Scaling Collaborative Filtering to large–scale Bipartite Rating Graphs using Lenskit and Spark

Christos Sardianos
Department of Informatics and Telematics
Harokopio University of Athens
Athens, Greece
Email: sardianos@hua.gr

Iraklis Varlamis
Department of Informatics and Telematics
Harokopio University of Athens
Athens, Greece
Email: varlamis@hua.gr

Magdalini Eirinaki
Computer Engineering Department
San Jose State University
San Jose, CA, USA
Email: magdalini.eirinaki@sjsu.edu

***Abstract*—Popular social networking applications such as Facebook, Twitter, Friendster, etc. generate very large graphs with different characteristics. These social networks are huge, comprising millions of nodes and edges that push existing graph mining algorithms and architectures to their limits. In product-rating graphs, users connect with each other and rate items in tandem. In such bipartite graphs users and items are the nodes and ratings are the edges and collaborative filtering algorithms use the edge information (i.e. user ratings for items) in order to suggest items of potential interest to users. Existing algorithms can hardly scale up to the size of the entire graph and require unlimited resources to finish. This work employs a machine learning method for predicting the performance of Collaborative Filtering algorithms using the structural features of the bipartite graphs. Using a fast graph partitioning algorithm and information from the user-friendship graph, the original bipartite graph is partitioned into different schemes (i.e. sets of smaller bipartite graphs). The schemes are evaluated against the predicted performance of the Collaborative Filtering algorithm and the best partitioning scheme is employed for generating the recommendations. As a result, the Collaborative Filtering algorithms are applied to smaller bipartite graphs, using limited resources and allowing the problem to scale or be parallelized.**

Tests on a large, real-life, rating graph, show that the proposed method allows the collaborative filtering algorithms to run in parallel and complete using limited resources.

***Keywords*-Recommender Systems; Collaborative Filtering; Graph Partitioning; Graph Metrics; Social Networks;**

I. Introduction

The traditional recommender systems have been examined thoroughly during the past decade and received a wider acceptance from e-commerce and content-rich sites (e.g. news sites). Recommender systems are primarily based on ***collaborative filtering (CF)*** techniques in order to provide a personalized aspect to content delivery, tailored to each user's preferences. In social networking applications, such systems recommend users, content, or both, through the confluence of user preferences (likes, views, follows, etc) social network-derived data (such as structure). The premise in CF is to generate recommendations taking into consideration the preferences (typically expressed as ratings) of the user's closest neighbors. The neighbors of a user can be found based on: a) the similarity between users' profile features such as demographics etc, b) the items that users have reviewed in common and the ratings they have provided for them, c) the proximity of a user with other users in the social graph. These inputs can be used individually or combined.

The main challenges for social recommender systems that employ the social network structure to enhance the recommendation process, are: a) to take advantage of all the available information in order to analyze the social, rating and content similarity graphs that are formed, b) to adapt to dynamically evolving graphs, and c) to scale to large graphs. This work focuses on the problem of *scaling the CF solution to graphs that contain several millions of nodes and edges by properly partitioning the ratings graph*.

The two main alternatives for scaling to large graphs are to either upgrade the infrastructure in order to cover the increasing processing requirements, or to partition the graph into smaller subgraphs which can be processed in parallel, thus increasing performance without loosing in quality of the produced results. Since infrastructure upgrades always have an upper limit in the number of resources that can be used in this work, we focus on the second alternative. In other words, we split the initial problem into sub-problems that can be solved more efficiently using parallel algorithms, without loss of the effectiveness of the provided solutions (measured by the quality of the generated recommendations).

We leverage our previous work on social graph partitioning [1], which assumes that the users of a bipartite rating graph are also connected in a unipartite network with friendship or trust edges. Taking advantage of user friendship information, we first partition the unipartite graph (containing only the user nodes and user-to-user edges) and then the bipartite graph (including user and item nodes and user-to-item edges only), as depicted in Figure 1. Once the social graph is partitioned all users that belong in the same partition are examined as a separate social network. The users of a social graph partition carry the ratings that

they have provided for items thus leading to a partitioning of the bipartite graph too. The CF algorithm processes the bipartite subgraphs in a separate process and generates recommendations.

Although graph partitioning is a fast process, collaborative filtering is quite slow and resource-demanding for large graphs. So it is preferable to find the best partitioning scheme by testing different alternatives and run the collaborative process once on this scheme. In order to evaluate our proposed method, we partition the original graph in various number of partitions, examine several features of the resulting subgraphs and decide on the optimal partitioning of the original graph. The decision for the optimal partitioning is based on a prediction of the performance of the collaborative filtering algorithm in each bipartite subgraph. For this prediction, we use several structural features of the bipartite graph and train a machine learning model that predicts the CF algorithm performance. The features aim to capture the amount of common ratings between users or between items, which affect the quality of recommendations, since CF algorithms use common ratings to measure user or item similarity and recommend similar items to similar users.

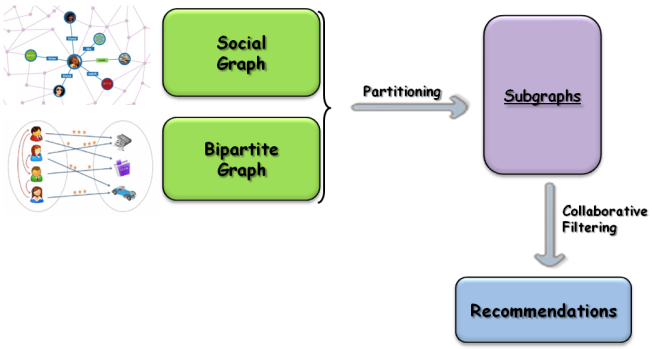

Figure 1. Graph partitioning for recommendations.

The main contributions of this work, rely on:

- the prediction of the performance of CF algorithms in each partition of the graph, using only the structural features of each graph partition. This is a critical step in the graph partitioning process as, the ability to predict the performance of the recommendation process without running the CF algorithm allows us to test different graph partitioning schemes and select the optimal scheme.
- the ability to scale up to huge graphs, with limited resources. The partitioning of the ratings' graph to subgraphs based on the social network information, results in many, smaller graphs that can be handled using limited resources.
- the execution of the CF algorithm in a parallel and/or distributed setup. The proposed system uses the CF algorithm implementations provided by Lenskit[1], which are single threaded and executes them over Apache Spark[2] thus allowing task parallelization.

The rest of the paper is organized as follows: In Section 2 we summarize related work. In Section 3 we present our proposed method for predicting the performance of CF in the subgraphs using graphs' structural features. In Section 4, we present the results of the experimental evaluation we performed on a dataset from Epinions and in Section 5 we provide the conclusions and the next steps of our work.

II. Related Work

Over the past years, the success of social networking applications and the integration of item review sites with social networks led to a holistic approach in recommender systems. This line of work is based on the assumption that a user's preferences are influenced more by these of their connected friends, than these of unknown users [2], rooted in the sociology concepts of homophily and social influence [3]. Tang et al. [4] give a narrow definition of social recommendation as "any recommendation with online social relations as an additional input, i.e., augmenting an existing recommendation engine with additional social signals" (a broader definition, not applicable to this work, refers to recommender systems targeting social media domains [5]).

In social recommender systems, the recommendation engine considers two graphs in parallel; a social graph, which contains edges between users, and a bipartite graph, which contains user ratings for items (e.g. products, or content provided by users). The social graph is used either for improving the quality of recommendations or for reducing the space for the recommendation algorithm by ignoring the preferences of users that are not near-neighbors of the target user.

Pham et al. in [6] propose a method that groups together users based on their social information and prove that they can perform better than traditional techniques in generating recommendations. Similarly, in [7] the authors suggest a combination of CF techniques [8] and techniques that factorize the user's social friendship graph. The authors in [9] use spectral Graph Partitioning Techniques on the social graph in order to select the neighborhood of a user and then apply User-Based Collaborative Filtering. Recently, authors in [10] implemented a multi-way spectral clustering approach, utilizing the top few eigenvectors and eigenvalues of the normalized Laplacian matrix to compute a multi-way partition of the data. A comprehensive survey of implementations that partition the original rating graph and perform parallel and/or distributed collaborative filtering is presented in [11].

In our previous work [1], we showed that graph partitioning can significantly improve the time performance of

[1]http://lenskit.org/
[2]http://spark.apache.org/

the algorithms without affecting recommendation quality. According to [12] the performance of the recommendation algorithms is strongly connected to structural characteristics of the resulting graph partitions, more specifically graph sparsity and entropy (which is defined as the distribution of co-ratings). In this article, we show that the sparsity and entropy of the bipartite graph are not always good indicators of the CF algorithm performance and propose a set of metrics (graph and node specific), which improve prediction performance. Based on performance predictions, we are able to compare CF algorithm performance of two or more partition schemes and decide on the optimal number of partitions before even applying the recommendation algorithm which is a time consuming process. This allows us to decide beforehand whether performing graph partitioning is worthy or not and to choose the best number of graph partitions that will lead to high quality recommendations and parallelization of the recommendation problem. The proposed solution has been implemented on Apache Spark, and employs Lenskit recommender toolkit for creating recommendations.

III. Bipartite Graph Partitioning based on Social Graph information

The main premise of CF is that a user u will be interested in items that other users with similar profiles (i.e. users that have rated many items in common with u and with similar scores) have rated. When a social graph is also available the social information is used to partition the user-item bipartite graph into smaller subgraphs that can be handled more efficient by the recommender (CF) algorithm. The intuition behind this type of partitioning (i.e. before the application of CF) is that a user will consider more seriously the ratings provided by his/her friends than the ratings of any other user. We should note at this point that any other methodology for partitioning the bipartite graph can be employed. The main components of this social recommender system process are depicted in Figure 2 and detailed in the subsections that follow.

More formally, we define the social graph G_{social}, which contains edges E_s between users V_u and the bipartite graph $G_{bipartite}$, which contains user ratings R for items V_i, as:

$$G_{social} = \{V_u, E_s\}, E_s : undirected, unweighted \quad (1)$$

$$G_{bipartite} = \{V_u, V_i, R\}, R : directed, weighted \quad (2)$$

A. Graph Partitioning

The first step includes the partitioning of the social graph, which is then used for the partitioning of the bipartite graph. The partitioning of the social graph aims to find groups of users that are socially connected (e.g. friends or followers) and can be performed using an appropriate graph partitioning or clustering algorithm. An example of a social and a bipartite graph is presented in Figure 3.

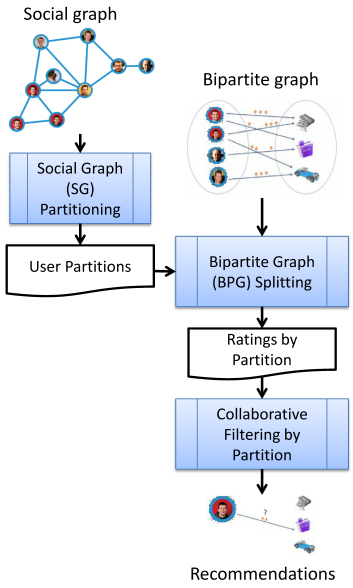

Figure 2. The pipeline of the recommender system process.

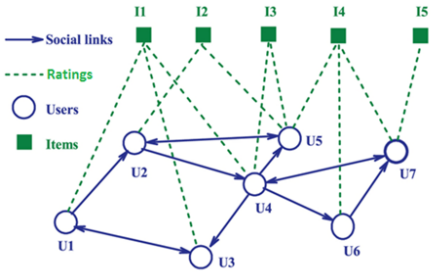

Figure 3. The two graphs of a product-rating social network.

The partitioning of the social graph is followed by the partitioning of the bipartite graph. If for example the social graph of Figure 3 is partitioned in two subgraphs U_1, U_2, U_3, U_5 and U_4, U_6, U_7, then the corresponding bipartite partitions will contain the respective users and their provided ratings (denoted as blue and red edges respectively in Figure 4).

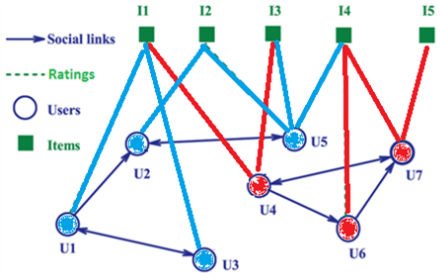

Figure 4. Partitioning the bipartite graph based on the partitioning of the social graph.

The outcome of the Graph Partitioning step is a set of

subgraph pairs (social - bipartite), namely $UserPartitions$ and $RatingsByPartition$, that contain all the i subgraphs of G_{social} and $G_{bipartite}$ defined as:

$$G_{social_i} : \begin{cases} V_u = \cup_{i=1..N} V_{S_i} \\ \forall i,j, i \neq j \rightarrow V_{S_i} \bigcap V_{S_j} = \emptyset \end{cases} \quad (3)$$

$$G_{bipartite_i} : V \in V_u \begin{cases} V \in V_{S_i} \\ e(us,it) : e \in E, us \in V_{S_i}, it \in V_i \rightarrow \\ G_{bipartite_i} = \{V_{S_i}, V_{i_{pointedV_{S_i}}}, E\} \end{cases} \quad (4)$$

Algorithm 1 depicts the two graph partitioning processes, namely the social graph (G_{social} or SG) partitioning algorithm (lines 1-14) and the bipartite graph ($G_{bipartite}$ or BG) partitioning algorithm (lines 15-22). Given the social graph $SG\{V, E\}$, for every edge $e_{V_p,V_q} \in E$ we get the partition that vertices V_p and V_q belong to, after the partitioning process. If V_p and V_q belong to the same partition then their social edge E is included in the set of edges of this partition. Otherwise, this edge is moved to a set of social edges that connect different partitions, which we call *Cross-Partition Edges* set. During the bipartite graph partitioning process, every item rating edge $e\{V_u, V_i, R\} \in E$ the bipartite graph $BG(V, E)$, is assigned to the same subgraph with all item rating edges of user V_u and all users clustered together with V_u in the previous step.

The output of the graph partitioning process includes all the bipartite graph partitions, which are then used as input to the Recommendation Evaluation process. The next step involves the application of the CF algorithm to each bipartite subgraph in order to generate recommendations.

Algorithm 1 Graph partitioning algorithm

1: **procedure** SPLITSG(SG{V,E}, Cross, SubSGs)
2: $\{S_sG\{V, E\}\}$ = **Partition**$(SG\{V, E\})$
3: $Cross = \emptyset$
4: **for** `e in E` **do**
5: V_i = e.from
6: V_j = e.to
7: S_sG_{From} = **GetSubgrForVertex**$(V_i, \{S_sG\{V, E\}\})$
8: S_sG_{To} = **GetSubgrForVertex**$(V_j, \{S_sG\{V, E\}\})$
9: **if** $(S_sG_{From} = S_sG_{To})$ **then**
10: $S_sG_{From} = S_sG_{From} \cup \{e\}$
11: **else**
12: $IntClust$ = **InterClusterEdge**$(e, S_sG_{From}, S_sG_{To})$
13: $Cross = Cross \cup \{IntCluster\}$
14: $SubSGs = \{S_sG\{V, E\}\}$
15: **procedure** SPLITBG$(BG\{V_u, V_i, R\}, SubBGs)$
16: $\{B_sG\{V_u, \emptyset, \emptyset\}\}$ = **PartitionUsers**$(SubSGs)$
17: **for** `e in` R **do**
18: $V'_u = e.from$
19: $B_sG\{V'_u, V'_i, R'\}$ = **GetSubgrForVertex**$(V'_u, SubSGs)$
20: $V'_i = V'_i \cup e.to$
21: $R' = R' \cup \{e\}$
22: $SubBGs = \{B_sG\{V'_u, V'_i, R'\}\}$

B. Collaborative Filtering

This subsection provides an overview of the most popular CF algorithms, which employ user or item similarity calculated over their common ratings. The partitioning of the original graph to smaller subgraphs allows a recommender system to run in a parallel setup: each bipartite subgraph is processed in a separate thread, thus reducing the total execution time compared to the single thread alternative. Usually CF is a memory-intensive process, where the graph information is loaded in memory in a sparse adjacency matrix format. By splitting the graph to subgraphs, the memory requirements for each thread are significantly smaller, and it is feasible to run CF even on extremely large graphs. In our previous work [1] we have shown that CF cannot be directly applied to very large graphs using popular CF algorithms and software in a commodity server.

The most popular CF method, $user\text{-}based$ collaborative filtering [13] is a straightforward algorithmic interpretation of the core premise of collaborative filtering: "*find other users whose past rating behavior is similar to that of the current user and use their ratings on other items to predict what the current user will like*". Besides the user ratings, a $user\text{-}based$ CF algorithm requires a similarity function and a prediction method to predict rankings and generate recommendations. User similarity can be the statistical correlation between two user's common ratings (Pearson's coefficient), the respective Spearman rank correlation coefficient of users' ranked items, or a cosine similarity between the users' rating vectors (users' ratings for all possible items).

$User\text{-}based$ collaborative filtering, while effective, suffers from scalability problems as the user base grows (as it happens with all k-NN algorithms) [14]. $Item\text{-}based$ collaborative filtering provides a more scalable solution, even when limited ratings are present. Rather than using similarities between users' rating behavior to predict preferences, $item\text{-}based$ CF uses similarities between the rating patterns of items. If two items tend to have the same users like and dislike them, then they are considered similar. Once again, item similarity is computed using cosine similarity over the ratings, Pearson correlation etc.

Finally, since the dimensions of both the User and the Item space can be large, there are algorithms that perform dimensionality reduction before computing similarities. SVD-based dimensionality reduction has been widely adapted to collaborative filtering [15] and is by far the most successful CF method. In $SVD - based$ CF, the original user-item

matrix is decomposed to a matrix that maps users to a set of fewer dimensions ($userxfeature$), a transposed matrix that maps items to the same dimensions ($featurexitem$) and a diagonal matrix that contains the singular values of the decomposition. Then the predicted preference of any user u for any item i can be computed as the weighted dot product of the user-topic interest vector and the item-topic relevance vector. Singular value decomposition, Principle component analysis or any other Matrix Factorization method can be used for reducing the dimensionality of the user or item similarity problem [16].

C. Evaluation of recommendations

The quality of recommendations produced by a CF algorithm or any other recommender system is usually evaluated against some known preferences (user ratings) which have been hidden by the system during the training phase. One of the most commonly used metrics is Mean Absolute Error (MAE), defined in Equation 5.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}| \tag{5}$$

where y_i is the actual rating, $\hat{y_i}$ is the predicted rating for all the n ratings that have been used for evaluation. MAE can be averaged by user first and then get an average for all users (MAE_ByUser or Micro-Average) or can be directly averaged for all ratings (MAE_ByRating or Macro-Average).

An even more commonly used metric is the Root Mean Square Error (RMSE), defined in Equation 6, which is also averaged by user (RMSE_ByUser or Micro-Error) or for all ratings (RMSE_ByRating or Macro-Error).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2} \tag{6}$$

RMSE and MAE measure how well the algorithm predicts the user ratings. In the case of top-n recommendations, we often need to evaluate the generated rankings. The most commonly used metric for this purpose is the normalized Discounted Cumulative Gain (nDCG) [17] evaluates the predicted ranking of recommended items against their actual ranking in user preferences etc. Cumulative Gain (CG) (Equation 7) accumulates the graded relevance (predicted vs actual rating) of the top-k recommendations.

$$CG_k = \sum_{i=1}^{k} rel_i \tag{7}$$

where rel_i is the graded relevance of item in position i of the recommendations list.

Discounted cumulative Gain (DCG) penalizes highly relevant items that are ranked lower in the list of recommendations as shown in Equation 8.

$$DCG_k = rel_1 + \sum_{i=2}^{k}\frac{rel_i}{log_2(i)} \tag{8}$$

Finally, normalized Discounted cumulative Gain (nDCG) (Equation 10) divides DCG to the maximum possible DCG (IDCG, see Equation 9), which is achieved when only relevant items are recommended to the user and in decreasing order of actual ratings.

$$IDCG_k = \sum_{i=1}^{|REL|}\frac{2^{rel_i}-1}{log_2(i+1)} \tag{9}$$

where —REL— is the list of all relevant items for the user (e.g. items with positive ratings or ratings above a threshold).

$$nDCG_k = \frac{DCG_k}{IDCG_k} \tag{10}$$

D. Finding the best graph partitioning

In order to have qualitative recommendations, it is important to choose the optimal bipartite partitioning scheme, so that the bipartite subgraphs contain all the information needed for recommending the correct items to users. For example, if we choose too many partitions, the subgraphs have only a few users who have rated a small portion of the total items and possibly good recommendations for them, are in a different partition. If we choose a scheme with few but large partitions, we will not be able to determine whether the CF algorithm will finish with the available resources.

This work proposes an indirect evaluation of CF performance, by predicting and evaluating the performance of the CF algorithm on different partitioning schemes of the bipartite graph. The performance of CF algorithms is strongly related to the similarity between users or items, which in turn is affected by the overlap (amount of common ratings) between users or items, as explained in Section III-B. Based on this principle, we extract several structural features for each subgraph in each partition scheme, which aim to capture the rating overlap between users and items and thus help us predict CF algorithm performance. We extract these features from bipartite graphs of known CF performance (i.e. we have executed the CF algorithms on these graphs and have evaluated their recommendations), and with this information we train a supervised model, which is able to predict the quality of recommendations in each rating subgraph and consequently in each partition scheme. Then we choose the optimal partitioning scheme based on predictions and run the CF algorithm in each subgraph of the scheme, taking advantage of task parallelization. The overall process is depicted in Figure 5.

The large bipartite graph is partitioned several times, with a fast graph partitioning algorithm (Metis [18]), giving different partition schemes. Starting with a large number of

partitions (n) and following a branch and bound strategy allows us to find a performance maximum quickly. However, any other strategy can be applied instead for finding overlapping or non-overlapping partitions.

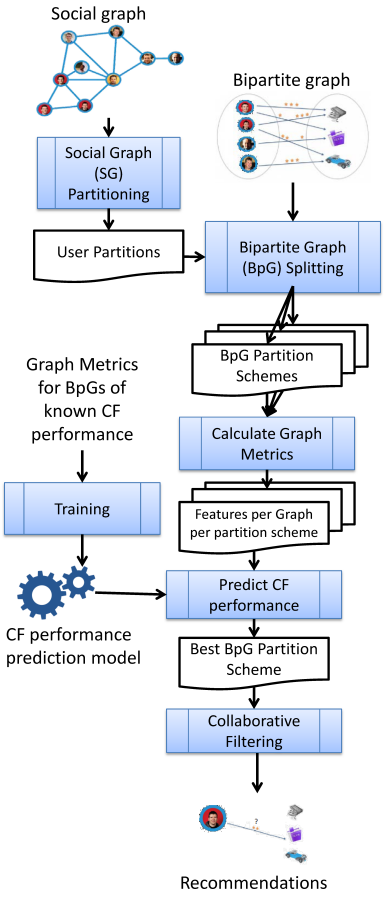

Figure 5. The complete workflow of the proposed recommender system.

IV. Structural features of bipartite graph partitions

According to previous work of Matuszyk and Spiliopoulou [12], it is possible to predict the performance of a CF algorithm on a bipartite graph using a linear regression model that takes into account the structural characteristics of the graph. More specifically, the authors have shown that the performance of the CF algorithm is linearly correlated to sparsity, Entropy and the Gini Index of the graph. In this work, we evaluate this assumption in similar bipartite subgraphs, produced by the proposed partitioning algorithm, but also extend the set of features in order to improve the prediction performance. Since CF is based on the overlap (or similarity) between user (and item) provided ratings, we measure the size of this overlap as well as node related features (nodes can be either users or items in a bipartite graph) that capture node centrality.

We have employed two types of graph metrics: a) metrics that give a single value (or set of values/features) for the subgraph as a whole (e.g. graph sparsity), and b) metrics that give a different value for each node in the subgraph (e.g. node clustering coefficient, pagerank, eigenvalue centrality etc). In the first case, the value (or set of values) is the feature that will help us predict the performance of the CF algorithm. In the second case, instead of using the average value for the nodes of each graph partition, we employ the percentile values for the 10 percentile levels computed on the set of vertices (user or item vertices) of the graph. Since some of the features are not defined on bipartite graphs, we slightly modify their definition as explained in the following.

The first feature that we employ is the sparsity of the graph, which represents the ratio of edges against all possible edges and for a graph G and is defined as:

$$Sparsity(G) = 1 - \frac{\sum_{u \in U} |R(u)|}{|U| \cdot |I|} \tag{11}$$

where U is the set of Users and I the set of Items.

The second feature is the Gini Index of the graph, which quantifies how uniform is the distribution of co-ratings among classes of users with similar number of ratings. The Gini Index for a graph G is defined as:

$$Gini(G) = 1 - \sum_{x,y} \left(\frac{cor([u_x],[u_y])}{\sum_{x,y} cor([u_x],[u_y])} \right)^2 \tag{12}$$

where $\mathcal{U} = \{|u| : u \in U\}$ is the set of equivalence classes over the set of users U, in which all users have provided the same number of ratings. Consequently $[u_x]$ and $[u_y]$ represent all users belonging in different user classes with and x and y ratings per user in each respective class. Gini is computed upon average number of co-ratings of user classes, where a uniform distribution is ideal i.e. all classes have the same number of co-ratings. Finally, $cor()$ is the correlation function between the two distributions.

In addition to Gini Index, authors in [12] propose the similar metric of Entropy, which is defined as:

$$E(G) = -\sum_{x,y} \frac{cor([u_x],[u_y])}{\sum_{x,y} cor([u_x],[u_y])} log_2 \left(\frac{cor([u_x],[u_y])}{\sum_{x,y} cor([u_x],[u_y])} \right) \tag{13}$$

Following the concepts elaborated in [19] and [20], we implement additional metrics that capture the local density of the bipartite graphs. For this reason, we define the clustering coefficient for any node v of degree at least 2, a metric that computes the probability, for any given node chosen at random, that any two of its neighbors are linked together. In the case of a bipartite graph, it is not feasible for the neighbors (e.g. items/users) of a node (e.g. user/item) to be linked together, so the Clustering Coefficient of a node in

the bipartite graph is defined as the average of its clustering coefficients with other nodes:

$$BCC_{\bullet(u)} = \frac{\sum_{i \in I} cc_{\bullet}(u,i)}{|I|} \quad (14)$$

where u corresponds to each user and i to each item, and they can switch position in the formula in order to compute the *bipartite clustering coefficient (BCC)* for all nodes.

In order to capture the local density of the subgraphs, we compute the Triad Count Statistics [21] of the subgraph, which total to 16 different values for a normal directed graph. In the case of bipartite graphs we end up with only 4 features that produce non-zero values.

Finally, the $EigenvectorCentrality$ and $PagerankCentrality$ scores of each node in the bipartite graph are computed. The directed edges of the bipartite graph do not allow Pagerank and Eigenvector Centrality to differ significantly among nodes, so the directed edges are assumed to be undirected. Once again we distinguish between item and user nodes. Centrality values for nodes are sorted in decreasing order and the 10 percentile values (i.e. the centrality score which is bigger than 10% of other scores is the 1st percentile, etc.) are used as features. This results to 10 Pagerank percentile features for user nodes, another 10 for item nodes and 20 percentile features for Eigenvector centrality. For the Bipartite Clustering Coefficient the node values for all nodes are ranked, without distinguishing between user and item nodes. Thus we result with a set of 57 structural features for each graph (7 graph features and 50 percentile features from 3 node metrics) and the target features that we want to predict. The candidate target feature can be any of the MAE, RMSE or nDCG.

As depicted in Figure 5, the performance prediction model have been trained using a set of bipartite graphs, for which we have run all three CF algorithms and have evaluated the recommendations quality (output feature). For these graphs we computed the set of aforementioned structural (input) features and trained the supervised model. Then on run-time, we are able to predict the performance of CF algorithms (output feature) on any subgraph of a partition scheme using the calculated graph metrics as input features. The average CF performance over all subgraphs is an indication of a good or bad partition scheme.

V. Experimental results

To evaluate the performance of our proposed approach we used a dataset from University of Koblenz-Landau[3] (Oct. 2013) that contains social network information and rating information from users. This is also known as the Epinions dataset –Epinions is one of the largest consumer product reviews site– which contains 131,828 unique users with 841,372 social relations and 13,668,320 ratings for 755,760 distinct items. The details are depicted in Table I.

Table I
Social and Bipartite graph characteristics

Graph Characteristics		
Social Graph	Num. of Distinct Users	131,828
	Num. of Social Edges	841,372
	Average Degree	12.765
Bipartite Graph	Num. of Distinct Users (raters)	120,492
	Num. of Distinct Items	755,760
	Num. of Ratings	13,668,320
	Avg. outDegree/User	113.44
	Avg. inDegree/Item	18.09

A. Evaluation of the regression model

Since we are interested in the ability of our feature set to predict the performance of the CF algorithm we use Pearson correlation coefficient to measure the correlation between the input variables and the target variable. If there is a strong correlation then it will be possible to use the same predictor for any partitioning of the large graph G into subgraphs. Based on the output of our predictor, we can the choose the best (most promising) partitioning and proceed with the recommendation algorithm.

For the evaluation we choose to train our machine learning model using subgraphs from different partitioning schemes of the original graph up-to 1000 partitions, so that we have a fairly large evaluation set while the subgraphs still contain a lot of nodes and edges. According to the statistics of Table I the average number of user nodes in each partition is 130 and the average number of items is 750.

Based on the preliminary work of Matysyzh [12], we evaluate our predictions using only the three *Graph Features* for the whole graph as suggested by the authors (i.e. Sparsity, Gini and Entropy), then we compare with the node based value-distribution features that we introduced in this paper (*Node Features*), as well as with the full set of graph and node features. We use Linear regression with feature selection and Singular Value Decomposition for the prediction of the target feature. For these two popular regression techniques, we employed the Java implementation provided in Weka data mining suite. More specifically, we employed the Linear Regression algorithm using M5 attribute selection technique (step-wise removal of attributes with the smallest standardized coefficient until their is no improvement in the estimation of the Akaike information criterion) and the SMO algorithm (a sequential minimal optimization algorithm for training a support vector classifier).

Linear Regression - LR and a Gaussian Radial Basis Function network -RBF [22] are used for the prediction of CF performance for the three main collaborative filtering

[3]http://konect.uni-koblenz.de/networks/

Table II
PEARSON CORRELATION BETWEEN PREDICTED AND ACTUAL CF ALGORITHM PERFORMANCE.

		3 Feat. (LR)	All feat. (LR)	3 Feat. (RBF)	All feat. (RBF)
User-User	RMSE	0.286	0.462	0.318	0.470
	RMSE by user	0.165	0.340	0.181	0.358
	nDCG	0.087	0.503	0.269	0.510
	MAE	0.240	0.450	0.257	0.460
	MAE by user	0.187	0.404	0.224	0.452
Item-Item	RMSE	0.245	0.384	0.268	0.382
	RMSE by user	0.078	0.231	0.081	0.228
	nDCG	0.076	0.502	0.270	0.507
	MAE	0.235	0.395	0.230	0.404
	MAE by user	0.209	0.383	0.217	0.410
SVD	RMSE	0.241	0.467	0.298	0.476
	RMSE by user	0.155	0.363	0.186	0.389
	nDCG	0.136	0.524	0.293	0.569
	MAE	0.240	0.492	0.276	0.509
	MAE by user	0.189	0.435	0.239	0.479

techniques (i.e. user-based, item-based and matrix factorization). For each technique 5 different CF performance metrics are examined (RMSE per user, overall RMSE, MAE per user and overall MAE, and nDCG). All experiments are repeated 100 times using a random 90%-10% training-test split. The average values are depicted in Table II[4].

From the results in Table II we can draw the following conclusions:

- Linear regression with feature selection predicts the performance of CF algorithms worse than the RBF network alternative.
- The correlation values resulting from the three initial graph features (Sparsity, Gini and Entropy) are worse than those resulting from all features and much worse than those reported in [12] (the reported correlation for RMSE was above 95%).
- The prediction of nDCG value is the most successful, making the predicted nDCG score a criterion for selecting the optimal partition in future setups.

The correlation between the predicted and the actual values of nDCG for the 1000 partitions using the three collaborative filtering algorithms is depicted in Figures 6, 7 and 8. The horizontal axis contains the graph id (the graphs are sorted in a decreasing order of the actual nDCG value for the CF algorithm) and the vertical axis contains the nDCG value. The red line corresponds to the actual nDCG of each graph and the blue line to the predicted nDCG. It is obvious from the results of all CF algorithms that a correlation above 0.5 is helpful and the predicted nDCG values can be an evidence for deciding on the best partitioning scheme.

[4]All results are within a ± 0.02 range at 99% CI.

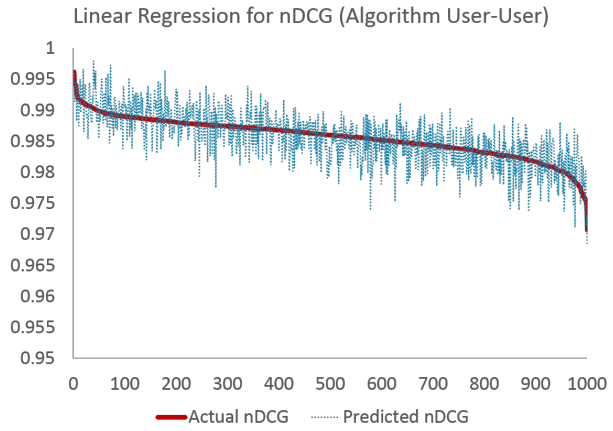

Figure 6. Regression model results for nDCG (User-based Algorithm).

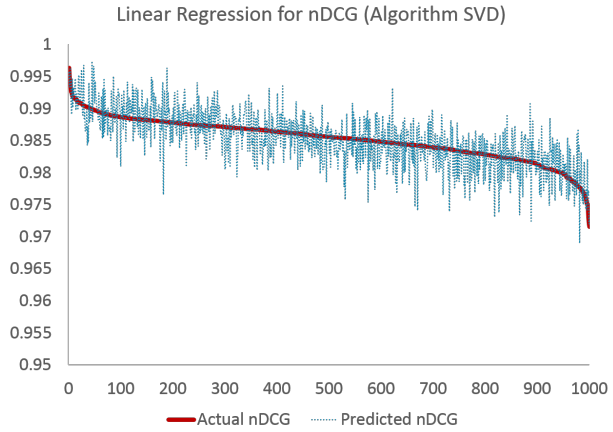

Figure 7. Regression model results for nDCG (Matrix Factorization Algorithm).

B. *Evaluation of subgraph classification model*

The correlation between the predicted and actual CF performance values for the different bipartite subgraphs was high, though not as high as expected, based on previous research results. However, the correlation was obvious in the plots. Moreover, when comparing between different partition schemes of the large bipartite graph, it is more important to choose the best partition scheme (i.e. the one that contains subgraphs that will give good recommendations). It is also important to distinguish between good and bad subgraphs and replace the bad ones with subgraphs that contain more ratings about items. Bad subgraphs are those that contain users with few overlapping ratings for items and one possible way to improve them is to add to them users (and their ratings) that rate many items. By duplicating such users it would be possible to improve the quality of recommendations created from each bipartite subgraph. This is out of the scope of this paper and part of our future work plans.

In order to evaluate the performance of our prediction

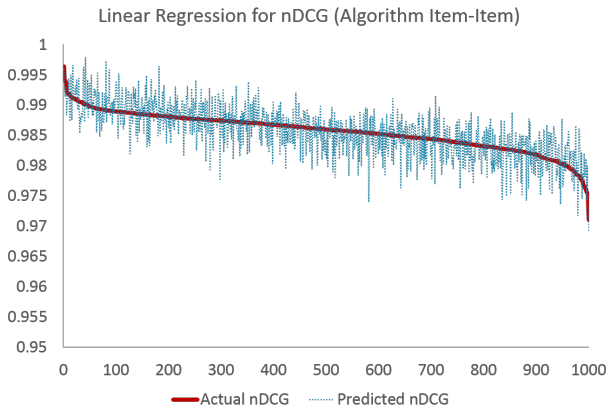

Figure 8. Regression model results for nDCG (Item-based Algorithm).

models in distinguishing between good and bad subgraphs, we repeat the experiments, this time training a binary classifier with the values of the $RMSE\_by\_user$ metric. Using a RandomForest classifier we get an accuracy of 90.8% $\pm$0.02 (at 99% CI) on predicting correctly whether the SVD CF algorithm will output a "high" or "low" overall RMSE value on a specific bipartite subgraph. The performance of SVM classifier with a radial kernel function reaches 93.5% $\pm$0.03 (at 99% CI) and similar are the results for other metrics and CF algorithms.

Based on the above findings, we decide to train a binary classification model, more specifically an SVM classifier on the $RMSE\_by\_user$ metric and we evaluate all future partitions of a large bipartite graph using this binary classifier. For a partition scheme we extract the graph features for all the generated subgraphs, we classify graphs using the SVM classifier and we calculate the ratio of "good" subgraphs in the scheme.

C. Time performance

All experiments were performed on a Dell PowerEdge R730 server with 24 CPU cores and 96Gb of RAM in total, running Apache Spark. The implementation of CF algorithms provided by LensKit was used. Without the Spark implementation, that we implemented, it was not be possible to take advantage of our infrastructure, since Lenskit is single threaded. Also, despite the large amount of memory available it was not possible to run the three CF algorithms on the whole graph not even on its 2 partitions. When the graph was split to 65 partitions (we explain in the following why) the sequential execution of the three CF algorithms, for each partition, took 2 hours and 5 minutes.

Following the proposed approach, We evaluated 64 different partition schemes raging from 1500 to 4 partitions. The partitioning process, which includes the execution of Metis 64 times was completed in about 30 seconds (which is less than 0.5 seconds per partitioning scheme. The time for computing all the metrics for partitioning schemes with more than 60 subgraphs is less than 10 minutes. Figure 9 displays the time needed for every partition scheme (from 4 to 1500).

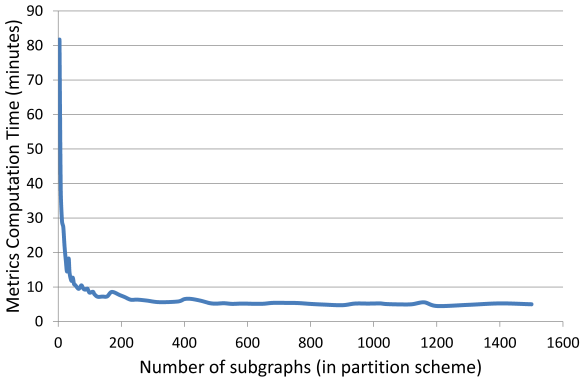

Figure 9. Time for the computation of graph metrics for various partition schemes.

Using the binary classifier explained in the previous section, we predict for each subgraph in a partition the CF performance as "good" or "bad". The time for predicting CF performance using either regression or classification model is less than 1 sec for any partition scheme. Figure 10 provides a plot of the ratio of "good" graphs in each partitioning scheme. From the plot, we see that for partitions with more than 65 subgraphs the ratio is less than 1, which means that a few subgraphs will not provide good recommendations for the respective users. Below 65 graphs, all subgraphs produce good recommendation results. So we can say that 65 is the optimal number of partitions in our case. Of course, partition schemes with more subgraphs (up to 100) still have few "bad" subgraphs, which can be treated by duplicating users from other subgraphs.

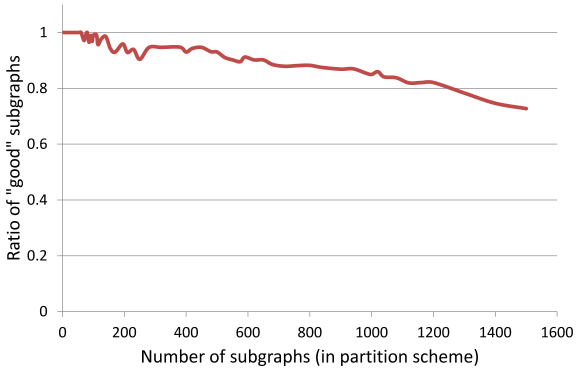

Figure 10. Ratio of "good" subgraphs in various partition schemes.

With the current setup, we are able to parallelize CF algorithms execution taking advantage of all CPUs and memory and reduce the total execution time for the 65

partitions to 8 minutes.

In conclusion, using the proposed approach, it takes 5 to 10 minutes for partitioning the large graph into subgraphs and testing whether these graphs can produce high quality recommendations. Depending on our strategy, we can test a different partitioning scheme or improve selected graphs of the initial partitioning scheme.

VI. Conclusion

This article presented a methodology for predicting the performance of collaborative filtering algorithms using the structural features of the bipartite graph. The methodology incorporates several metrics that either apply to the whole graph or to each node or edge separately and results show that the improvement from the use of the node metrics' value distribution features is significant (correlation is above 0.5). Any further improvement that takes into account more structural features, both graph-based, such as redundancy coefficient, and node-based, such as betweeness centrality, etc must also consider the computational complexity and the time needed for finding the optimal partitioning scheme.Using the proposed methodology, a recommender system will be able to generate different partitioning schemes, predict the performance of CF in each one of them and select the optimal partitioning. Thus it can handle huge graphs by splitting and processing each sub-graph separately and perform faster if more resources are available to be used in parallel.

References

[1] C. Sardianos and I. Varlamis, "A scalable solution for personalized recommendations in large-scale social networks," in *Proceedings of the 18th Panhellenic Conference on Informatics*. ACM, 2014, pp. 1–6.

[2] J. Weng, E.-P. Lim, J. Jiang, and Q. He, "Twitterrank: Finding topic-sensitive influential twitterers," in *Proceedings of the Third ACM International Conference on Web Search and Data Mining*. ACM, 2010, pp. 261–270.

[3] J. M. C. Miller McPherson, Lynn Smith-Lovin, "Birds of a feather: Homophily in social networks," *Annual Review of Sociology*, vol. 27, pp. 415–444, 2001.

[4] J. Tang, X. Hu, and H. Liu, "Social recommendation: a review." *Social Netw. Analys. Mining*, vol. 3, 2013.

[5] I. Guy and D. Carmel, "Social recommender systems," in *Proceedings of the 20th International Conference Companion on World Wide Web*. ACM, 2011, pp. 283–284.

[6] M. C. Pham, Y. Cao, R. Klamma, and M. Jarke, "A clustering approach for collaborative filtering recommendation using social network analysis." *J. UCS*, vol. 17, pp. 583–604, 2011.

[7] P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Improving recommendation quality by merging collaborative filtering and social relationships," in *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*. IEEE, 2011, pp. 587–592.

[8] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

[9] A. Bellogin and J. Parapar, "Using graph partitioning techniques for neighbour selection in user-based collaborative filtering," in *Proceedings of the sixth ACM conference on Recommender Systems*. ACM, 2012, pp. 213–216.

[10] P. Symeonidis, N. Iakovidou, N. Mantas, and Y. Manolopoulos, "From biological to social networks: Link prediction based on multi-way spectral clustering," *Data & Knowledge Engineering*, vol. 87, pp. 226–242, 2013.

[11] E. Karydi and K. Margaritis, "Parallel and distributed collaborative filtering: A survey," *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, p. 37, 2016.

[12] P. Matuszyk and M. Spiliopoulou, "Predicting the performance of collaborative filtering algorithms," in *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*. ACM, 2014, p. 38.

[13] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, 1994, pp. 175–186.

[14] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW '01. ACM, 2001, pp. 285–295.

[15] 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*, vol. 41, no. 6, p. 391, 1990.

[16] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

[17] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T.-Y. Liu, "A theoretical analysis of ndcg ranking measures," in *Proceedings of the 26th Annual Conference on Learning Theory (COLT 2013)*, 2013.

[18] D. LaSalle and G. Karypis, "Multi-threaded graph partitioning," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 225–236.

[19] M. Latapy, C. Magnien, and N. Del Vecchio, "Basic notions for the analysis of large two-mode networks," *Social networks*, vol. 30, no. 1, pp. 31–48, 2008.

[20] F. Tarissan, B. Quoitin, P. Mérindol, B. Donnet, J.-J. Pansiot, and M. Latapy, "Towards a bipartite graph modeling of the internet topology," *Computer Networks*, vol. 57, 2013.

[21] O. Frank, "Triad count statistics," *Annals of Discrete Mathematics*, vol. 38, pp. 141–149, 1988.

[22] E. Frank, "Fully supervised training of gaussian radial basis function networks in weka," 2014.