_{min}, r_{max}]$, this implies that q_{xi} is either less than the $max(\mathbf{l})$ or greater than the $min(\mathbf{u})$. It cannot be any other inequality.

Given the boundary values of q_{xi}, Theorem 3 defines the solution to Problem (1.12).

Theorem 3. *Given* $\mathbf{T}$, $\mathbf{R}, \mathbf{p}_x, \mathbf{l}$ *and* $\mathbf{u}$, *let*

$$\hat{q_{xi}} = ([\mathbf{M}(:,i) \cdot *(\mathbf{R} - \mathbf{T})(:,i)]^\intercal \mathbf{p}_x)/(\|\mathbf{M}(:,i) \cdot *\mathbf{p}_x\|_2^2).$$

The unique solution q_{xi} – *referred as* `FindElement` *in Algorithm 1 to least squares problem* (1.12) *is given as*

$$q_{xi} = \begin{cases} max(\mathbf{l}), & if\, \hat{q_{xi}} < max(\mathbf{l}) \\ min(\mathbf{u}), & if\, \hat{q_{xi}} > min(\mathbf{u}) \\ \hat{q_{xi}}, & otherwise. \end{cases}$$

Proof. Out of Boundary: $q_{xi} < max(\mathbf{l})$ or $q_{xi} > min(\mathbf{u})$. Under this circumstance, the best value for q_{xi} is either $max(\mathbf{l})$ or $min(\mathbf{u})$. We can prove this by contradiction. Let us assume there exists a $\tilde{q_{xi}} = max(\mathbf{l}) + \delta; \delta > 0$ that is optimal to the Problem (1.12) for $q_{xi} < max(\mathbf{l})$. However, for $q_{xi} = max(\mathbf{l}) < \tilde{q_{xi}}$ is still a feasible solution for the Problem (1.12). Also, there does not exist a feasible solution that is less than $max(\mathbf{l})$, because the Problem (1.12) is quadratic in q_{xi}. Hence for $q_{xi} < max(\mathbf{l})$, the optimal value for the Problem (1.12) is $max(\mathbf{l})$. In similar direction we can show that the optimal value of q_{xi} is $min(\mathbf{u})$ for $q_{xi} > min(\mathbf{u})$.
Within Boundary: $max(\mathbf{l}) \leq q_{xi} \leq min(\mathbf{u})$.

Let us consider the objective function of unconstrained optimization problem (1.12). That is., $f = \min_{q_{xi}} \|\mathbf{M}(:,i) \cdot *((\mathbf{R} - \mathbf{T})(:,i) - \mathbf{p}_x q_{xi})\|_2^2$. The minimum value is determined by taking the derivative of f with respect to q_{xi} and equating it to zero.

$$
\begin{aligned}
\frac{\partial f}{\partial q_{xi}} &= \frac{\partial}{\partial q_{xi}} \left(\sum_{\substack{\texttt{all known ratings} \\ \texttt{in column } i}} (\mathbf{E}_i - \mathbf{p}_x q_{xi})^2 \right) \quad (\texttt{where } \mathbf{E} = \mathbf{R} - \mathbf{T}) \\
&= \frac{\partial}{\partial q_{xi}} \left(\sum_{\substack{\texttt{all known ratings} \\ \texttt{in column } i}} (\mathbf{E}_i - \mathbf{p}_x q_{xi})^\intercal (\mathbf{E}_i - \mathbf{p}_x q_{xi}) \right) \\
&= \frac{\partial}{\partial q_{xi}} \left(\sum_{\substack{\texttt{all known ratings} \\ \texttt{in column } i}} (\mathbf{E}_i^\intercal - q_{xi}\mathbf{p}_x^\intercal)(\mathbf{E}_i - \mathbf{p}_x q_{xi}) \right) \\
&= \frac{\partial}{\partial q_{xi}} \left(\sum_{\substack{\texttt{all known ratings} \\ \texttt{in column } i}} q_{xi}^2 \mathbf{p}_x^\intercal \mathbf{p}_x - q_{xi}\mathbf{E}_i^\intercal \mathbf{p}_x - q_{xi}\mathbf{p}_x^\intercal \mathbf{E}_i + \mathbf{E}_i^\intercal \mathbf{E}_i \right) \\
&= 2\|\mathbf{M}(:,i). * \mathbf{p}_x\|_2^2 q_{xi} - 2[\mathbf{M}(:,i). * (\mathbf{R} - \mathbf{T})(:,i)]^\intercal \mathbf{p}_x
\end{aligned} \tag{1.13}
$$

Now, equating $\frac{\partial f}{\partial q_{xi}}$ to zero will yield the optimum solution for the unconstrained optimization problem (1.12) as $q_{xi} = ([\mathbf{M}(:,i) \cdot *(\mathbf{R} - \mathbf{T})(:,i)]^\intercal \mathbf{p}_x)/(\|\mathbf{M}(:,i) \cdot *\mathbf{p}_x\|_2^2)$

□

In the similar direction the proof for Theorem 4 can also be established.

1.3.3 Bounding Existing ALS Algorithms (BALS)

Over the last few years, the recommender system algorithms have improved by leaps and bounds. The additional sophistication such as using only nearest neighbors during factorization, implicit ratings, time, etc., gave only a diminishing advantage for the Root Mean Square Error (RMSE) scores. That is, the improvement in RMSE score over the naive low rank approximation with implicit ratings is more than the improvement attained by utilizing both implicit ratings and time. Today, these algorithms are artificially truncating the estimated unknown ratings. However, it is important to investigate establishing bounds scientifically on these existing Alternating Least Squares (ALS) type algorithms.

Using matrix block BCD, we introduce a temporary variable $\mathbf{Z} \in \mathbb{R}^{n \times m}$ with box constraints to solve Problem (1.1),

$$
\begin{aligned}
&\min_{\boldsymbol{\Theta}, \mathbf{Z}, \mathbf{P}, \mathbf{Q}} \|\mathbf{M} \cdot *(\mathbf{R} - \mathbf{Z})\|_F^2 + \alpha \|\mathbf{Z} - f(\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q})\|_F^2 \\
&\text{subject to} \\
&\qquad r_{min} \leq \mathbf{Z} \leq r_{max}.
\end{aligned} \tag{1.14}
$$

The key question is identifying optimal $\mathbf{Z}$. We assume the iterative algorithm has a specific update order, for example, $\boldsymbol{\theta}_1 \rightarrow \cdots \rightarrow \boldsymbol{\theta}_l \rightarrow \mathbf{P} \rightarrow \mathbf{Q}$.

Before updating these parameters, we should have an optimal $\mathbf{Z}$, with the most recent values of $\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}$.

Theorem 4. *The optimal* $\mathbf{Z}$*, given* $\mathbf{R}, \mathbf{P}, \mathbf{Q}, \mathbf{M}, \boldsymbol{\Theta}$*, is* $\frac{\mathbf{M}.*(\mathbf{R}+\alpha f(\boldsymbol{\Theta},\mathbf{P},\mathbf{Q}))}{1+\alpha} + \mathbf{M}'.*f(\boldsymbol{\Theta}),\mathbf{P},\mathbf{Q})$*, where* $\mathbf{M}'$ *is the complement of the indicator boolean matrix* $\mathbf{M}$*.*

Proof. Given $\mathbf{R}, \mathbf{P}, \mathbf{Q}, \mathbf{M}, \boldsymbol{\Theta}$, the optimal $\mathbf{Z}$, is obtained by solving the following optimization problem.

$$\begin{aligned} G = \min_{\mathbf{Z}} & \|\mathbf{M} \cdot *(\mathbf{R} - \mathbf{Z})\|_F^2 + \alpha \|\mathbf{Z} - f(\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q})\|_F^2 \\ \text{subject to} & \\ & r_{min} \leq \mathbf{Z} \leq r_{max}. \end{aligned} \tag{1.15}$$

Taking the gradient $\frac{\partial G}{\partial Z}$ of the above equation to find the optimal solution yields, $\frac{\mathbf{M}.*(\mathbf{R}+\alpha f(\boldsymbol{\Theta},\mathbf{P},\mathbf{Q}))}{1+\alpha} + \mathbf{M}'.*f(\boldsymbol{\Theta},\mathbf{P},\mathbf{Q})$. The derivation is in the same lines as explained in equations (1.13) □

In the same direction of Theorem 3, we can show that if the values of $\mathbf{Z}$ are outside $[r_{min}, r_{max}]$, that is., $\mathbf{Z} > r_{max}$ and $\mathbf{Z} < r_{min}$, the optimal value is $\mathbf{Z} = r_{max}$ and $\mathbf{Z} = r_{min}$ respectively.

In the next Section, the implementation of the algorithm for BMA and its variants such as scalable and block implementations will be studied. Also, imposing bounds on existing ALS algorithms using $BALS$ will be investigated. As an example we will take existing algorithms from Graphchi [22] implementations and study imposing bounds using the $BALS$ framework.

1.4 Implementations

1.4.1 Bounded Matrix Low Rank Approximation

Given the discussion in the previous sections, we now have the necessary tools to construct the algorithm. In Algorithm 1, the $\mathbf{l}$ and $\mathbf{u}$ from Theorem 2 are referred to as `LowerBounds` and `UpperBounds`, respectively. Also, q_{xi} from Theorem 3 is referred to as `FindElement`. The BMA algorithm has three major functions: (1) Initialization, (2) Stopping Criteria and (3) Find the low rank factors $\mathbf{P}$ and $\mathbf{Q}$. In later sections, the initialization and stopping criteria are explained in detail. For now, we assume that two initial matrices $\mathbf{P}$ and $\mathbf{Q}$ are required, such that $\mathbf{PQ} \in [r_{min}, r_{max}]$, and that a stopping criterion will be used for terminating the algorithm, when the constructed matrices $\mathbf{P}$ and $\mathbf{Q}$ provide a good representation of the given matrix $\mathbf{R}$.

In the case of BMA algorithm, since multiple elements can be updated independently, we reorganize the scalar block BCD into $2k$ vector blocks. The BMA algorithm is presented as Algorithm 1.

input : Matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$, $r_{min}, r_{max} > 1$, reduced rank k
output: Matrix $\mathbf{P} \in \mathbb{R}^{n \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times m}$

`// Rand initialization of P, Q.`
Initialize $\mathbf{P}$, $\mathbf{Q}$ *as non-negative random matrices* ;
`// modify random` $\mathbf{PQ}$ `such that`
`//` $\mathbf{PQ} \in [r_{min}, r_{max}]$
`// maxelement of` $\mathbf{PQ}$ `without first`
`// column of` $\mathbf{P}$ `and first row of` $\mathbf{Q}$
$maxElement = max(\mathbf{P}(:, 2 : end) * \mathbf{Q}(2 : end, :))$;
$\alpha \leftarrow \sqrt{\frac{r_{max} - 1}{maxElement}}$;
$\mathbf{P} \leftarrow \alpha \cdot \mathbf{P}$;
$\mathbf{Q} \leftarrow \alpha \cdot \mathbf{Q}$;
$\mathbf{P}(:, 1) \leftarrow 1$;
$\mathbf{Q}(1, :) \leftarrow 1$;
$\mathbf{M} \leftarrow$ `ComputeRatedBinaryMatrix(`$\mathbf{R}$`)`;
while *stopping criteria not met* **do**
  for $x \leftarrow 1$ **to** k **do**
    $\mathbf{T} \leftarrow \sum_{j=1, j \neq x}^{k} \mathbf{p}_j \mathbf{q}_j^\intercal$;
    for $i \leftarrow 1$ **to** m **do**
      `// Find vectors` $\mathbf{l}, \mathbf{u} \in \mathbb{R}^n$ `as in Definition 1`
      $\mathbf{l} \leftarrow$ `LowerBounds(`$r_{min}, r_{max}, \mathbf{T}, i, \mathbf{p}_x$`)`;
      $\mathbf{u} \leftarrow$ `UpperBounds(`$r_{min}, r_{max}, \mathbf{T}, i, \mathbf{p}_x$`)`;
      `// Find vector` $\mathbf{q}_x^\intercal$ `fixing` $\mathbf{p}_x$ `as in Theorem 3`
      $q_{xi} \leftarrow$ `FindElement(`$\mathbf{p_x}, \mathbf{M}, \mathbf{R}, \mathbf{T}, i, x$`)`;
      if $q_{xi} < max(\mathbf{l})$ **then**
        $q_{xi} \leftarrow max(\mathbf{l})$;
      else if $q_{xi} > min(\mathbf{u})$ **then**
        $q_{xi} \leftarrow min(\mathbf{u})$;
    for $i \leftarrow 1$ **to** n **do**
      `// Find vectors` $\mathbf{l}, \mathbf{u} \in \mathbb{R}^m$ `as in Definition 1`
      $\mathbf{l} \leftarrow$ `LowerBounds(`$r_{min}, r_{max}, \mathbf{T}, i, \mathbf{q}_x^\intercal$`)`;
      $\mathbf{u} \leftarrow$ `UpperBounds(`$r_{min}, r_{max}, \mathbf{T}, i, \mathbf{q}_x^\intercal$`)`;
      `// Find vector` $\mathbf{p}_x$ `fixing` $\mathbf{q}_x^\intercal$ `as in Theorem 3`
      $p_{ix} \leftarrow$ `FindElement(`$\mathbf{q}_x^\intercal, \mathbf{M}^\intercal, \mathbf{R}^\intercal, \mathbf{T}^\intercal, i, x$`)`;
      if $p_{ix} < max(\mathbf{l})$ **then**
        $p_{ix} \leftarrow max(\mathbf{l})$;
      else if $p_{ix} > min(\mathbf{u})$ **then**
        $p_{ix} \leftarrow min(\mathbf{u})$;

Algorithm 1: Bounded Matrix Low Rank Approximation (BMA)

Algorithm 1 works very well and yields low rank factors $\mathbf{P}$ and $\mathbf{Q}$ for a given matrix $\mathbf{R}$ such that $\mathbf{PQ} \in [r_{min}, r_{max}]$. However, when applied for very large scale matrices, such as recommender systems, it can only be run on machines with a large amount of memory. We address scaling the algorithm on multi core systems and machines with low memory in the next section.

1.4.2 Scaling up Bounded Matrix Low Rank Approximation

In this section, we address the issue of scaling the algorithm for large matrices with missing elements. Two important aspects of making the algorithm run for large matrices are running time and memory. We discuss the parallel implementation of the algorithm, which we refer to as *Parallel Bounded Matrix Low Rank Approximation*. Subsequently, we also discuss a method called *Block Bounded Matrix Low Rank Approximation*, which will outline the details of executing the algorithm for large matrices in low memory systems. Let us start this section by discussing *Parallel Bounded Matrix Low Rank Approximation*.

1.4.2.1 Parallel Bounded Matrix Low Rank Approximation

In the case of the BCD method, the solutions of the sub-problems that depend on each other have to be computed sequentially to make use of the most recent values. However, if solutions for some blocks are independent of each other, it is possible to compute them simultaneously. We can observe that, according to Theorem 3, all elements $q_{xi}, q_{xj} \in \mathbf{q}_x^\intercal, i \neq j$ are independent of each other. We are leveraging this characteristic to parallelize the **for** loops in Algorithm 1. Nowadays, virtually all commercial processors have multiple cores. Hence, we can parallelize finding the q_{xi}'s across multiple cores. Since it is trivial to change the **for** in step 12 and step 20 of Algorithm 1 to **parallel for** the details will be omitted.

It is obvious to see that the $\mathbf{T}$ at step 11 in Algorithm 1 requires the largest amount of memory. Also, the function *FindElement* in step 15 takes a sizable amount of memory. Hence, it is not possible to run the algorithm for large matrices on machines with low memory, e.g., with rows and columns on the scale of 100,000's. Thus, we propose the following algorithm to mitigate this limitation: Block BMA.

1.4.2.2 Block Bounded Matrix Low Rank Approximation

To facilitate understanding of this section, let us define β – a data structure in memory factor. That is, maintaining a floating scalar as a sparse matrix with one element or full matrix with one element takes different amounts of memory. This is because of the data structure that is used to represent the numbers in the memory. The amount of memory is also dependent on using single or double precision floating point precision. Typically, in Matlab, the data structure in memory factor β for full matrix is around 10. Similarly, in Java, the β factor for maintaining a number in an ArrayList is around 8. Let, $memsize(v)$ be the function that returns the approximate memory size of a variable v. Generally, $memsize(v)$ = number of elements in v * size of each element * β. Consider an example of maintaining 1000 floating point numbers on an ArrayList of a Java program. The approximate memory would be 1000*4*8 = 32000 bytes $\approx$ 32KB in contrast to the actual 4KB due to the factor β=8.

As discussed earlier, for most of the real world large datasets such as Netflix, Yahoo music, online dating, book crossing, etc., it is impossible to keep the entire matrix $\mathbf{T}$ in memory. Also, notice that, according to Theorem 3 and Definition 1, we need only the i-th column of $\mathbf{T}$ to compute q_{xi}. The block size of $q_{x\mathbf{i}}$ to compute in one core of the machine is dependent on the size of $\mathbf{T}$ and $FindElements$ that fits in memory.

On the one hand, partition $\mathbf{q}_x$ to fit the maximum possible $\mathbf{T}$ and $FindElements$ in the entire memory of the system. If very small partitions are created such that, we can give every core some amount of work so that the processing capacity of the system is not underutilized. The disadvantage of the former, is that only one core is used. However, in the latter case, there is a significant communication overhead. Figure 1.5 gives the pictorial view of the Block Bounded Matrix Low Rank Approximation.

We determined the number of blocks = memsize(full($\mathbf{R}$)+other variables of FindElement)/(system memory * number of d cores). The full($\mathbf{R}$) is a non-sparse representation and $d \leq$ number of cores available in the system. Typically, for most of the datasets, we achieved minimum running time when we used 4 cores and 16 blocks. That is, we find 1/16-th of $\mathbf{q}_x^\intercal$ concurrently on 4 cores.

For convenience, we have presented the Block BMA as Algorithm 2. We describe only the algorithm to find the partial vector of $\mathbf{q}_x^\intercal$ given $\mathbf{p}_x$. To find more than one element, Algorithm 1 is modified such that the vectors $\mathbf{l}, \mathbf{u}, \mathbf{p}_x$ are matrices $\mathbf{L}, \mathbf{U}, \mathbf{P}_{blk}$, respectively, in Algorithm 2. Algorithm 2 replaces the steps 12 – 19 in Algorithm 1 for finding $\mathbf{q}$ and similarly for finding $\mathbf{p}$ from step 20 – 27. The initialization and the stopping criteria for Algorithm 2 are similar to those of Algorithm 1. Also included are the necessary steps to handle numerical errors as part of Algorithm 2 explained in Section 1.5. Figure 1.6 in Section 1.5, presents the speed up of the algorithm.

input : Matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$, set of indices $\mathbf{i}$, current $\mathbf{p}_x$, x, current $\mathbf{q}'_x$, r_{min}, r_{max}
output: Partial vector $\mathbf{q}_x$ of requested indices $\mathbf{i}$

`// ratings of input indices i`
$\mathbf{R}_{blk} \leftarrow \mathbf{R}(:, \mathbf{i})$;
$\mathbf{M}_{blk} \leftarrow$ `ComputeRatedBinaryMatrix(`$\mathbf{R}_{blk}$`)`;
$\mathbf{P}_{blk} \leftarrow$ `Replicate(`$\mathbf{p}_x, size(\mathbf{i})$`)`;
`// save` $\mathbf{q}_x(\mathbf{i})$
$\mathbf{q}'_{blk} \leftarrow \mathbf{q}_x(\mathbf{i})$;
`//` $\mathbf{T}_{blk} \in n \times size(\mathbf{i})$ `of input indices i`
$\mathbf{T}_{blk} \leftarrow \sum_{j=1, j \neq x}^{k} \mathbf{p}_j \mathbf{q}_{blk}^{\intercal}$;
`// Find matrix` $\mathbf{L}, \mathbf{U} \in \mathbb{R}^{n \times size(\mathbf{i})}$ `as in Definition 1`
$\mathbf{L} \leftarrow$ `LowerBounds(`$r_{min}, r_{max}, \mathbf{T}, \mathbf{i}, \mathbf{p}_x$`)`;
$\mathbf{U} \leftarrow$ `UpperBounds(`$r_{min}, r_{max}, \mathbf{T}, \mathbf{i}, \mathbf{p}_x$`)`;
`// Find vector` $\mathbf{q}_{blk}$ `fixing` $\mathbf{p}_x$ `as in Theorem 3`
$\mathbf{q}_{blk} = ([\mathbf{M}_{blk} \cdot *(\mathbf{R}_{blk} - \mathbf{T}_{blk})]^{\intercal} \mathbf{P}_{blk}) / (\|\mathbf{M}_{blk} \cdot * \mathbf{P}_{blk}\|_F^2)$;
`// Find indices of` $\mathbf{q}_{blk}$ `that are not within bounds`
$\mathbf{idxlb} \leftarrow find(\mathbf{q}_{blk} < max(\mathbf{L}))$;
$\mathbf{idxub} \leftarrow find(\mathbf{q}_{blk} > min(\mathbf{U}))$;
`// case A & B numerical errors in Section 1.5`
$idxcase1 \leftarrow find([\mathbf{q}'_{blk} \approx max(\mathbf{L})] or [\mathbf{q}'_{blk} \approx min(\mathbf{U}])$;
$idxcase2 \leftarrow find([max(\mathbf{L}) \approx min(\mathbf{U})] or [max(\mathbf{L}) > min(\mathbf{U})])$;
$idxdontchange \leftarrow idxcase1 \cup idxcase2$;
`// set appropriate values of` $\mathbf{q}_{blk} \notin [max(\mathbf{L}), min(\mathbf{U})]$
$\mathbf{q}_{blk}(\mathbf{idxlb} \setminus idxdontchange) \leftarrow max(\mathbf{L})(idxlb \setminus idxdontchange)$;
$\mathbf{q}_{blk}(\mathbf{idxub} \setminus idxdontchange) \leftarrow min(\mathbf{U})(idxub \setminus idxdontchange)$;
$\mathbf{q}_{blk}(idxdontchange) \leftarrow \mathbf{q}'_{blk}(idxdontchange)$;

Algorithm 2: Block BMA

1.4.3 Bounding Existing ALS Algorithms(BALS)

In this section we examine the algorithm for solving the Equation (1.14) based on Theorem 4 to find the low rank factors $\mathbf{P}$, $\mathbf{Q}$ and $\boldsymbol{\Theta}$. For the time being, assume that we need an initial, $\boldsymbol{\Theta}, \mathbf{P}$ and $\mathbf{Q}$ to start the algorithm. Also, we need update functions for $\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}$ and a stopping criteria for terminating the algorithm. The stopping criteria determines whether the constructed matrices $\mathbf{P}$ and $\mathbf{Q}$ and $\boldsymbol{\Theta}$ provide a good representation of the given matrix $\mathbf{R}$.

In the *ComputeZ* function, if the values of $\mathbf{Z}$ are outside $[r_{min}, r_{max}]$, that is., $\mathbf{Z} > r_{max}$ and $\mathbf{Z} < r_{min}$, set the corresponding values of $\mathbf{Z} = r_{max}$ and $\mathbf{Z} = r_{min}$ respectively.

Most of the ALS based recommender system algorithms have clear defined blocks on $\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}$ as discussed in Section 1.3.1. That is, either they are partitioned as a matrix, vector or element blocks. Also, there is always an update order that is adhered to. For example, $\boldsymbol{\theta}_1 \rightarrow \boldsymbol{\theta}_2 \rightarrow \cdots \boldsymbol{\theta}_l \rightarrow \mathbf{p}_1 \rightarrow \mathbf{p}_2 \rightarrow \cdots \rightarrow \mathbf{p}_k \rightarrow \mathbf{q}_1 \rightarrow \mathbf{q}_2 \rightarrow \cdots \rightarrow \mathbf{q}_k$. If the algorithm meets these

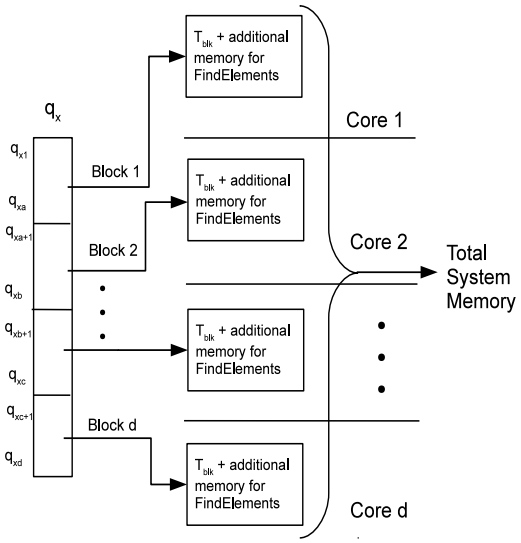

Fig. 1.5 Block Bounded Matrix Low Rank Approximation

input : Matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$, $r_{min}, r_{max} > 1$, reduced rank k
output: Parameters $\boldsymbol{\Theta}$, Matrix $\mathbf{P} \in \mathbb{R}^{n \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times m}$

`// Rand initialization of Θ, P, Q.`
Initialize $\boldsymbol{\Theta}, \mathbf{P}$, $\mathbf{Q}$ *as a random matrix* ;
$\mathbf{M} \leftarrow$ `ComputeRatedBinaryMatrix(`$\mathbf{R}$`)`;
`// Compute Z as in Theorem 4`
$\mathbf{Z} \leftarrow$ `ComputeZ(`$\mathbf{R}, \mathbf{P}, \mathbf{Q}, \boldsymbol{\Theta}$`)`;
while *stopping criteria not met* **do**
  $\boldsymbol{\theta_i} \leftarrow \underset{\boldsymbol{\theta}_i}{\operatorname{argmin}} \|\mathbf{M} \cdot *(\mathbf{R} - f(\boldsymbol{\theta}_i, \mathbf{P}, \mathbf{Q}))\|_F^2 \;\; \forall 1 \leq i \leq l$;
  $\mathbf{P} \leftarrow \underset{\mathbf{P}}{\operatorname{argmin}} \|\mathbf{M} \cdot *(\mathbf{R} - f(\boldsymbol{\theta}_i, \mathbf{P}, \mathbf{Q}))\|_F^2$;
  $\mathbf{Q} \leftarrow \underset{\mathbf{Q}}{\operatorname{argmin}} \|\mathbf{M} \cdot *(\mathbf{R} - f(\boldsymbol{\theta}_i, \mathbf{P}, \mathbf{Q}))\|_F^2$;
  $\mathbf{Z} \leftarrow$ `ComputeZ(`$\mathbf{R}, \mathbf{P}, \mathbf{Q}, \boldsymbol{\Theta}$`)`;
  if $z_{ij} > r_{max}$ **then**
    $z_{ij} = r_{max}$
  if $z_{ij} < r_{min}$ **then**
    $z_{ij} = r_{min}$

Algorithm 3: Bounding Existing ALS Algorithm (BALS)

characteristics, we can prove that the algorithm converges to a stationary point.

Corollary 1. *If the recommender system algorithm f based on alternating least squares, satisfies the following characteristics: (1) is an iterative coordinate descent algorithm, (2) defines blocks over the optimization variables $\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}$, and (3) has orderly optimal block updates of one block at a time and always uses the latest blocks, f converges to a stationary point of (1.14).*

Proof. This is based on Theorem 1. The BALS Algorithm 3 for the formulation (1.14) satisfies the above characteristics and hence the algorithm converges to a stationary point. □

At this juncture, it is important to discuss the applicability of BMA for gradient descent type of algorithms such as SVD++, Bias-SVD, time-SVD++, and others. For brevity, we consider the SGD for the simple matrix factorization problem explained in equation (1.1). For a gradient descent type of algorithm, we update the current value of $p_{uk} \in \mathbf{P}$ and $q_{ki} \in \mathbf{Q}$ based on the gradient of the error $e_{ui} = r_{ui} - f(\mathbf{P'}, \mathbf{Q'})$. Assuming $\lambda_p = \lambda_q = \lambda$, the update equations are $p_{uk} = p'_{uk} + (e_{ui}q'_{ki} - \lambda p'_{uk})$ and $q_{ki} = q'_{ki} + (e_{ui} * p'_{uk} - \lambda q'_{ki})$. In the case of BALS for unobserved pairs (u, i), we use $z_{ui} = f(\mathbf{P'}, \mathbf{Q'})$ instead of $r_{ui} = 0$. Thus, in the case of the BALS extended gradient descent algorithms, the error $e_{ui} = 0$, for unobserved pairs (u, i). That is, the updates of p_{uk} and q_{ki} are dependent only on the observed entries and our estimations for unrated pairs (u, i) do not have any impact.

It is also imperative to understand that we are considering only the existing recommender system algorithms that minimizes the RMSE error of the observed entries against its estimation as specified in (1.2). We have not analyzed the problems that utilize different loss functions such as KL-Divergence.

In this section, we also present an example for bounding an existing algorithm, Alternating Least Squares, with Regularization in Graphchi. GraphChi [22] is an open source library that allows distributed processing of very large graph computations on commodity hardware. It breaks large graphs into smaller chunks and uses a parallel sliding windows method to execute large data mining and machine learning programs in small computers. As part of the library samples, it provides implementations of many recommender system algorithms. In this section we discuss extending the existing ALSWR implementation in Graphchi to impose box constraints using our framework.

Graphchi programs are written in the vertex-centric model and runs vertex-centric programs asynchronously (i.e., changes written to edges are immediately visible to subsequent computation), and in parallel. Any Graphchi program has three important functions: (1) $beforeIteration$, (2) $afterIteration$, and (3) *update* function. The function $beforeIteration$ and $afterIteration$ are executed sequentially in a single core, whereas the *update* function for all the vertices is executed in parallel across multiple cores of the

machine. Such parallel updates are useful for updating independent blocks. For example, in our case, every vector $\mathbf{p}_i^\intercal$ is independent of $\mathbf{p}_j^\intercal$, for $i \neq j$.

Graphchi models the collaborative filtering problem as a bipartite graph between a user vertex and item vertex. The edges that flow between the user-item partition are ratings. That is, a weighted edge, between a user u and an item i vertex represent the user u's rating for the item i. The user vertex has only outbound edges and an item vertex has only inbound edges. The *update* function is called for all the user and item vertices. The vertex *update* solves a regularized least-squares system, with neighbors' latent factors as input.

One of the major challenges for existing algorithms to enforce bounds using the BALS framework is memory. The matrix $\mathbf{Z}$ is dense and may not be accommodative in memory. That is, consider the $\mathbf{Z}$ for the book crossing dataset[1]. The dataset provides ratings of 278,858 users against 271,379 books. The size of $\mathbf{Z}$ for such a dataset would be $numberofusers * numberofitems * size(double)$ = 278858*271379*8 bytes $\approx$ 563GB. This data size is too large even for a server system. To overcome this problem, we save the $\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}$ of previous iterations as $\boldsymbol{\Theta}', \mathbf{P}', \mathbf{Q}'$. Instead of having the the entire $\mathbf{Z}$ matrix in memory, we compute the z_{ui} during the update function.

In the interest of space, we present the pseudo code for the *update* function alone. In the $beforeIteration$, function, we backup the existing variables $\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}$ as $\boldsymbol{\Theta}', \mathbf{P}', \mathbf{Q}'$. The $afterIteration$ function computes the RMSE of the validation/training set and determines the stopping criteria.

input : Vertex v user/item, GraphContext ctx, $\mathbf{Q}, \mathbf{P}', \mathbf{Q}', \boldsymbol{\Theta}'$
output: The u^{th} row of $\mathbf{P}$ matrix $\mathbf{p}_u^\intercal \in \mathbb{R}^k$ or the i^{th} column $\mathbf{q}_i \in \mathbb{R}^k$
// u^{th} row of matrix Z based on Theorem 4. $\boldsymbol{\Theta}', \mathbf{P}', \mathbf{Q}'$ are the $\boldsymbol{\Theta}, \mathbf{P}, \mathbf{Q}$ from previous iteration.
// Whether the vertex is a user/item vertex is determined by the number of incoming/outgoing edges. For user vertex the number of incoming edges = 0 and for item vertex the number of outgoing edges = 0
1 **if** *vertex v is user u vertex* **then**
    // update $\mathbf{p}_u^\intercal$
2   $\mathbf{z}_u^\intercal \in \mathbb{R}^m \leftarrow f(\mathbf{P}', \mathbf{Q}')$;
    // We are replacing the $\mathbf{a}_u$ in the original algorithm with $\mathbf{z}_u$
3   $\mathbf{p}_u^\intercal \leftarrow (\mathbf{Q}\mathbf{Q}^\intercal) \setminus (\mathbf{z}_u^\intercal * \mathbf{Q}^\intercal)^\intercal$;
4 **else**
    // update $\mathbf{q}_i$
5   $\mathbf{z}_i \in \mathbb{R}^n \leftarrow f(\mathbf{P}', \mathbf{Q}')$;
    // We are replacing the $\mathbf{a}_i$ in the original algorithm with $\mathbf{z}_i$
6   $\mathbf{q}_i \leftarrow (\mathbf{P}^\intercal\mathbf{P}) \setminus (\mathbf{P}^\intercal * \mathbf{z}_i)$;

Algorithm 4: update function

Considering Algorithm 4, it is important to observe that we use the previous iteration's $\mathbf{P}', \mathbf{Q}', \boldsymbol{\Theta}'$ only for the computation of $\mathbf{Z}$. However, for $\mathbf{P}, \mathbf{Q}$

[1] The details about this dataset can be found in Table 1.2

updates, the current latest blocks are used. Also, we cannot store the matrix $\mathbf{M}$ in memory. We know that Graphchi, as part of the $Vertex$ information, passes the set of incoming and outgoing edges to and from the vertex. The set of outgoing edges from the user vertex u to the item vertex v, provides information regarding the items rated by the user u. Thus, we use this information rather than maintaining $\mathbf{M}$ in memory. The performance comparison between ALSWR and ALSWR-Bounded on the Netflix dataset [1] is presented in Table 1.5. Similarly, we also bounded Probabilistic Matrix Factorization (PMF) in the Graphchi library and compared the performances of bounded ALSWR and bounded PMF algorithms using the BALS framework with its artificially truncated version on various real world datasets (see Table 1.6).

1.4.4 Parameter Tuning

In the case of recommender systems the missing ratings are provided as ground truth in the form of test data. The dot product of $\mathbf{P}(u,:)$ and $\mathbf{Q}(:,i)$ gives the missing rating of a (u,i) pair. In such cases, the accuracy of the algorithm is determined by the Root Mean Square Error (RMSE) of the predicted ratings against the ground truth. It is unimportant how good the algorithm converges for a given rank k.

This section discusses ways to improve the RMSE of the predictions against the missing ratings by tuning the parameters of the BMA algorithm and BALS framework.

1.4.4.1 Initialization

The BMA algorithm can converge to different points depending on the initialization. In Algorithm 1, it was shown how to use random initialization so that $\mathbf{PQ} \in [r_{min}, r_{max}]$. In general, this method should provide good results.

However, in the case of recommender systems, this initialization can be tuned, which can give even better results. According to Koren [18], one good baseline estimate for a missing rating (u,i) is $\mu + g_u + h_i$, where μ is the average of the known ratings, and g_u and h_i are the bias of user u and item i, respectively. We initialized $\mathbf{P}$ and $\mathbf{Q}$ in the following way

$$\mathbf{P} = \begin{pmatrix} \frac{\mu}{k-2} & \cdots & \frac{\mu}{k-2} & g_1 & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ \frac{\mu}{k-2} & \cdots & \frac{\mu}{k-2} & g_n & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{Q} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \\ h_1 & h_2 & \cdots & h_m \end{pmatrix},$$

such that $\mathbf{PQ}(u,i) = \mu + g_u + h_i$. That is, let the first $k-2$ columns of $\mathbf{P}$ be $\frac{\mu}{k-2}$, $\mathbf{P}(:,k-1) = \mathbf{g}$ and $\mathbf{P}(:,k) = 1$. Let all the $k-1$ rows of $\mathbf{Q}$ be 1's and $\mathbf{Q}(k,:) = \mathbf{h}^\intercal$. We call this a baseline initialization.

1.4.4.2 Reduced Rank k

In the case of regular low rank approximation with all known elements, the higher the k, the closer the low rank approximation is to the input matrix [16]. However, in the case of predicting with the low rank factors, a good k depends on the nature of the dataset. Even though, for a higher k, the low rank approximation is closer to the known rating of the input $\mathbf{R}$, the RMSE on the test data may be poor. In Table 1.3, we can observe the behavior of the RMSE on the test data against k. In most cases, a good k is determined by trial and error for the prediction problem.

1.4.4.3 Stopping Criterion $\mathfrak{C}$

The stopping criterion defines the goodness of the low rank approximation for the given matrix and the task for which the low rank factors are used. The two common stopping criteria are – (1) For a given rank k, the product of the low rank factors $\mathbf{PQ}$ should be close to the known ratings of the matrix and (2) The low rank factors $\mathbf{P}, \mathbf{Q}$ should perform the prediction task on a smaller validation set which has the same distribution as the test set. The former is common when all the elements of $\mathbf{R}$ are known. We discuss only the latter, which is important for recommender systems.

The stopping criterion $\mathfrak{C}$ for the recommender system is the increase of $\sqrt{\frac{\|\mathbf{M} \cdot *(\mathbf{V} - \mathbf{PQ})\|_F^2}{numRatings\ in\ \mathbf{V}}}$, for some validation matrix $\mathbf{V}$, which has the same distribution as the test matrix between successive iterations. Here, $\mathbf{M}$ is for the validation matrix $\mathbf{V}$. This stopping criterion has diminishing effect as the number of iterations increases. Hence, we also check whether $\sqrt{\frac{\|\mathbf{M} \cdot *(\mathbf{V} - \mathbf{PQ})\|_F^2}{numRatings\ in\ \mathbf{V}}}$ did not change in successive iterations at a given floating point precision, e.g., 1e-5.

It is trivial to show that, for the above stopping criterion $\mathfrak{C}$, Algorithm 1 terminates for any input matrix $\mathbf{R}$. At the end of an iteration, we terminate if the RMSE on the validation set has either increased or marginally decreased.

1.5 Experimentation

Experimentation was conducted in various systems with memory as lows as 16GB. One of the major challenges during experimentation is numerical errors. The numerical errors could result in $\mathbf{T}+\mathbf{p}_x\mathbf{q}_x^\intercal \notin [r_{min}, r_{max}]$. The two fundamental questions to solve the numerical errors are: (1) How to identify the occurrence of a numerical error? and (2) What is the best possible value to choose in the case of a numerical error?

We shall start by addressing the former question of potential numerical errors that arise in the BMA Algorithm 1. It is important to understand that if we are well within bounds, i.e., if $max(\mathbf{l}) < q_{xi} < min(\mathbf{u})$, we are not essentially impacted by the numerical errors. It is critical only when q_{xi} is out of the bounds, that is, $q_{xi} < max(\mathbf{l})$ or $q_{xi} > min(\mathbf{u})$ and approximately closer to the boundary discussed as in (*Case A* and *Case B*). For discussion let us assume we are improving the old value of q'_{xi} to q_{xi} such that we minimize the error $\|\mathbf{M} \cdot *(\mathbf{R} - \mathbf{T} - \mathbf{p}_x\mathbf{q}_x^\intercal)\|_F^2$.

Case A: $q'_{xi} \approx max(\mathbf{l})$ or $q'_{xi} \approx min(\mathbf{u})$:
This is equivalent to saying q'_{xi} is already optimal for the given $\mathbf{p}_x$ and $\mathbf{T}$ and there is no further improvement possible. Under this scenario, if $q'_{xi} \approx q_{xi}$ it is better to retain q'_{xi} irrespective of the new q_{xi} found.

Case B: $max(\mathbf{l}) \approx min(\mathbf{u})$ or $max(\mathbf{l}) > min(\mathbf{u})$:
According to Theorem 2, we know that $max(\mathbf{l}) < min(\mathbf{u})$. Hence, if $max(\mathbf{l}) > min(\mathbf{u})$, it is only the result of numerical errors.

In all the above cases during numerical errors, we are better off retaining the old value q'_{xi} against the new value q_{xi}. This covers Algorithm 2 – Block BMA for consideration of numerical errors.

We experimented with this Algorithm 2 among varied bounds using very large matrix sizes taken from the real world datasets. The datasets used for our experiments included the Movielens 10 million [1], Jester [8], Book crossing [32] and Online dating dataset [3]. The characteristics of the datasets are presented in Table 1.2.

Dataset	**Rows**	**Columns**	**Ratings** (millions)	**Density**	**Ratings Range**
Jester	73421	100	4.1	0.5584	[-10,10]
Movielens	71567	10681	10	0.0131	[1,5]
Dating	135359	168791	17.3	0.0007	[1,10]
Book crossing	278858	271379	1.1	0.00001	[1,10]
Netflix	17770	480189	100.4	0.01	[1,5]

Table 1.2 Datasets for experimentation

We have chosen Root Mean Square Error (RMSE) – a defacto metric for recommender systems. The RMSE is compared for BMA with baseline initialization (BMA –Baseline) and BMA with random initialization (BMA

–Random) against the other algorithms on all the datasets. The algorithms used for comparison are ALSWR (alternating least squares with regularization) [31], SGD [7], SVD++ [18] and Bias-SVD [18] and its implementation in Graphlab (http://graphlab.org/) [24] software package. We implemented our algorithm in Matlab and used the parallel computing toolbox for parallelizing across multiple cores.

For parameter tuning, we varied the number of reduced rank k and tried different initial matrices for our algorithm to compare against all other algorithms mentioned above. For every k, every dataset was randomly partitioned into 85% training, 5% validation and 10% test data. We ran all algorithms on these partitions and computed their RMSE scores. We repeated each experiment 5 times and reported their RMSE scores in Table 1.3, where each resulting value is the average of the RMSE scores on a randomly chosen test set for 5 runs. Table 1.3 summarizes the RMSE comparison of all the algorithms.

Dataset	k	**BMA Baseline**	**BMA Random**	**ALSWR**	**SVD++**	**SGD**	**Bias-SVD**
Jester	10	4.3320	4.6289	5.6423	5.5371	5.7170	5.8261
Jester	20	4.3664	4.7339	5.6579	5.5466	5.6752	5.7862
Jester	50	4.5046	4.7180	5.6713	5.5437	5.6689	5.7956
Movielens10M	10	0.8531	0.8974	1.5166	1.4248	1.2386	1.2329
Movielens10M	20	0.8526	0.8931	1.5158	1.4196	1.2371	1.2317
Movielens10M	50	0.8553	0.8932	1.5162	1.4204	1.2381	1.2324
Dating	10	1.9309	2.1625	3.8581	4.1902	3.9082	3.9052
Dating	20	1.9337	2.1617	3.8643	4.1868	3.9144	3.9115
Dating	50	1.9434	2.1642	3.8606	4.1764	3.9123	3.9096
Book Crossing	10	1.9355	2.8137	4.7131	4.7315	5.1772	3.9466
Book Crossing	20	1.9315	2.4652	4.7212	4.6762	5.1719	3.9645
Book Crossing	50	1.9405	2.1269	4.7168	4.6918	5.1785	3.9492

Table 1.3 RMSE Comparison of Algorithms on Real World Datasets

The Algorithm 1 consistently outperformed existing state-of-the-art algorithms. One of the main reason for the consistent performance is the absence of hyper parameters. In the case of machine learning algorithms, there are many parameters that need to be tuned for performance. Even though the algorithms perform the best when provided with the right parameters, identifying these parameters is a formidable challenge, usually by trial and error methods. For example, in Table 1.3, we can observe that the Bias-SVD, an algorithm without hyper parameters, performed better than its extension SVD++ with default parameters in many cases. The BMA algorithm without hyper parameters performed well on real world datasets, albeit a BMA with hyper parameters and the right parametric values would have performed even better.

Recently, there has been a surge in interest to understand the temporal impact on the ratings. Time-svd++ [19] is one such algorithm that leverages

the time of rating to improve prediction accuracy. Also, the most celebrated dataset in the recommender system community is the Netflix dataset, since the prize money is attractive and it represents the first massive dataset for recommender systems that was publicly made available. The Netflix dataset consists of 17,770 users who rated 480,189 movies in a scale of [1 ... 5]. There was a total of 100,480,507 ratings in the training set and 1,408,342 ratings in the validation set. All the algorithms listed above were invented to address the Netflix challenge. Even though the book crossing dataset [32] is bigger than the Netflix, we felt our study is not complete without experimenting on Netflix and comparing against time-SVD++. However, the major challenge is that the Netflix dataset has been withdrawn from the internet and its test data is no longer available. Hence, we extracted a small sample of 5% from the training data as a validation set and tested the algorithm against the validation set that was supplied as part of the training package. We performed this experiment and the results are presented in Table 1.4. For a better comparison, we also present the original Netflix test scores for SVD++ and time-SVD++ algorithms from [19]. These are labeled *SVD++-Test* and *time-SVD++-Test*, respectively. Our BMA algorithm outperformed all the algorithms on the Netflix dataset when tested on the validation set supplied as part of the Netflix training package.

Algorithm	$k = 10$	$k = 20$	$k = 50$	$k = 100$
BMA Baseline	0.9521	0.9533	0.9405	0.9287
BMA Random	0.9883	0.9569	0.9405	0.8777
ALSWR	1.5663	1.5663	1.5664	1.5663
SVD++	1.6319	1.5453	1.5235	1.5135
SGD	1.2997	1.2997	1.2997	1.2997
Bias-SVD	1.3920	1.3882	1.3662	1.3354
time-svd++	1.1800	1.1829	1.1884	1.1868
SVD++-Test	0.9131	0.9032	0.8952	0.8924
time-SVD++-Test	0.8971	0.8891	0.8824	0.8805

Table 1.4 RMSE Comparison of BMA with other algorithms on Netflix

Additionally, we conducted an experiment to study the speed-up of the algorithm on the Netflix dataset. This is a simple speed-up experiment conducted with Matlab's Parallel Computing Toolbox on a dual socket Intel E7 system with 6 cores on each socket. We collected the running time of the Algorithm 2 to compute the low rank factors $\mathbf{P}$ and $\mathbf{Q}$ with $k = 50$, using 1, 2, 4, 8, and 12 parallel processes. Matlab's Parallel Computing Toolbox permits starting at most 12 Matlab workers for a local cluster. Hence, we conducted the experiment up to a pool size of 12. Figure 1.6 shows the speed-up of Algorithm 2. We observe from the graph that, up to pool size 8, the running time decreases with increasing pool size. However, the overhead costs such as communication and startup costs for running 12 parallel tasks surpasses the advantages of parallel execution. This simple speed-up experiment shows

promising reductions in running time of the algorithm. A sophisticated implementation of the algorithm with low level parallel programming interfaces such as MPI, will result in better speed-ups.

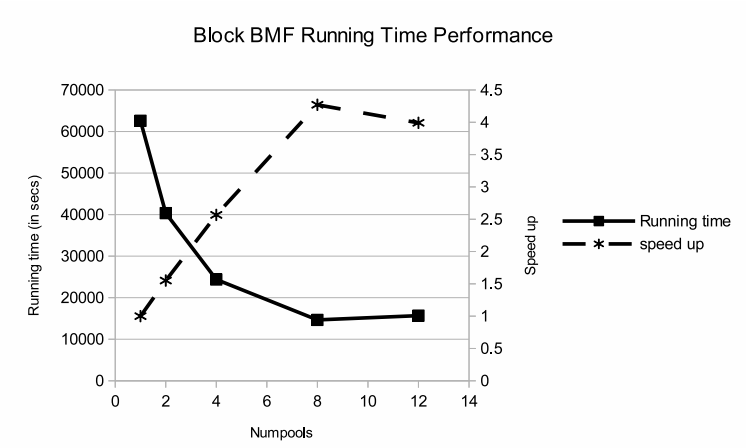

Fig. 1.6 Speed up experimentation for Algorithm 2

In this section, we also present the results of bounding existing ALS type algorithms as explained in Section 1.3.3 and 1.4.3. The performance comparison between ALSWR and ALSWR-Bounded on the Netflix dataset is presented in Table 1.5. Similarly, we also bounded Probabilistic Matrix Factorization (PMF) in Graphchi library. We then compared the performances of both ALSWR and PMF algorithms on various real world datasets, which are presented in Table 1.6.

Algorithm	$k = 10$	$k = 20$	$k = 50$
ALSWR	0.8078	0.755	0.6322
ALSWR-Bounded	0.8035	0.7369	0.6156

Table 1.5 RMSE Comparison of ALSWR on Netflix

1.6 Conclusion

In this chapter, we presented a new matrix factorization for recommender systems called Bounded Matrix Low Rank Approximation (BMA), which imposes a lower and an upper bound on every estimated missing element of the input matrix. Also, we presented substantial experimental results on

Dataset	k	ALSWR Bounded	PMF Bounded	ALSWR	PMF
Jester	10	4.4406	4.2011	4.4875	4.2949
Jester	20	4.8856	4.3018	5.0288	4.4608
Jester	50	5.6177	4.6893	6.1906	4.7383
ML-10M	10	0.8869	0.8611	0.9048	0.8632
ML-10M	20	0.9324	0.8752	0.9759	0.8891
ML-10M	50	1.0049	0.8856	1.1216	0.9052
Dating	10	2.321	1.9503	2.3206	1.9556
Dating	20	2.3493	1.9652	2.4458	1.9788
Dating	50	2.7396	2.0647	2.7406	2.0752
Book Crossing	10	4.6937	5.4676	4.7805	5.4901
Book Crossing	20	4.7977	5.3977	4.8889	5.4862
Book Crossing	50	5.0102	5.2281	5.0018	5.4707

Table 1.6 RMSE Comparison using BALS framework on Real World Datasets

real world datasets illustrating that our proposed method outperformed the state-of-the-art algorithms for recommender system.

In future work we plan to extend BMA to tensors, i.e., multi-way arrays. Also, similar to time-SVD++, we will use time, neighborhood information, and implicit ratings during the factorization. A major challenge of BMA algorithm is that it loses sparsity during the product of low rank factors $\mathbf{PQ}$. This limits the applicability of BMA to other datasets such as text corpora and graphs where sparsity is important. Thus, we plan to extend BMA for sparse bounded input matrices as well. During our experimentation, we observed linear scale-up for Algorithm 2 in Matlab. However, the other algorithms from Graphlab are implemented in C/C++ and take less clock time. A C/C++ implementation of Algorithm 2 would an important step in order to compare the running time performance against the other state-of-the-art algorithms. Also, we will experiment with BMA on other types of datasets that go beyond those designed for recommender systems.

1.7 Acknowledgement

This work was supported in part by the NSF Grant CCF-1348152, the Defense Advanced Research Projects Agency (DARPA) XDATA program grant FA8750-12-2-0309, Research Foundation Flanders (FWO-Vlaanderen), the Flemish Government (Methusalem Fund, METH1), the Belgian Federal Government (Interuniversity Attraction Poles - IAP VII), the ERC grant 320378 (SNLSID), and the ERC grant 258581 (SLRA). Mariya Ishteva is an FWO Pegasus Marie Curie Fellow. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or the DARPA.

References

1. Movielens dataset. `http://movielens.umn.edu`, 1999. [Online; accessed 6-June-2012].
2. D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1999.
3. L. Brozovsky and V. Petricek. Recommender system for online dating service. In *Proceedings of Conference Znalosti 2007*, Ostrava, 2007. VSB.
4. A. Cichocki and A.-H. Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E92-A:708–721, 2009.
5. A. Cichocki, R. Zdunek, and S. Amari. Hierarchical als algorithms for nonnegative matrix and 3d tensor factorization. *Lecture Notes in Computer Science*, 4666:169–176, 2007.
6. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.
7. S. Funk. Stochastic gradient descent. `http://sifter.org/~simon/journal/20061211.html`, 2006. [Online; accessed 6-June-2012].
8. K. Goldberg. Jester collaborative filtering dataset. `http://goldberg.berkeley.edu/jester-data/`, 2003. [Online; accessed 6-June-2012].
9. G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.
10. L. Grippo and M. Sciandrone. On the convergence of the block nonlinear gauss-seidel method under convex constraints. *Oper. Res. Lett.*, 26(3):127–136, Apr. 2000.
11. N.-D. Ho, P. V. Dooren, and V. D. Blondel. Descent methods for nonnegative matrix factorization. *CoRR*, abs/0801.3199, 2008.
12. R. Kannan, M. Ishteva, and H. Park. Bounded matrix low rank approximation. In *Proceedings of the 12th IEEE International Conference on Data Mining(ICDM-2012)*, pages 319–328, 2012.
13. R. Kannan, M. Ishteva, and H. Park. Bounded matrix factorization for recommender system. *Knowledge and Information Systems*, 39(3):491–511, 2014.
14. H. Kim and H. Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*, 23(12):1495–1502, 2007.
15. H. Kim and H. Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.
16. J. Kim, Y. He, and H. Park. Algorithms for nonnegative matrix and tensor factorizations: a unified view based on block coordinate descent framework. *Journal of Global Optimization*, pages 1–35, 2013.
17. J. Kim and H. Park. Fast nonnegative matrix factorization: An active-set-like method and comparisons. *SIAM Journal on Scientific Computing*, 33(6):3261–3281, 2011.
18. Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '08*, pages 426–434, 2008.
19. Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 447, 2009.
20. Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, Aug. 2009.
21. D. Kuang, H. Park, and C. H. Q. Ding. Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of SIAM International Conference on Data Mining - SDM'12*, pages 106–117, 2012.

22. A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: large-scale graph computation on just a pc. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 31–46, Berkeley, CA, USA, 2012. USENIX Association.
23. C. J. Lin. Projected Gradient Methods for Nonnegative Matrix Factorization. *Neural Comput.*, 19(10):2756–2779, Oct. 2007.
24. Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2010.
25. L. W. Mackey, D. Weiss, and M. I. Jordan. Mixed membership matrix factorization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 711–718, 2010.
26. I. Markovsky. Algorithms and literate programs for weighted low-rank approximation with missing data. In J. Levesley, A. Iske, and E. Georgoulis, editors, *Approximation Algorithms for Complex Systems*, pages 255–273. Springer-Verlag, 2011. Chapter: 12.
27. A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of 13th ACM International Conference on Knowledge Discovery and Data Mining - KDD'07*, pages 39–42, 2007.
28. R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, pages 880–887, 2008.
29. L. Xiong, X. Chen, T.-K. Huang, J. G. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the SIAM International Conference on Data Mining-SDM'10*, pages 211–222, 2010.
30. H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Proceedings of the IEEE International Conference on Data Mining-ICDM'12*, pages 765–774, 2012.
31. Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale Parallel Collaborative Filtering for the Netflix Prize. *Algorithmic Aspects in Information and Management*, 5034:337–348, 2008.
32. C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web-WWW'05*, pages 22–32, 2005.