information

MDPI

Article

Optimizing Parallel Collaborative Filtering Approaches for Improving Recommendation Systems Performance

Christos Sardianos *, Grigorios Ballas Papadatos and Iraklis Varlamis

Department of Informatics & Telematics, Harokopio University of Athens, 176 76 Athens, Greece; grigorisbp@gmail.com (G.B.P.); varlamis@hua.gr (I.V.)

* Correspondence: sardianos@hua.gr; Tel.: +30-210-9549-429

Received: 24 March 2019; Accepted: 25 April 2019; Published: 26 April 2019

Abstract: Recommender systems are one of the fields of information filtering systems that have attracted great research interest during the past several decades and have been utilized in a large variety of applications, from commercial e-shops to social networks and product review sites. Since the applicability of these applications is constantly increasing, the size of the graphs that represent their users and support their functionality increases too. Over the last several years, different approaches have been proposed to deal with the problem of scalability of recommender systems' algorithms, especially of the group of Collaborative Filtering (CF) algorithms. This article studies the problem of CF algorithms' parallelization under the prism of graph sparsity, and proposes solutions that may improve the prediction performance of parallel implementations without strongly affecting their time efficiency. We evaluated the proposed approach on a bipartite product-rating network using an implementation on Apache Spark.

Keywords: recommender systems; collaborative filtering; scalability; graph partitioning; distributed systems; parallel execution; social networks

1. Introduction

Among the plethora of choices for communication at a professional or social level, online social networks stand ahead as an option that allows individuals to connect, communicate and interact within a virtual online environment. Social networks aim to facilitate individuals to connect through social relations (e.g., friendship, follower, colleague links, etc.) and communicate with each other either privately or publicly [1,2].

Product review sites (e.g., Epinions, Yelp, Amazon, etc.) have emerged as a new type of social network that aims to support consumers in making buying decisions for products. In such networks, consumers write reviews and provide ratings for products they have bought, form their opinion by reading product reviews written by other users, or form trust bonds with other users whose opinions they trust.

Graph representations and graph based algorithms have been very popular among researchers that perform analysis of social networks [3]. In the graph representation of a typical social network, users are represented as vertices and the relations that connect them are represented as edges, forming the social graph, which is then used by algorithms to extract useful knowledge (influencers, cliques, communities, etc.). In the simplest form of product reviewing social networks, a new type of vertices is introduced to represent product items and a new type of weighted directed edge is added to represent ratings assigned by users to products. Bipartite graphs are ideal for this representation, since they define two disjoint sets of nodes (i.e., users, items) and directed edges that connect nodes from different sets (a user with an item).

Since the introduction of Recommender Systems in the users' daily activities, the evolution both in the way users interact with the web and with each other has been exceptional. The link between these two domains is currently very tight and this is the result of approaches that applied knowledge from one domain to improve user experience in the other. From one side, recommender systems benefitted early from the content that users share in social networks, which has been used to create a richer user profile [4]. From the other side, social networks took advantage of recommender systems to filter the abundance of information available to their users and create a personalised experience. Several research works attempted to combine both worlds, recommender systems and social networks, in ways that aim to change how people connect with each other, how they interact and what content items they share [5].

Lately, the field of application of recommender systems has expanded in a wide area of domains such as creating movie or music recommendations, suggesting related news, finding useful scientific research papers, recommending possible social connections or potential product users could be interested in buying. However, the type of domains' recommender systems are used for are not limited to the above. There have been developed many domain-specific recommender systems such as for finding experts based on a query string and the domain characteristics [6], or potential researchers for collaborating with [7], even for supporting suggestions on loans, etc. [8], or just simply suggesting pages of interest in Twitter [9]. In general, RSs aim to solve the information overload problem for the users of a social network by recommending items, which can either be content, products, services or even other users.

In the case of product review or product rating sites, the analysis of the bipartite graphs and the information they carry has attracted the interest of researchers in recommender systems, and gave rise to new solutions and algorithms. Recommender systems have become very popular on sites such as IMDB, MovieLens and Netflix, where users rate the films they have seen and receive recommendations for more films of potential interest to them.

It was back in October 2006 when Netflix launched an open contest, which challenged research teams across the world to beat (in terms of the Root Mean Square Error of predicted ratings' metrics) their ratings prediction algorithm. The competition aimed to find the best algorithm that predicts user ratings for films, based on their previous ratings and the ratings of other users. The winning algorithm by the Bell and Koren's team (BellKor Pragmatic Chaos) bested Netflix's own algorithm by 10.06% and was a Collaborative Filtering (CF) algorithm that gave rise to Matrix Factorisation approaches in recommender systems [10,11].

Collaborative Filtering (CF) is one of the most well examined and primarily used techniques in recommender systems to create personalized content that will be provided as recommendations to the targeted users, tailored to each user's preferences. The premise in CF is to use the information from the preferences (typically expressed as ratings) of the user's closest neighbors to generate recommendations. 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, and (c) the proximity of a user with other users in the social graph.

When used in the context of social networks, recommender systems suggest users, content, or both and this is based on the explicit or implicit preferences of the user. Explicit preferences are expressed with likes and follows and implicit with views and other actions that denote interest.

Some of the most well-known social networks, such as Facebook or Twitter, have millions of users, which means that the respective graph size is huge. In addition, the expansion rate of such networks is also impressive, since thousands of new users are added daily, and this pushes the limits of the recommendation algorithms, which must be able to scale up to billions of users and trillions of items and implicit or explicit edges [12]. The maintenance of high performance and low latency of the social network services is unquestionable, no matter how many new users are connecting or how many new content items have been added.

Using bipartite graphs to represent the user-item rating matrix is a straightforward option, especially in social networks, where trust or friendship information is also available in the form of a graph [13,14]. Deep learning and matrix factorization methods are gaining the hype in the recommendation systems research. Although they operate differently than graph-based methods, they are still based on the rating matrix information, but either process the whole matrix information in a single node or randomly partition the matrix in (usually overlapping) sub-matrices for scalability [15]. In this work, we mainly use the bipartite graph representation as a means for partitioning the rating matrix using a less random and more meaningful method. Our partitioning method can take advantage of the social network structure or any other similarity information between users or items. Building on this concept, we examine various methods that build on a better partition overlap and try for a better collaborative filtering performance both in time and quality performance.

The diversity in the size and characteristics of the information created in social networks and applications in many cases push the current state-of-the-art recommender systems algorithms to their limits. The scalability of processing algorithms is further challenged by additional nodes and edges added to the user graph, which may correspond to content items and user interactions with them [16].

The two main alternatives for scaling to large graphs (or large rating matrices in general) are to either upgrade the infrastructure in order to cover the increasing processing requirements, or to partition the graph (or the rating matrix) into smaller sub-graphs, which can be processed in parallel, thus increasing performance without losing quality of the produced results.

Our previous work in the field showed the potential of graph partitioning solutions, which can benefit from by the social graph formed in social networks [17]. People usually consider the opinions of their friends more than the opinions of any random user, so the partitioning of the bipartite (ratings) graph can be based on a pre-partitioning of the social graph. However, this partitioning has a limit, due to the sparsity of the resulting bipartite partitions [18]. As a result, it is important to consider the sparsity of the partitions of the bipartite graph and, if possible, to reduce this sparsity by replicating edges (ratings) across partitions.

In our last work [18], we split the initial problem into sub-problems that can be solved more efficiently using parallel and distributed algorithms, without loss of the effectiveness of the provided solutions (measured by the quality of the generated recommendations). In this work, we study the scalability and the efficiency of this approach in generating user-to-item recommendations using the Collaborative Filtering algorithms of Apache Spark. More specifically, we experiment only with Singular Value Decomposition (SVD++) algorithm [19] and its implementation in Apache Spark, which is a state-of-the-art collaborative filtering algorithm [20]. The proposed approach is based on an initial partitioning of the bipartite graph. For the sake of generality, we do not assume a social graph behind the bipartite graph as we did in previous works. The partitioning of the bipartite graph is done at random and only some of the ratings are replicated across partitions. We evaluate different edge replication strategies across partitions, which aim to increase the performance of the CF algorithm in each partition, and report on their performance.

The contributions of this manuscript focus on introducing an architecture for parallel collaborative filtering addressing the problem of collaborative filtering parallelization in the case of sparse graphs and proposing approaches for improving CF's prediction performance.

The rest of the paper is organized as follows: in Section 2, we summarize related work on distributed and parallel collaborative filtering solutions. In Section 3, we present our proposed method for optimizing Collaborative Filtering in distributed or parallel environments, which is based on the replication of selected edges (ratings) across partitions. In Section 4, we evaluate the various strategies for selecting which edges to copy across partitions, using a dataset from Epinions for our experiments, and discuss the results. Finally, in Section 5, we provide the conclusions and the next steps of our work.

2. Related Work

Collaborative Filtering algorithms have been the state-of-the-art solution for the generation of recommendations in various social network applications. Two of the main problems that CF algorithms have to confront [21] are the ***information sparsity*** and the ***lack of scalability*** in huge datasets. Recent advances in CF algorithms capitalize on the use of deep neural network architectures for adding implicit feedback [22] to the original rating matrix, thus creating a latent space with reduced sparsity. Implicit information is either collected from textual reviews or images that accompany user ratings, thus creating a multi-modal [23] and multi-aspect model [24] for handling item ratings, amd increasing the complexity of the problem and -in some cases- the sparsity of the space. The concept of mapping the original high-dimensional space to a latent low-dimensional semantic space [25] is a common practice for reducing sparsity. A completely different approach to CF on the low-dimensional matrix has been introduced in [26], where a smoothing technique is applied to the user–item bipartite graph using the target user's known preference as a query vector. The technique in [26] is based on the main principle that common ratings (co-citations) denote item similarity and uses this principle to rank items to be recommended instead of predicting a specific rating for each item.

The undeniable progress in the field of recommender systems has led to a great research interest towards parallel and distributed implementations of collaborative filtering algorithms. The majority of today's online shops and applications demand lots of high processing units to support their functionality. With current loads of information in these applications, traditional state-of-the-art implementations of CF algorithms could not cope with these needs. A first direction in addressing the efficiency problem of Collaborative Filtering (CF) comprises methods that hash users and items as latent vectors in the form of binary codes, so that user-item affinity can be efficiently calculated in a Hamming space [27]. In a similar direction, low-dimensional item embeddings that incorporate both graph structure and feature information have been used in [13] to reduce the space complexity and thus increase scalability. The approach involved the use of Graph Convolutional Networks to combine random walks and graph convolutions on the bipartite graph. The second direction focuses on the distribution and parallel execution of the CF algorithm in multiple machines. The tendency of using computer clusters to combine processing power and distribute the processing load has affected the implementation of various CF algorithms to deal with the scalability issues over the last years. Based on this, Java Threads, Pthreads, OpenMP frameworks for parallel programming and Mahout, Apache Spark for data processing have been utilized to implement collaborative filtering [21,28].

One of the first distributed approaches for generating recommendations was presented by [29] that proposed a peer-to-peer SVD model for aggregating user profile information and creating recommendations. Another distributed approach of matrix factorization (SGD) algorithm was implemented in [30], where authors proposed a model of a system for distributed sharing of user-generated content streams. In the same context of recommender systems in decentralized environments like P2P, PipeCF algorithm [31,32] has also been proposed. PipeCF algorithm first divides the original user database into buckets which are stored in different peers and each is assigned an identifier in order to use this as a key when needed. Then, PipeCF uses the information from all users in the same bucket with the active user to compute the predictions. The algorithm increases the weights for the contributions of the most similar users (unanimous amplification) and decreases the weights for users that have rated many items (significance refinement) in a process that is similar to the Term Frequency and Inverse Document Frequency (TF/IDF) weighting of terms in a text collection.

Wang and Pouwelse [33] introduced a predictive model for user-item score prediction, using buddy-tables, which are used to store the top-N most relevant to the user items and are shared across a P2P network users. Recently, leveraging the benefits of distributed processing, authors in [34] proposed a hybrid model called Distributed Partitioned Merge (DPM) model, for processing large social graphs. Using a combination of Fork-Join programming and Pregel framework authors stated that based on their evaluation DPM outperforms both Fork–Join and Pregel's recommendation time.

A television show collaborative recommendation system that uses item-to-item collaborative filtering and delegates most of the work to the numerous network clients rather than centralizing process to the server is presented in [35]. User reviews and comments are common sources for extracting user preferences and creating item recommendations. In this context, researchers in [36] used short review texts to create recommendations based on word vector representations. They used word vector representations for users and items; then, they created a set of training data with the rating scores that users give to items based on these representations and finally used a regression model that was trained to predict the unknown user-item ratings. Furthermore, Ref. [37] introduces two methods for modeling users' review texts, one based on Bag-of-Words Paragraph Vectors [38] and another using recurrent neural networks (RNNs) respectively, in order to produce representations of products used in Collaborative Filtering.

Since Hadoop is one of the most widely used frameworks for distributed processing, Ref. [39] proposed a hybrid approach for recommending movies combining user-based collaborative filtering and content-based filtering and using Hadoop and Hive to create SQL queries. Another interesting hybrid approach for creating recommendations was developed in [40]. Authors used Apache Spark for parallel implementation of a hybrid collaborative filtering algorithm, whereas, in order to deal with the collaborative filtering limitations, they used dimensionality reduction and clustering techniques.

Scalability is one of the biggest issues for modern collaborative filtering approaches [18]. Dealing with this problem, Lee and Hong [41] proposed an Adaptive Collaborative Filtering algorithm Based on Scalable Clustering (ACFSC), which first creates a cluster model for users/items that is based on their feature vectors and then uses the users' neighborhood only to get recommendations based on CF. As already mentioned before, partitioning and clustering are very promising methods for dealing with scalability. Consequently, researchers in [42] combine user and item similarities, multi-dimensional clustering to cluster metadata information and cluster pruning to measure the predicted weighted average score of the user ratings. In addition, N-cut graph clustering can be used to group similar user in the same cluster to increase Collaborative Filtering performance [43].

The main limitation on using clustering incorporated with CF in large sparse graphs is the higher demands on processing power for the clustering process and the increasing probability of prediction errors due to higher sparsity. The proven usability of frameworks like Apache Spark for distributed processing and the scalability limitations of collaborative filtering is the motivation that drives our proposed approach. Based on our evaluation results, the idea of using graph partitioning with partition refinement seems to be a feasible and promising approach.

3. System Architecture

The concept behind collaborative filtering algorithms is that users with very similar profiles are more likely to be interested in the same items or inversely when a user is interested in an item then he/she is more likely to be interested in more items with similar features. Based on this concept, the performance of collaborative filtering algorithms depends on the information we have for users and items. When collaborative filtering algorithms rely only to the bipartite graph of user ratings for items, then the user-to-user similarity is defined on the number of items they rated in common or on the agreement of their ratings for these item (e.g., using cosine similarity). When a new user enters the social network, there is no information about his/her ratings so the similarity to all other users cannot be defined, leading to the cold-start problem. Even for existing users, when the total number of ratings is very low compared to the number of users and items in the social network (i.e., sparse network), it is very likely that users may have rated completely different items, so the user similarity, which is measured on commonly rated items, cannot be defined or is not properly measured. Using information from other sources, such as the social connections between users [44] or user and item context is a promising solution, but is not always applicable.

The distributed processing of a sparse ratings' graph will result in even more sparse partitions and thus affect the performance of collaborative filtering algorithms. In our previous work [18], we applied graph partitioning to the large ratings' graph in order to tackle the problem of scalability of collaborative filtering algorithms, but restricted the number of partitions in order to avoid high sparsity. In this work, we extend the previous approach, which also took advantage of distributed architectures (like Apache Spark) and algorithms (like SVD or SVD++), by adding some new refinements in the initial methodology of graph partitioning on the bipartite graph. Our extension begins with the partitions of the bipartite graph, which could be created with any graph partitioning technique, and replicate selected ratings in all partitions in order to increase the density in each partition and boost the performance of the SVD++ algorithm, even for partitions with only one rating.

We develop two strategies for choosing the ratings that will be replicated across partitions. The first strategy aims at identifying the most **active users** inside each partition, where active users are the users that contribute the majority of ratings in the partition. Then, it replicates the ratings of the active users to all partitions in order to increase the total number of ratings per partition. This strategy has been implemented in two different scenarios. In the first scenario (**Active Users All Ratings**—AUAR), all the ratings provided by an active user in partition p_i, where $i = 1...n$, are replicated to all the other $n - 1$ partitions. This scenario improves the collaborative filtering algorithm since it introduces more users to the partitions and thus allows finding more similar users to a selected user. Its main drawback is that it can probably introduce new items to the partition and thus may increase the sparsity ($sparsity = \frac{items \times users}{ratings}$) of the partition and consequently the prediction error.

The second scenario (**Active Users Selected Ratings—AUSR**) also replicates the ratings of active users across partitions, but tackles the main drawback of the AUAR scenario. This is done by finding the active users of a partition (e.g., p_i), but replicating only the ratings for items that already exist in each target partition. For example, if an active user u_x in p_i has rated items $i_{u_x,1}, i_{u_x,2}, ..., i_{u_x,k}$ from which only $i_{u_x,2}$ exists in partition p_j, then only the rating for $i_{u_x,2}$ is replicated to p_j. This tactic is very promising for bipartite graphs with a high number of items compared to users since it does not affect the item set of the target partitions, but only adds rating information, thus reducing sparsity and the average error for the predicted scores.

A similar issue to cold-start users in Collaborative Filtering are the items that have none or very few ratings. For these items, the algorithm can hardly find similarities since it has very little information. Our second strategy aims to support these **least rated items** (LRIs) by adding ratings from other partitions. In its first step, it identifies the least rated items across all partitions (e.g., the items that have received only one rating in their partition). In the second step, it uses this list of (LRIs) and repeats the steps of the first strategy for these items only. Thus, it first redefines the concept of Active Users in a partition p_i to be the most Informative Users that have rated the majority of LRIs. Then, it replicates the ratings of these Informative Users across partitions. Only the selected ratings scenario is evaluated **Informative Users Selected Ratings—IUSR**, since the all ratings scenario suffers from increased sparsity.

As described so far, our approach (depicted in Figure 1) is based on a three-stages architecture: (i) Graph partitioning, (ii) Partition refinement, and (iii) Distributed execution of the CF algorithm in Spark nodes, which also includes the evaluation of results. The starting point of our model is a bipartite graph with a set of Users U, a set of Items I and the respective edges of ratings R among these sets. The first stage in our approach involves the random split of the initial graph into partitions, whereas the only premise is the equality in terms of size (number of ratings) among the different sub-graphs. The partition refinement stage that follows the two strategies (three scenarios) described above is followed, in order to improve the overall density of the partitions and consequently reduce the prediction error. The final stage of our approach consists of the execution of a distributed Collaborative Filtering algorithm over the refined partitions produced in stage two and the evaluation of the results using a 10-fold cross validation approach. In order to evaluate the contribution of the various strategies, we measure the number of ratings that are replicated across partitions, the resulting sparsity of partitions, the time complexity and the prediction error of each method.

Algorithm 1 describes the graph partitioning with edge replication (GPwER) approach.

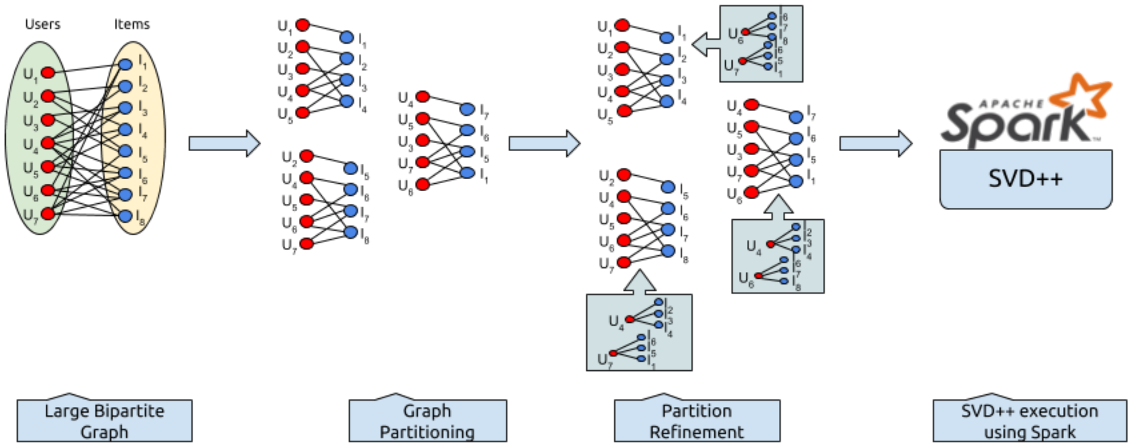

Figure 1. The steps of the proposed approach for generating recommendations in large sparse bipartite graphs: (**i**) Graph partitioning; (**ii**) partition refinement; (**iii**) execution of SVD++ in Apache Spark

Algorithm 1 Graph partitioning with edge replication across partitions

```
procedure PartitionAndEnhanceBG(In:$BG\{V_U, V_I, R\}, Out : \{EBGP_j\{V_{U_j}, V_{I_j}, R_j\}\}$)
    $\{BGP_j\{V_{U_j}, V_{I_j}, R_j\}\} = $ PartitionRatings$(BG\{V_U, V_I, R\})$
    for $g_j$ in $BGP_j$ do
        $EBGP_j\{V_{U_j}, V_{I_j}, R_j\} = $ ReplicateEdges$_{scenario}(g_j, BGP_j)$

procedure $ReplicateEdges_{AUAR}(In : g, \{BGP_j\}, Out : g')$          ▷ Start of Scenario: AUAR
    $V_{U_i} = $ FindMostActiveUsers$(g\{V_U, V_I, R\})$
    for U inV$_{U_i}$ do
        $E = $ GetRatingsForVertex$(U, \{BGP_j\})$
        for e in E do
            $IntPart = $ InterPartitionEdge$(e, \{BGP_j\})$
            $\{g'\{V_U, V_I, R\}\} = $ ReplicateEdge$(g\{V_U, V_I, R\}, IntPart)$          ▷ End of Scenario: AUAR
procedure $ReplicateEdges_{AUSR}(In : g, \{BGP_j\}Out : g')$          ▷ Start of Scenario: AUSR
    $V_{U_i} = $ FindMostActiveUsers$(g\{V_U, V_I, R\})$
    for U inV$_{U_i}$ do
        $E = $ GetRatingsForVertex$(U, \{BGP_j\})$
        for e in E do
            if $(e_{V_U}\ in\ B_sG_{V_I})$ then
                $IntPart = $ InterPartitionEdge$(e, \{BGP_j\})$
                $\{g'\{V_U, V_I, R\}\} = $ ReplicateEdge$(g\{V_U, V_I, R\}, IntPart)$          ▷ End of Scenario: AUSR
procedure $ReplicateEdges_{IUSR}(In : g, \{BGP_j\}Out : g')$          ▷ Start of Scenario: IUSR
    $V_{I_i} = $ FindLeastRatedItems$(g\{V_U, V_I, R\})$
    for i inV$_{I_i}$ do
        $E = $ GetRatingsForVertex$(I, \{BGP_j\})$
        for e in E do
            $IntPart = $ InterPartitionEdge$(e, \{BGP_j\})$
            $\{g'\{V_U, V_I, R\}\} = $ ReplicateEdge$(g\{V_U, V_I, R\}, IntPart)$          ▷ End of Scenario: IUSR
```

The procedure begins with the partitioning of the bipartite graph ($BG\{V_U, V_I, R\}$), which comprises user (V_U) and item (V_I) vertices and rating edges (R), into a set of n not overlapping partitions ($BGP_j\{V_{U_j}, V_{I_j}, R_j\}$) (i.e., $\cup_{j\in[1\dots n]} R_j = R$ and $\cap_{j\in[1\dots n]} R_j = \emptyset$). It then continues with any of the three scenarios proposed in this work: **AUAR, AUSR** (**FindMostActiveUsers**$(g\{V_U, V_I, R\})$) and **IUSR** (**FindLeastRatedItems**$(g\{V_U, V_I, R\})$) that add edges to each bipartite graph partition and result in a set of enhanced partitions ($EBGP_j$) that are given as input to the CF algorithm.

4. Experimental Evaluation

In our previous work [18], we proposed a method for partitioning the bipartite graph (or matrix) of user-item ratings using information from the social graph formed among the users of a social network. This partitioning method allowed us to partition the collaborative filtering problem to smaller problems and scale up to large bipartite (rating) graphs. However, previous results showed that the sparsity of the resulting bipartite graphs in some partitions led to poor CF performance. In addition, an increase to the number of partitions further increased the sparsity problem. In this work, we introduce a methodology for controlling the sparsity of graph partitions by adding more edges (ratings). The aim of the experimental evaluation process is to check which of the proposed alternatives for selecting edges to replicate across partitions performs and better, and how the proposed methods compare against a baseline random partitioning without overlaps and against a state-of-the-art CF algorithm (SVD++) applied on the complete, unpartitioned, bipartite graph (rating matrix). Since sparsity is the main issue that affects CF algorithm performance, we report on the number of replicated across partitions (Section 4.2.1), on the average graph sparsity after replication (Section 4.2.2), on the total execution time (Section 4.2.3) and the achieved prediction performance (Section 4.2.4 of each method.

4.1. Dataset

For the experimental evaluation of the proposed approach, a part of the Epinions dataset, has been employed. Epinions is one of the largest product reviews sites, and a dataset comprising 13,668,320 ratings is provided from the University of Koblenz-Landau (http://konect.uni-koblenz.de/networks/). The details of the dataset (Epinions product ratings dataset—October 2013) are given in Table 1.

Table 1. Characteristics of the Epinions dataset bipartite graph.

Graph Characteristics	
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
Sparsity	1.50×10^4

In our experiments, we use a subset of the Epinions dataset comprising 50,000 ratings, as shown in Table 2. The purpose of using only a subset of the original dataset was to examine the feasibility of our proposed methodology and study its performance using limited processing resources (i.e., a laptop with an Intel Core i7 quadcore CPU @1.8 GHz with 8 GB RAM and a solid state drive, running MS Windows 10. The same partitioning methodology can then be generalized and applied on larger graphs with different characteristics.

Table 2. Characteristics of the Epinions subset used in the study

Graph Characteristics	
Num. of Distinct Users (raters)	9435
Num. of Distinct Items	16,288
Num. of Ratings	50,000
Avg. outDegree/User	3.89
Avg. inDegree/Item	2.48
Sparsity	3.25×10^4

4.2. Results

Regarding the methodology followed in the experiments, as explained in the previous section, the main bipartite graph is randomly partitioned into k partitions that contain an equal number of ratings. Ratings are split to partitions at random, which may result in the case that a user appears in more than one partitions providing ratings for different items in each partition. Similarly, an item may appear in more than one partition and is rated by different users in each partition. After creating the k-partitions, each partition is processed and the most active m-users are selected, based on the strategies described in Section 3. All the ratings (or selected ratings depending on the strategy followed) of these $k \times m$ users are replicated across partitions. The resulting partitions are shuffled and the SVD++ algorithm is applied in a 10-fold cross-validation experiment. The average Root Mean Square Error (RMSE) of the 10-folds is reported. RMSE is a commonly used metric for the evaluation of rating prediction algorithms and compares the predicted rating $\hat{y}_i$ against the actual rating y_i for all the ratings of each fold as shown in Equation (1). RMSE is one of the most commonly used metrics for evaluating models and estimating their average prediction performance. It is preferred when the error distribution is expected to be Gaussian and when large errors are particularly undesirable. Furthermore, it penalizes large errors giving a higher weight to them:

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

In our experiments, we choose $k = 2$ and a varying number of users m (from 50 to 5000) and we evaluate the performance of the three proposed methods: (i) AUAR that replicates all the ratings of selected users to all partitions, which results in (indirectly) adding new items to the partitions, (ii) AUSR that replicates only selected ratings from the selected users, so that more ratings are added to the items of a partition but no extra items are introduced, and (iii) IUSR that replicates selected ratings from selected users, who are selected in order to introduce ratings to the less rated items. In addition, we report on the performance of the baseline partitioning method (method “Random” in the plots) that randomly splits the bipartite graph (i.e., the set of rating edges) in a predefined number of partitions. This baseline method is the starting point of our edge replication methods. In the text or in the plots, we also refer to the performance achieved without partitioning the bipartite graph. In the two latter cases, the performance is a single value, but in some plots it is depicted with a horizontal dashed line in order to allow comparison with an increasing number of added users.

4.2.1. Number of Replicated Ratings

The three methods proposed in this work selectively replicate a number of ratings to each partition, following a different strategy for selecting the users that will contribute to this replication and this results in a different number of ratings added in each partition. This clearly affects the complexity of the algorithm and the amount of information added to each sub-problem. In this point, we assume that predicting ratings in each partition is a sub-problem of the original rating prediction problem in the original bipartite graph. In Figure 2, we depict the number of ratings added in average per partition. Given that $k = 2$, the baseline random partitioning results in an average of 25,000 ratings per partition, so adding 11,500 ratings in average to each partition, as is the case for AUAR using the 500 most active users, means that we increase the size of each bipartite partition by 46%. From the figure, it is obvious that the IUSR approach adds much fewer ratings to each partition than the other two methods (four times less), which affects the complexity of the SVD++ algorithm as shown in the following.

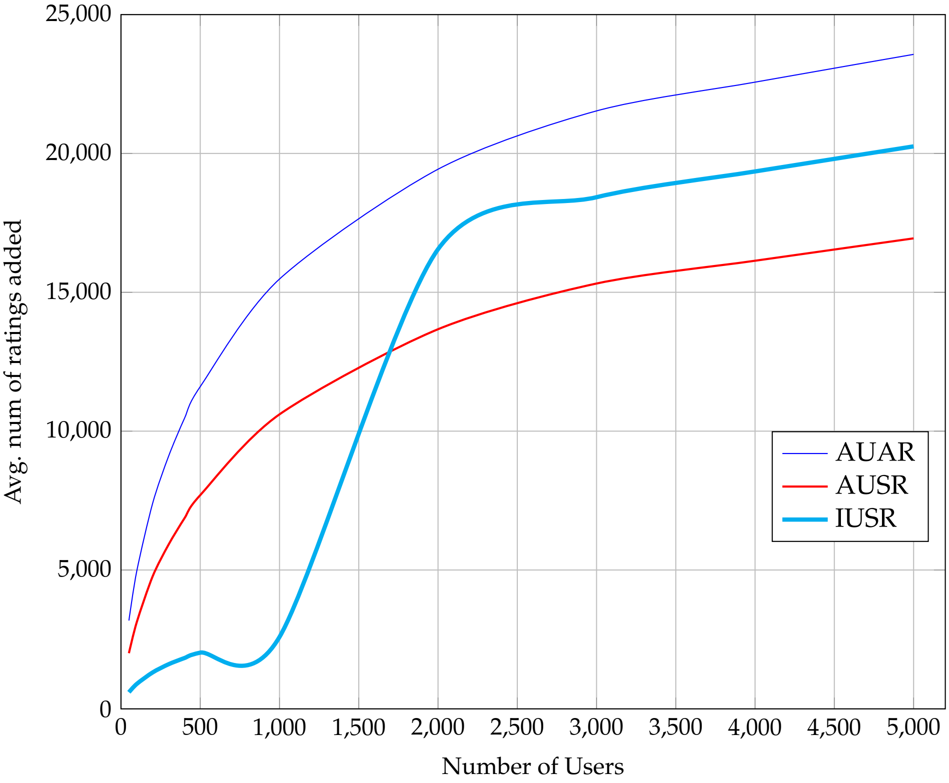

Figure 2. Average number of ratings added for different number of replicated users in each scenario.

4.2.2. Graph Density

The replication of ratings (edges) across partitions aims at increasing the amount of information available for items and users. It is also expected to reduce the sparsity (or increase the density) of the resulting bipartite graph, but this is not always the case, as depicted in Figure 3. Graph density is defined as the ratio of edges of the actual graph divided by the number of edges the graph would have if it was fully connected. In the case of a bipartite graph with n_u users, n_i items and n_e edges (ratings) the density is given by Equation (2):

$$GraphDensity = \frac{n_e}{n_u \times n_i}. \tag{2}$$

The density of the original bipartite graph as well as the average density of its randomly created partitions is a very small number between 0 and 1. This means that the graphs are very sparse and the CF algorithm has to deal with this sparsity. According to Figure 3, the density of the original bipartite graph is 3.25×10^{-4} and after the random partitioning the average density of the partitions raises to 3.85×10^{-4}. The average density of the graph increases for the three proposed methods, when less than 1000 users are added. From that point forward, the density of the graph decreases because the denominator of Equation (2) increases more than the nominator. Comparing between the two methods that add ratings from the active users, AUAR that introduces new items to each partition (i.e., increases both n_i and n_u) results in a quicker drop than AUSR, which only adds new users in each partition (i.e., increases the n_u only). Finally, the IUSR method has a smaller increase to the graph density since the added users are those that rated items with few ratings, so they do not necessarily add many new ratings to the partition (i.e., do not increase n_e as much as AUSR), but also keeps n_i constant so is comparable to AUAR for few users and compares to AUSR when more users are selectively added so that they contribute to the items with few ratings.

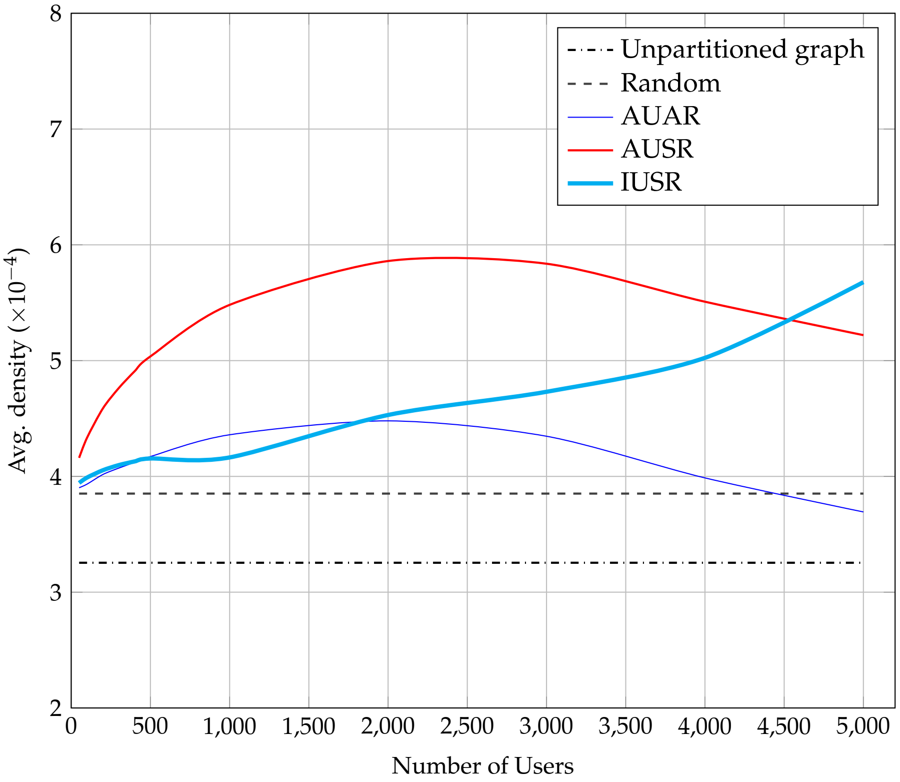

Figure 3. Average density per partition for different numbers of replicated users for the three scenarios.

4.2.3. Execution Time

The execution time of the SVD++ algorithm strongly depends on the number of users and items in the partition, so it is important for methods that add information to the partition to do it wisely. This is evident in Figure 4, where the baseline random partitioning has an execution time of 19.7 minutes (wall-time) and all the three methods have always higher execution times. More specifically, AUAR has the highest time complexity, since it introduces many items per partition and almost doubles the execution time when the 5000 most active user ratings are replicated across partitions. AUSR follows a more conservative approach, which results in better time performance, even for high numbers of users. Finally, IUSR demonstrates the best time performance among the three methods, since it adds much fewer ratings than all other methods and keeps the complexity of SVD++ low and close to the baseline.

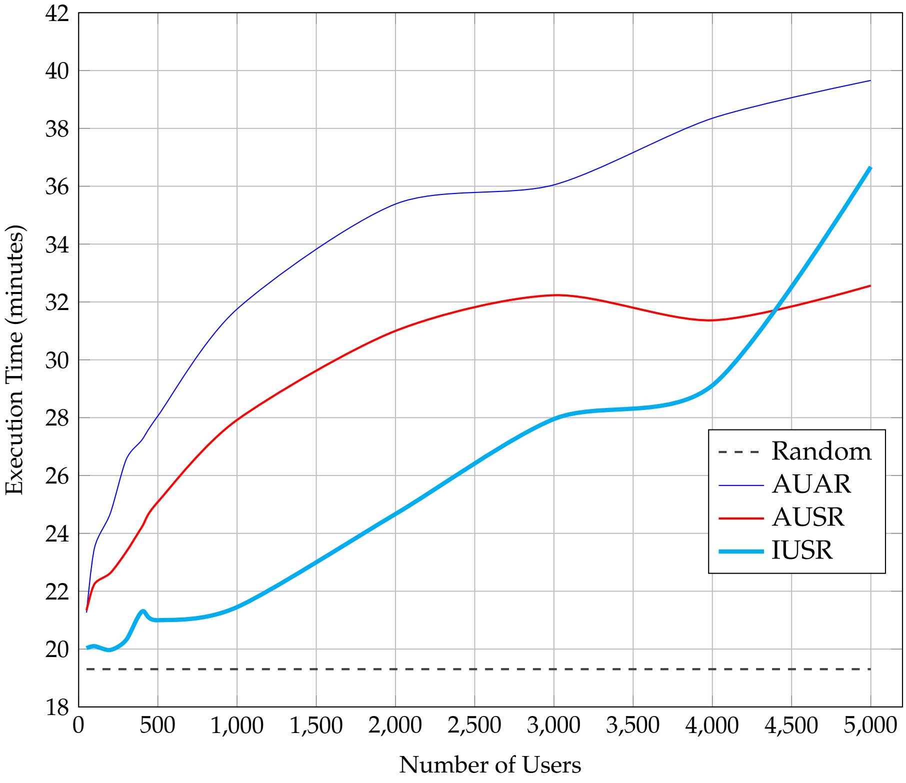

Figure 4. Execution time for different numbers of user replications for the three scenarios.

4.2.4. Prediction Performance

When the bipartite graph is very sparse, it is quite possible that the collaborative filtering algorithm will not be able to predict the rating of a user for a specific item. When an item appears only in the test subset and there is no evidence about it in the training subset, the collaborative filtering algorithm fails and recommender systems usually rely on baseline (fallback) solutions for predicting the rating of the user for the item (e.g., they predict using the average of all ratings, or the average of all ratings of the same user or the same item, depending on the CF approach used in the first step). In the k-fold cross validation scenario, this happens more frequently, so the percentage of cases that fall back in a baseline solution is even higher. The Bipartite Clustering Coefficient (BCC) [45] is a measure that represents this local notion of density in the bipartite graph and is high when we have more ratings from the same user (for various items) and for the same item (from various users). In our previous work [18], we have shown that BCC is critical for the performance of the CF algorithm. The results in Figure 5 show that in the original graph the ratio of fallback cases is high -around 33%- and drops to 23% in average after partitioning. The three methods manage to further decrease this ratio below 20%, with AUSR having the best performance when the number of added users increases. Once again, this is explained by the number of information that is being added to each partition, but this improvement comes at cost in the total execution time as explained in the previous paragraph.

The goal of each collaborative filtering algorithm is to predict the user rating for an item and as a result the outmost criterion for evaluating the performance of any method is the correctness of predictions. As explained in the methodology, the methods' performance is compared using RMSE, which must be as low as possible. The results in Figure 6 show that, compared to the baseline of random partitioning, the proposed methods manage to improve CF performance. Among the three methods, IUSR achieves the lowest RMSE values (using the selected ratings of 300 and 500 selected users). However, its performance is comparable to AUSR when more users have to be added. Adding more users always comes at a time performance cost, which has to be considered in a full scale scenario. Finally, the AUAR method has the worst performance of the three, which in some cases (when few active users and all their ratings are replicated) is worse than that of the random partition baseline. This means that indiscriminately replicating ratings across partitions can negatively affect the performance of the recommendation algorithm.

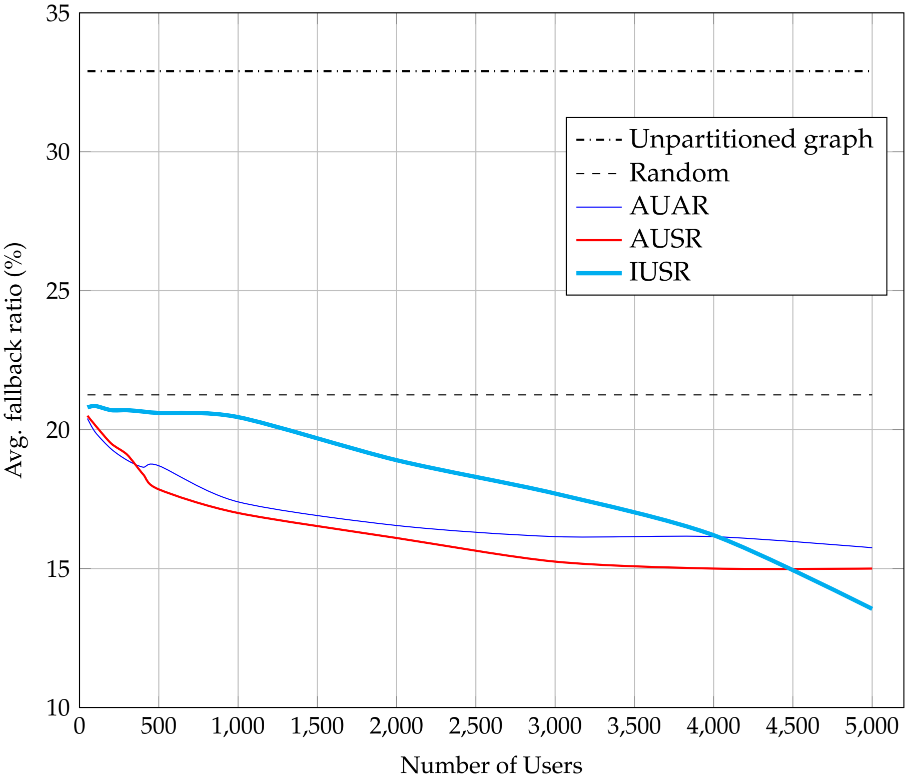

Figure 5. Average percentage of fallback predictions for different number of replicated users in each scenario.

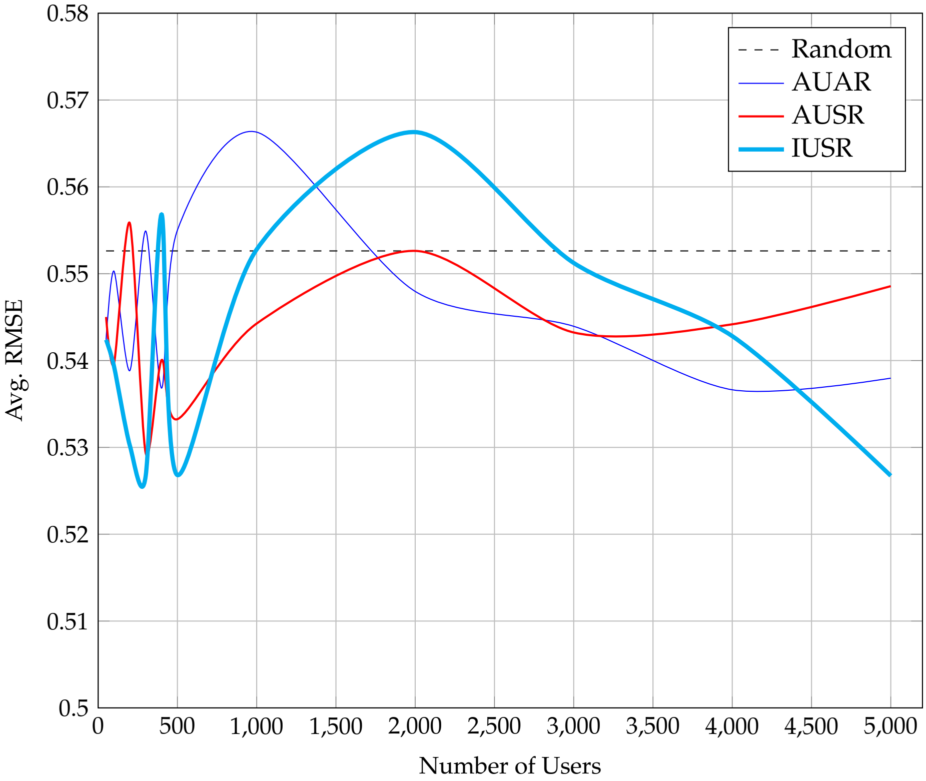

Figure 6. The average Root Mean Square Error (RMSE) for the three scenarios compared to the baseline approach, for different number of replicated users.

4.3. Discussion

The results of the experimental evaluation demonstrate the benefits of the edge replication approaches over the baseline random partitioning of the bipartite graph. The proposed methodology that selectively replicates the ratings provided by some users for some items can be beneficial to the collaborative filtering algorithms, when they are applied to partitions of the original graph.

The main advantage of this ratings' replication methodology is that it increases the information available for the CF algorithm and balances the information loss that occurs by the graph partitioning. Its main drawback is that it increases the execution time of CF algorithms since it adds more users and items (especially the AUAR method). Choosing which users to consider (IUSR) and which ratings to replicate (AUSR and IUSR) results in an smaller prediction error and thus in better recommendations, without adding much processing overhead. Adding 5% more users (500 users) and their ratings for the items in each partition in our experiments (approximately 2000 ratings for IUSR corresponds to an increase of 8% to the number of ratings) achieved the best prediction performance. This addition increased the total execution time only by 14% (for IUSR).

Considering the time performance of the algorithm when a large number of users are selected for replication, it is obvious that there is no need to replicate more than 30% of the users already in a partition, since this slows down the execution, losing all the benefits of the parallel processing. The prediction accuracy of the SVD++ algorithm seems to improve only in the case of IUSR for a high number of users. However, the same performance can be achieved with lower complexity, when only 5% more users (and their selected ratings) are added in each partition.

5. Conclusions

This work introduced our approach in optimizing the performance of Collaborative Filtering algorithms that run in a parallel setup, with distribution of the bipartite information. In this paper, we evaluated several strategies that attempt to replicate ratings across more than one partition in order to increase the amount of information available for the CF algorithm and to confront the sparsity problem in the resulting graph partitions. Results showed that it is possible to improve the performance of the CF algorithms that run in parallel on different partitions of the graph if selected users and their ratings are replicated across partitions, so that items that have received few ratings in the original split are enhanced with more ratings from other partitions.

In this paper, we tested the idea of replicating edge information (ratings) across more than one partition and results show that this improves CF performance. Following this idea, it is our next step to evaluate various nodes and edge selection strategies, which can further improve the performance of the CF algorithm, whilst providing a good balance in the execution performance. It is also in our next steps to compare against deep learning state-of-the-art methods, which can be applied on partitions of the bipartite graph (or rating matrix) such as graph CNN techniques that compress rating matrix information using random walks or probabilistic matrix factorization techniques that apply on matrix partitions.

Author Contributions: C.S.: methodology, validation, and writing; G.B.P.: software, visualization, and validation; I.V.: conceptualization, supervision, and writing.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CF	Collaborative Filtering
RMSE	Root Mean Square Error
AUAR	Active Users All Ratings
AUSR	Active Users Selected Ratings
LRI	Least Rated Items
IUSR	Informative Users Selected Ratings

References

1. Symeonidis, P.; Ntempos, D.; Manolopoulos, Y. *Recommender Systems for Location-Based Social Networks*; Springer: Berlin/Heidelberg, Germany, 2014.
2. Nepali, R.; Wang, Y. SocBridge: Bridging the Gap between Online Social Networks. 2014. Available online: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1608&context=amcis2014 (accessed on 24 March 2019).
3. Pitas, I. *Graph-Based Social Media Analysis*; CRC Press: Boca Raton, FL, USA, 2016.
4. Bonhard, P.; Sasse, M.A. 'Knowing me, knowing you'—Using profiles and social networking to improve recommender systems. *BT Technol. J.* **2006**, *24*, 84–98. [CrossRef]
5. Zhou, X.; Xu, Y.; Li, Y.; Josang, A.; Cox, C. The state-of-the-art in personalized recommender systems for social networking. *Artif. Intell. Rev.* **2012**, *37*, 119–132. [CrossRef]
6. Chen, H.H.; Ororbia, I.; Alexander, G.; Giles, C.L. ExpertSeer: A keyphrase based expert recommender for digital libraries. *arXiv* **2015**, arXiv:1511.02058.
7. Chen, H.H.; Gou, L.; Zhang, X.; Giles, C.L. Collabseer: A search engine for collaboration discovery. In Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries, Ottawa, ON, Canada, 13–17 June 2011; pp. 231–240.
8. Felfernig, A.; Isak, K.; Szabo, K.; Zachar, P. The VITA financial services sales support environment. In Proceedings of the National Conference On Artificial Intelligence, Vancouver, BC, Canada, 22–27 July 2007; p. 1692.
9. Gupta, P.; Goel, A.; Lin, J.; Sharma, A.; Wang, D.; Zadeh, R. Wtf: The who to follow service at twitter. In Proceedings of the 22nd International Conference on World Wide Web, Rio de Janeiro, Brazil, 13–17 May 2013; pp. 505–514.
10. Schafer, J.B.; Frankowski, D.; Herlocker, J.; Sen, S. Collaborative filtering recommender systems. In *The Adaptive Web*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 291–324.
11. Koren, Y.; Bell, R.; Volinsky, C. Matrix factorization techniques for recommender systems. *Computer* **2009**, *8*, 30–37. [CrossRef]
12. Sardianos, C.; Tsirakis, N.; Varlamis, I. A Survey on the Scalability of Recommender Systems for Social Networks. In *Social Networks Science: Design, Implementation, Security, and Challenges*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 89–110.
13. Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W.L.; Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 974–983.
14. Jamali, M.; Ester, M. A matrix factorization technique with trust propagation for recommendation in social networks. in Proceedings of the Fourth ACM Conference on Recommender Systems, Barcelona, Spain, 26–30 September 2010; pp. 135–142.
15. Ahn, S.; Korattikara, A.; Liu, N.; Rajan, S.; Welling, M. Large-scale distributed Bayesian matrix factorization using stochastic gradient MCMC. In Proceedings of the 21th ACM SIGKDD International Conference On Knowledge Discovery and Data Mining, Sydney, Australia, 10–13 August 2015; pp. 9–18.
16. Varlamis, I.; Eirinaki, M.; Louta, M. Application of social network metrics to a trust-aware collaborative model for generating personalized user recommendations. In *The Influence of Technology on Social Network Analysis and Mining*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 49–74.

17. Sardianos, C.; Varlamis, I. A scalable solution for personalized recommendations in large-scale social networks. In Proceedings of the 18th Panhellenic Conference on Informatics, Athens, Greece, 2–4 October 2014; pp. 1–6.
18. Sardianos, C.; Varlamis, I.; Eirinaki, M. Scaling Collaborative Filtering to Large–Scale Bipartite Rating Graphs Using Lenskit and Spark. In Proceedings of the 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService), Redwood City, CA, USA, 6–9 April 2017; pp. 70–79.
19. Koren, Y. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In Proceedings of the 14th ACM SIGKDD International Conference On Knowledge Discovery and Data Mining, Las Vegas, NV, USA, 24–27 August 2008; pp. 426–434.
20. Bosagh Zadeh, R.; Meng, X.; Ulanov, A.; Yavuz, B.; Pu, L.; Venkataraman, S.; Sparks, E.; Staple, A.; Zaharia, M. Matrix computations and optimization in apache spark. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 31–38.
21. Eirinaki, M.; Gao, J.; Varlamis, I.; Tserpes, K. Recommender systems for large-scale social networks: A review of challenges and solutions. *Future Gener. Comput. Syst.* **2018**, *78*, 413–418. [CrossRef]
22. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T.S. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 173–182.
23. Cheng, Z.; Chang, X.; Zhu, L.; Kanjirathinkal, R.C.; Kankanhalli, M. MMALFM: Explainable recommendation by leveraging reviews and images. *ACM Trans. Inf. Syst.* **2019**, *37*, 16. [CrossRef]
24. Cheng, Z.; Ding, Y.; He, X.; Zhu, L.; Song, X.; Kankanhalli, M.S. Aˆ 3NCF: An Adaptive Aspect Attention Model for Rating Prediction. In Proceedings of the 27th International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 13–19 July 2018; pp. 3748–3754.
25. Cheng, Z.; Shen, J. On effective location-aware music recommendation. *ACM Trans. Inf. Syst.* **2016**, *34*, 13. [CrossRef]
26. He, X.; Gao, M.; Kan, M.Y.; Wang, D. Birank: Towards ranking on bipartite graphs. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 57–71. [CrossRef]
27. Zhang, H.; Shen, F.; Liu, W.; He, X.; Luan, H.; Chua, T.S. Discrete collaborative filtering. In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, Pisa, Italy, 17–21 July 2016; pp. 325–334.
28. Karydi, E.; Margaritis, K. Parallel and distributed collaborative filtering: A survey. *ACM Comput. Surv.* **2016**, *49*, 37. [CrossRef]
29. Canny, J. Collaborative filtering with privacy. In Proceedings of the 2002 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 12–15 May 2002; pp. 45–57.
30. Isaacman, S.; Ioannidis, S.; Chaintreau, A.; Martonosi, M. Distributed rating prediction in user generated content streams. In Proceedings of the Fifth ACM conference on Recommender Systems, Chicago, IL, USA, 23–27 October 2011; pp. 69–76.
31. Han, P.; Xie, B.; Yang, F.; Shen, R. A scalable P2P recommender system based on distributed collaborative filtering. *Expert Syst. Appl.* **2004**, *27*, 203–210. [CrossRef]
32. Han, P.; Xie, B.; Yang, F.; Wang, J.; Shen, R. A novel distributed collaborative filtering algorithm and its implementation on p2p overlay network. In Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, Sydney, Australia, 26–28 May 2004; pp. 106–115.
33. Wang, J.; Pouwelse, J.; Lagendijk, R.L.; Reinders, M.J. Distributed collaborative filtering for peer-to-peer file sharing systems. In Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France, 23–27 April 2006; pp. 1026–1030.
34. Corbellini, A.; Godoy, D.; Mateos, C.; Schiaffino, S.; Zunino, A. DPM: A novel distributed large-scale social graph processing framework for link prediction algorithms. *Future Gener. Comput. Syst.* **2018**, *78*, 474–480. [CrossRef]
35. Ali, K.; Van Stam, W. TiVo: making show recommendations using a distributed collaborative filtering architecture. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 22–25 August 2004; pp. 394–401.
36. Zheng, X.; He, W.; Li, L. Distributed representations based collaborative filtering with reviews. *Appl. Intell.* **2019**, 1–18. [CrossRef]

37. Almahairi, A.; Kastner, K.; Cho, K.; Courville, A. Learning distributed representations from reviews for collaborative filtering. In Proceedings of the 9th ACM Conference on Recommender Systems,Vienna, Austria, 14–20; pp. 147–154.
38. Le, Q.V.; Mikolov, T. Distributed Representations of Sentences and Documents. Available online: http://proceedings.mlr.press/v32/le14.pdf (accessed on 24 March 2019).
39. Bharti, R.; Gupta, D. Recommending Top N Movies Using Content-Based Filtering and Collaborative Filtering with Hadoop and Hive Framework. In *Recent Developments in Machine Learning and Data Analytics*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 109–118.
40. Panigrahi, S.; Lenka, R.K.; Stitipragyan, A. A hybrid distributed collaborative filtering recommender engine using apache spark. *Procedia Comput. Sci.* **2016**, *83*, 1000–1006. [CrossRef]
41. Lee, O.J.; Hong, M.S.; Jung, J.J.; Shin, J.; Kim, P. Adaptive collaborative filtering based on scalable clustering for big recommender systems. *Acta Polytech. Hung.* **2016**, *13*, 179–194.
42. Li, X.; Murata, T. Using multidimensional clustering based collaborative filtering approach improving recommendation diversity. In Proceedings of the 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, Macau, China, 4–7 December 2012; pp. 169–174.
43. Bellogin, A.; Parapar, J. Using graph partitioning techniques for neighbour selection in user-based collaborative filtering. In Proceedings of the Sixth ACM conference on Recommender Systems, Dublin, Ireland, 9–13 September 2012; pp. 213–216.
44. Guo, G.; Zhang, J.; Thalmann, D. Merging trust in collaborative filtering to alleviate data sparsity and cold start. *Knowl. Based Syst.* **2014**, *57*, 57–68. [CrossRef]
45. Latapy, M.; Magnien, C.; Del Vecchio, N. Basic notions for the analysis of large two-mode networks. *Soc. Netw.* **2008**, *30*, 31–48. [CrossRef]