Parallel Multi-View Graph Matrix Completion for Large Input Matrix

Arezou Koohi
Electrical and Computer Engineering
George Mason University
Fairfax,Virginia
akoohi@gmu.edu

Houman Homayoun
Electrical and Computer Engineering
George Mason University
Fairfax,Virginia
hhomayoun@gmu.edu

***Abstract*— We propose a method for parallel multi-view graph matrix completion for the prediction of ratings in recommender systems. The missing ratings are computed based on both the similarity matrix in addition to a rating matrix. The rating matrix is sparse and some items might not have any rating information available. The similarity matrix can be calculated from different item attributes available from ecommerce websites. As the input matrix becomes large, the need for more computationally efficient matrix completion increases. The main contribution of this paper is to show speed-up in calculating the missing ratings by using multi-threaded programming. Simulation results are based on the large input matrix and show reduction in RMSE for the case of cold start prediction.**

***Keywords*— recommender systems, multi-view graph, multi-threaded programming, parallel computing, matrix decomposition**

I. Introduction

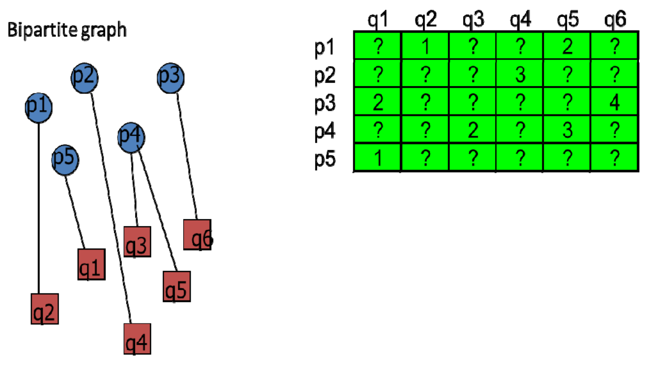
Multiview graph matrix completion is a process of completing the interaction, or rating, matrix based on incomplete observation of data and multiple views of the graph. Examples of an interaction/rating matrix include biological networks and recommender systems. Predictive modeling for an interaction matrix will use interactions between graph nodes to predict new interaction or edges between graph nodes.

Fig. 1 shows an interaction matrix built based on bipartite graph of two different nodes. As is shown in the figure, the objective is to predict values for the incomplete part of matrix that currently don't have any known values, marked as "?".

This graph can be considered one view of nodes Q's or P's. Additional within-node information can be used for matrix completion. Fig. 2 shows the addition of similarity information to the bipartite graph of Fig. 1. This similarity information can be used in case of Multiview matrix completion if within-node similarity information is provided and decided to be used as contributing information.

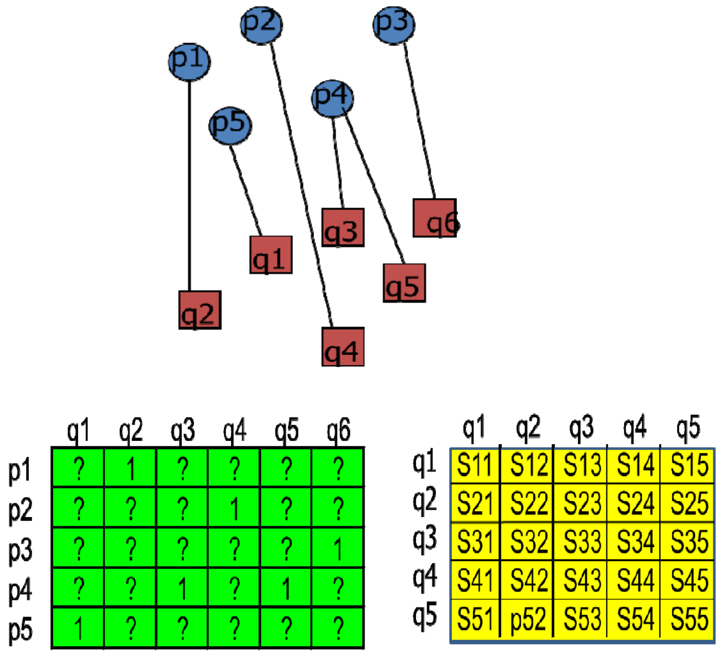
The rating matrix can be considered a weighted, bipartite graph and item or user attributes can be used for the calculation of within-node similarity.

Recommender systems [1] are rating system engines in which predictive models are built based on an incomplete rating of users and items. These matrices are very sparse and often up to 98% or more of ratings are unknown.



	q1	q2	q3	q4	q5	q6
p1	?	1	?	?	2	?
p2	?	?	?	3	?	?
p3	2	?	?	?	?	4
p4	?	?	2	?	3	?
p5	1	?	?	?	?	?

Fig. 1 Bipartite graph and its corresponding weighted interaction matrix



	q1	q2	q3	q4	q5	q6
p1	?	1	?	?	?	?
p2	?	?	?	1	?	?
p3	?	?	?	?	?	1
p4	?	?	1	?	1	?
p5	1	?	?	?	?	?

	q1	q2	q3	q4	q5
q1	S11	S12	S13	S14	S15
q2	S21	S22	S23	S24	S25
q3	S31	S32	S33	S34	S35
q4	S41	S42	S43	S44	S45
q5	S51	p52	S53	S54	S55

Fig. 2 Multiview graph with added within node similarity and its corresponding matrices

Collaborative Filtering [2] is widely used in predictive models for matrix completion, both for Recommender systems and other interaction matrices. One of the famous examples in which collaborative filtering methods are used for a recommender system is the Netflix prize competition [3] in which known ratings from the Netflix movie website were presented to competitors to build a model to predict ratings of other movies. An illustration of the Netflix prize competition is shown in Fig. 3.

978-1-7281-0554-3/19/$31.00 ©2019 IEEE

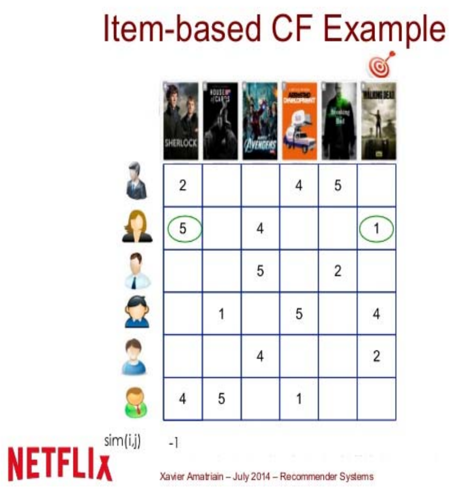


Fig. 3 Netflix completion overview of Collaborative Filtering

One use of within-node similarity information is for cold start prediction, in which the assumed node doesn't have any rating information to be used in the model [4]; as a result, the prediction is not possible for that node based on just rating/interaction matrix.

II. Large Data Matrix Input for Multiview-Graph Matrix Completion

Recommender systems are big part of e-commerce websites like Amazon, Groupon, Living Social, Facebook, and Netflix. With the increasing numbers of users that access these websites in addition to the diverse number of items or advertisement that users receive, recommender systems are essentially for big data predictive modeling. A fast, scalable solution can improve the advertisement and shopping experience, as well as users' movie watching or music listening experience.

If the input matrix is large, using parallel processing nodes can decrease the computational cost of this model. As more data is available commercially with use of social networks and the World Wide Web, it is important to take advantage of multiple processing units.

Since the nature of updates can be sequential, it is required to find independence in the rating matrix in order to make the updating process parallel.

III. Related Work

Use of item to item similarity in addition to a rating matrix in recommender systems is proposed in [4]. The optimization scheme proposed in that paper is used in sequential form and on a smaller matrix input. In comparison, this paper shows a speed up for the optimization scheme using parallel, multi-threaded programming.

Some of related works use parallelization for matrix factorization of large data without considering an item-item similarity matrix. The use of only a rating matrix has allowed these parallelization scheme to achieve data-level independence during parallel execution. [5] used a distributed system like Hadoop for the parallel processing of the data, which can be distributed to different processors. Each processor applies the optimization for independent data sets. [6] shows parallel matrix completion for a large scale matrix input using a multi-threaded scheme. Similar to [5], parallelization is achieved on a data-level by separating the independent user and item features.

By adding item-item similarity regularization the independence in data level will not exist since the similarity matrix, and therefore, similarity features should be accessed by all the processing units. The parallelization can happen at that dimension without added the similarity matrix, which in this case is based on the users' features. The proposed method in this work uses parallelization at the user level to achieve speed up for the case of multi-view matrix completion.

IV. Method

The optimization objective used for the regularized SVD is shown in (1), in which R is the rating/interaction matrix and P and Q are its decomposed matrices. The error function that is being minimized is shown in (2).

$$\|R - PQ^T\|_F + \lambda(\|P\|_F + \|Q\|_F \tag{1}$$

$$(\overline{e_{ui}})^2 = (R_{ui} - \textstyle\sum_k p_{uk} q_{ik})^2 + \lambda(\|p_u\|^2 + \|q_u\|^2) \tag{2}$$

p_u and q_t are the feature vectors for the node u and i. The equation that is being updated is based on the known observations, or ratings R_{ui}, of matrix R.

V. Gradient Descent Vs Stochastic Gradient Descent

Gradient descent can be used for the updating rules of (1). The gradient descent updating rule is as follows:

```
For t = 1 .. T
    $A_t \leftarrow A_{t-1} - \eta\,(1/n) \sum_{i=1}^{n} \partial_A lossfunction(A_{t-1},\, x_i)$
End
```

In which A_t is the parameter we are trying to optimize, t is the iterative step, and x_i is the known training point. For the case of RSVD, A_t is the feature vectors p_u and q_i , and the observation is known rating, R_{ut}.

Gradient descent is slow to converge because all observations are used for each update. When stochastic gradient descent [7] is used instead, only one observation is used in each step.

```
For t = 1 .. T

Draw observation $x_i$ from all the observations.
    $A_t \leftarrow A_{t-1} - \eta \partial_A lossfunction(A_{t-1}, x_i)$
End
```

VI. Stochastic GD for the Case of Regularized SVD

The following updating rules can be used for the case of RSVD. We can draw an observation R_{ut} and update the feature vector p_u and q_t based on that.

$$p_u = p_u + \eta[(R_{ut} - p_u^T q_i)q_i] - \lambda_1 p_u \quad (3)$$

$$q_i = q_i + \eta[(R_{ui} - p_u^T q_i)p_u] - \lambda_2 q_i \quad (4)$$

VII. Updating Rules for Multiview Matrix Completion

The objective function [4] for the case of Multiview matrix completion is shown in (5).

$$\|R - PQ^T\|_F + \lambda(\|P\|_F) + \beta(\|S - QQ^T\|_F) \quad (5)$$

The updating rules for above equation are as follows [4]:

```
While not converge:
For all of rating $R_{ut}$ do
    $p_u = p_u + \eta[(R_{ut} - p_u^T q_t)q_t - \lambda_1 p_u]$   (6)
    $q_i = q_i + \eta[(R_{ut} - p_u^T q_i)p_u] - \beta(q_i \sum_{i'=1}^{M} S_{ii'} - \sum_{i'=1}^{M} S_{ii'} q_{i'})$   (7)
End
End While
```

The above updating rules are sequential, so that every R_{ut} will update the corresponding feature vectors p_u and q_i sequentially. The updated p_u will be used in the next training point which contains the node u. Since each feature update for q will need all other similar items' features according to (7), we cannot parallelize the input data for the item dimension and all the processors should be able to access all the item features.

VIII. Proposed Method

The proposed method will use parallel SGD to update features p. In other words different parallel processors will not have features p in common while each of them will access all of the feature q's. This method makes it possible to access the neighbors of feature q's for the one we are currently updating, according to (7).

Fig. 4 shows the block assignment for the proposed method. The figure shows how training blocks can be processed in each processor. q_i is the feature vector of item i and $\mathbf{i} \in \mathbf{1..n}$, each processor trains all the qi's.

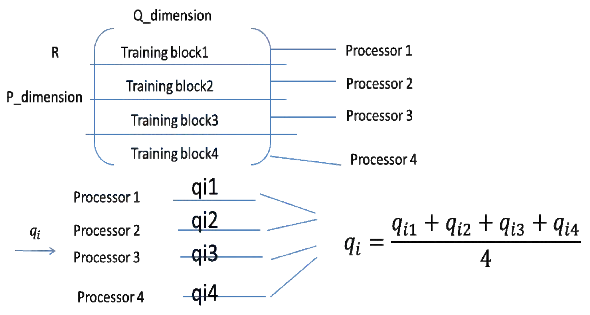


Fig. 4 Overview of proposed method

```
Initialize P and Q feature vector
Divide the training points of matrix of R to k blocks along the dimension of features P.
Sample a training point (rating r) from the block assigned to the processor {do in parallel in K processor}
While not converge do
for b = 1,..., K do /* in parallel */
Update p according to (6)
Update q according to (7)
Average all the q results from all the processors.
Reassign the averaged q to the start of the loop.
End
End While
```

This method uses averaging after each iteration to calculate common item features for the next iteration. Parallelization happens at the p dimension, so each processor will calculate the feature P's that cover the training blocks that are only seen by that processor. The number of processing points can vary and are set to optimize performance.

IX. Data Set

The data set used is music ratings provided by Yahoo [8]. 200,000 users have rated 136,000 songs. In addition to ratings, song attributes are also provided and can be used for calculating the similarity between different songs.

X. Similarity Measurement

The similarity of two songs has been determined based on the song attribute file provided by Yahoo. The song attribute file includes the album, singer or artist, and music genre that the song belongs to. Although the column provided for the music genre includes a lot of unknowns, the other two attributes are completely provided for each song. Equation (7) [4] is used to calculate the similarity between two songs based on their corresponding singers and albums that they belong to. The song genre is not included in this calculation since the data provided on that was mainly incomplete. It is also more specific to measure the similarity based on the song's album and artist. a_{ij} and $a_{i'j}$ are the attributes of items i and i', for example in this case they are singer and albums names.

$$sim(i, i') = \frac{M(a_{ij}, a_{i'j})}{Total\ number\ of\ attributes} \quad (7)$$

$$M(a_{ij}, a_{i'j}) = \begin{cases} 1\ if\ a_{ij} = a_{i'j} \\ 0\ else \end{cases}$$

XI. RESULTS

There were 76,000,000 ratings available for the input matrix. Fig. 5 shows the RMSE on the test data for 2,000,000 test ratings provided by Yahoo for the users and the songs used in the training set. The test data set includes 10 song ratings for each user. The figure shows the comparison of the results between parallel and sequential runs. Both methods achieve the same RMSE. Using the parallel method has made the runtime 3.2 times faster. 5 threads are created in a C++ program environment in parallel to achieve this speed up. The top 10 most similar items are used in the similarity regularization term and β is set to 0.1.

In order to test the effect of a similarity view in predictive modeling, we eliminate ratings for 10% of songs during the training phase. Fig. 6 shows that by adding the similarity view through the regularization term according to (6), we can reduce the RMSE considerably for the case of large data input matrix. We have simulated the results for including different numbers of similar items in the regularization term. The simulation included the 5, 10 and 15 most similar items. Fig. 7 and Fig. 8 show RMSE for different numbers of similar items included. RMSE was the lowest for KNN=10.

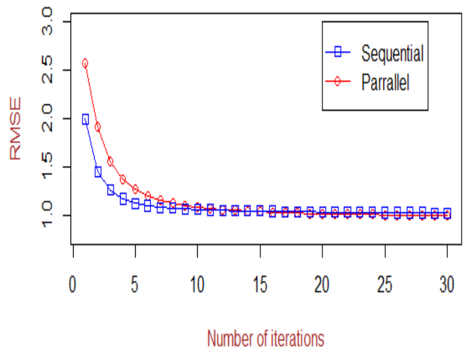


Fig. 5 RMSE results for parallel and sequential

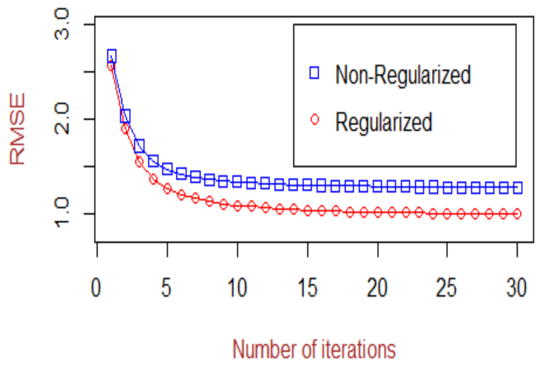


Fig. 6 RMSE results for the cold-start prediction

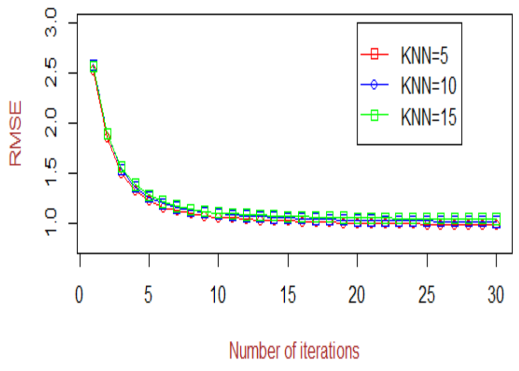


Fig. 7 RMSE results for different number of similarity items included (a)

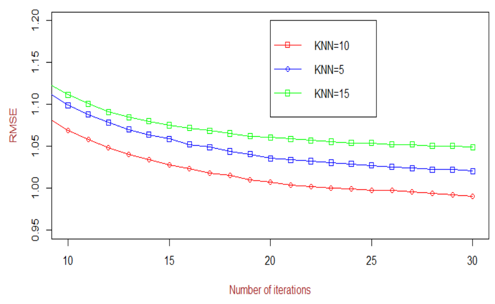


Fig. 8 RMSE results for different number of similarity items included (b)

XII. CONCLUSION

As the number of users on different social and commercial websites increases, the need for scalable predictive computational model has also increased. This work proposes a parallel method for the rating prediction of recommender systems. It uses multi-threaded programing in a C++ environment in order to achieve the parallel processing. It shows much shorter execution time to compute the missing ratings of a large matrix by using 5 threads. The ratings are calculated based on both a rating matrix as well as a similarity matrix.

One of the advantages of using the similarity matrix is to find the rating prediction for the items without any prior known ratings in the rating matrix [4]. This case is called cold start prediction. The simulation results on the Yahoo music data set show that, by using the similarity matrix, the RMSE reduces for the case of cold start prediction of a large matrix input. This work has shown the advantage of using multi-threaded programming in a C++ environment for the speed up. In the future other parallel programming schemes like GPU can also be used. We have used the shared memory approach in this work by using a server memory space. If the training data is too big for shared memory, a distributed file system like HDFS can be considered.

References

[1] P. Melville and V. Sindhwani, "Recommender systems," *Encyclopedia of Machine Learning,* pp. 829-838, 2010.

[2] X. Su and T. Khoshgoftar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence ,* vol. 2009, no. 4, 2009.

[3] "Netflix prize," Netflix, [Online]. Available: https://www.netflixprize.com/index.html. [Accessed 1 June 2018].

[4] Y. Yu, C. Wang and Y. Gao, "Attributes coupling based item enhanced matrix factorization technique for recommender systems," *IEEE Transactions on Knowledge and Data Engineering,* vol. 46, no. 3, p. 15, 2014

[5] R. Gemulla, E. Nijkamp, P. J. Haas and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *17th ACM SIGKDD nternational Conference on Knowledge Discovery and Data Mining*, New York.

[6] Benjamin Recht and C. Ré, "Parallel stochastic gradient algorithms for large-scale matrix completion," *Mathematical Programming Computation,* vol. 5, no. 2, 2011.

[7] L. Bottou, "Stochastic gradient descent," *Neural Networks: Tricks of the Trade,* vol. 7700, Part of the Lecture Notes in Computer Science book series.

[8] "Yahoo research outreach dataset," Yahoo, [Online]. Available: https://webscope.sandbox.yahoo.com/catalog.php?datatype=r. [Accessed 1 June 2018].