Application of Random Walks to Decentralized Recommender Systems

Anne-Marie Kermarrec, Vincent Leroy, Afshin Moin, Christopher Thraves-Caro

Application of Random Walks to Decentralized Recommender Systems*

Anne-Marie Kermarrec
INRIA Rennes - Bretagne Atlantique, Rennes, France

Vincent Leroy
INSA de Rennes, UEB, Rennes, France

Afshin Moin
INRIA Rennes - Bretagne Atlantique, Rennes, France

Christopher Thraves
INRIA Rennes - Bretagne Atlantique, Rennes, France

September 22, 2010

Abstract

The need for efficient decentralized recommender systems has been appreciated for some time, both for the intrinsic advantages of decentralization and the necessity of integrating recommender systems into P2P applications. On the other hand, the accuracy of recommender systems is often hurt by data sparsity. In this paper, we compare different decentralized user-based and item-based Collaborative Filtering (CF) algorithms with each other, and propose a new user-based random walk approach customized for decentralized systems, specifically designed to handle sparse data. We show how the application of random walks to decentralized environments is different from the centralized version. We examine the performance of our random walk approach in different settings by varying the sparsity, the similarity measure and the neighborhood size. In addition, we introduce the *popularizing* disadvantage of the significance weighting term traditionally used to increase the precision of similarity measures, and elaborate how it can affect the performance of the random walk algorithm. The simulations on MovieLens 10,000,000 ratings dataset demonstrate that over a wide range of sparsity, our algorithm outperforms other decentralized CF schemes. Moreover, our results show decentralized user-based approaches perform better than their item-based counterparts in P2P recommender applications.

*This work is supported by the ERC Starting Grant GOSSPLE number 204742.

1 Introduction

Recommender systems are crucial to the success of e-commerce websites like Amazon, eBay or Netflix. Different theoretical [4, 14, 3] or empirical [15, 6, 7, 13] approaches have addressed recommender systems. *Collaborative Filtering (CF)* is the most popular strategy in recommender systems. The reason behind this popularity is that CF requires no information about the content of the items. *Neighborhood Model* [7, 18] is the most widly used model of CF due to some of its advantages like better explainability. It is important for a recommender system to be capable of explaining the reason behind a given recommendation. Consequently, the users may increase the quality of future predictions by giving feedback about received recommendations.

A neighborhood model consists of two phases: neighborhood formation and rating estimation. In the neighborhood formation phase a set of similar items (item-based approach) is formed for each item or alternatively a set of similar users (user-based approach) is formed for each user based on some similarity measure like Cosine similarity or Pearson correlation. Then, the neighborhood is input to a prediction function in the rating estimation phase to predict scores for items unseen by the client. Item-based approach has recently received more attention in the domain of centralized recommenders for its better scalability. More specifically, the number of users is usually larger and grows faster than the number of items. These schemes also benefit from better explainability because users have a better knowledge of items than of users.

Yet, recommender systems are confronted to a growing amount of data to process as the number of online users increases, and typically require expensive computational operations and significant storage to provide accurate results. While this combination of factors may saturate centralized systems, fully decentralized approaches provide an attractive alternative with multiple advantages. Firstly, the computation of the predictions can be distributed among all users, removing the need for a costly central server and enhancing scalability. Secondly, a decentralized recommender improves the *privacy* of the users for there is no central entity storing and owning the private information of the users. Several existing algorithms [5], which are out of the scope of this paper, can eventually be deployed in decentralized environments to communicate users' opinions in encrypted form without disclosing their identity. Finally, a distributed recommender service is a valuable feature for peer-to-peer (P2P) applications like BitTorrent and Gnutella as very popular media for users to share their content.

Beside scalability, *sparsity* is another well-known issue of recommender systems. Typically, each user only rates a small amount of items. Consequently, the number of ratings given by the users is very small in comparison with the total number of $(\text{user}, \text{item})$ pairs in the system. For example, the MovieLens 10,000,000 ratings dataset has a density of 1.31%. Therefore, the efficient use of the data at hand is an essential matter to recommender systems.

Despite the numerous advantages that decentralized recommenders offer, the majority of work on recommendation algorithms has been focused on central-

ized systems so far. These algorithms are then not directly applicable to distributed settings. *In this paper, we investigate decentralized neighborhood-based CF recommenders for P2P applications.* Each user can only leverage her own information and data provided by a small (wrt the size of the system) number of other peers[1]. We rely on epidemic algorithms as a decentralized method to form the neighborhood. CF is particularly suitable for the P2P context where no assumption can be made on the content of the items because of the incoherence of meta-data. The contributions of this paper are as follows:

First, decentralized user-based and item-based CF algorithms are implemented and compared in a P2P context using different similarity measures. We show that decentralized user-based approaches deliver better precision and less complexity than decentralized item-based approaches. In fact, decentralized user-based approach does not suffer from drawbacks usually attributed to their centralized counterpart.

Second, we propose a new decentralized recommender system based on random walks. We explain how the decentralized nature of P2P complicates the application of random walks compared to centralized settings. In our algorithm, each peer is provided with a neighborhood composed of a small (wrt the size of the system) set of similar peers by means of an epidemic protocol. Then, the ratings for unknown items of the neighborhood is estimated by running a random walk on this neighborhood. Once the peers have formed their neighborhood, i.e. the epidemic protocol has converged, each peer is thoroughly independent from other peers in generating her recommendations. This algorithm has the best performance over previous decentralized CF algorithms when the data is sparse.

Third, the behavior of the random walk algorithm is discussed in detail in function of three parameters: sparsity, similarity measure, and neighborhood size. This latter strongly affects the precision and complexity of the algorithm in a P2P context. The optimal parameter values of the algorithm is empirically found for MovieLens 10,000,000 ratings dataset. Fortunately, our algorithm significantly improves the precision over a wide range of sparsity while keeping the execution time affordable for peers. At the end of the paper, we show how significance weighting can be a barrier against the success of random walk algorithms.

The rest of this paper is organized as follows. In Section 2 we provide the preliminaries necessary for understanding our approach. Related work is summarized in Section 3. Decentralization of CF algorithms and our user-based random walk recommender system are described in Sections 4 and 5 respectively. In Section 6, we represent the simulation results and compare the performance of different algorithms. The behavior of random walk is also analyzed in this section. Section 7 concludes the paper.

[1]The terms peer and user are interchangeable in this paper.

2 Preliminaries

Traditionally, recommender systems are modeled by a two-dimensional matrix denoted by R, with rows representing users and columns representing items. Each entry r_{ui} of R contains the rating of user u for item i. We assume an M user and N item system, that is $u \in \{1, 2, ..., M\}$ and $i \in \{1, 2, ..., N\}$. Each row R_{u*} is called the *rating vector of user* u, and each column R_{*i} the *rating vector of item* i. The goal of the recommender system is to predict the missing entries of this matrix. In this section, we provide some necessary background on the CF approach. Moreover, epidemic protocols [10] are briefly discussed as the decentralization method we use to form a neighborhood of similar users.

2.1 Collaborative Filtering

User-based CF is presented in [7]. In this approach, a neighborhood of similar users is assigned to each user using some *similarity measure*. One popular coefficient is *Cosine similarity*. For users u and v it is defined as:

$$cos(u, v) = \frac{\sum_{i \in I_u \cap I_v} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u \cap I_v} r_{ui}^2} \sqrt{\sum_{i \in I_u \cap I_v} r_{vi}^2}}$$

where I_u and I_u are the set of items rated by u and v respectively. A disadvantage of Cosine similarity is that it does not take into account the differences in users' rating behaviors. For example in a 5-star rating system, a user may rate from 3 to 5, but another one rates from 1 to 3 to reflect the same opinion on items.

The *Pearson correlation* lifts this drawback by considering the offset of each rating from the user's mean rating. It is defined as:

$$\rho_{uv} = \frac{\sum_{i \in I_u \cap I_v} (r_{ui} - \overline{r}_u)(r_{vi} - \overline{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{ui} - \overline{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{vi} - \overline{r}_v)^2}}$$

where $\overline{r}_u$ is the mean rating of user u. Pearson correlation considers only the items rated by both users, but does not take into account the number of such items. As a result, one may choose a user in her neighborhood while having very few items in common.

To deal with this shortage, some authors opt for integrating a factor of *trust* to Pearson correlation known as *significance weighting* [7]. This is achieved by multiplying the Pearson correlation by a term reflecting the number of common items. In [7], this term is defined as $min(|I_v \cap I_u| / 50, 1)$. Choosing 50 as the minimum number of ratings not to be attenuated is achieved empirically and must be updated with the growth of the dataset and evolution of user ratings. In this paper we use *log* as the term of significance weighting. Since the steep of logarithmic function decreases constantly, it is more discriminating for smaller numbers of common items. We call this *modified Pearson coefficient* and define it as:

$$corr(u, v) = \rho_{uv} \log(|I_v \cap I_u|). \quad (1)$$

Significance weighting has a *popularizing* disadvantage discussed in Section 6.

Once the neighborhood is formed, the rating estimation phase is accomplished following some prediction rule, usually a weighted sum aggregation function:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N(u,i)} \omega_{uv}(r_{vi} - \bar{r}_v)}{\sum_{v \in N(u,i)} \omega_{uv}} \tag{2}$$

where $\hat{r}_{ui}$ is the estimated rating of user u for item i, and $N(u,i)$ the set of users in the neighborhood of u having rated i. Henceforth, we call ω_{uv} the *similarity weight* between users u and v. In this paper, depending on the setting, it can be either of the similarity measures presented in this section, or is the output of the random walk algorithm.

Item-based CF [18] is quite similar to user-based CF. However, the rating vectors of items are used instead of the rating vectors of users to form a neighborhood of similar items for each item. More details and the relevant equations are provided in the technical report [12].

2.2 Epidemic Protocols

In centralized recommender systems, the entire rating matrix R is known to the central recommender. Consequently, the recommender algorithm can search among all the users to assign a neighborhood to a client. This is not efficiently achievable in a decentralized system. Instead, we use epidemic protocols to create users' neighborhood.

In epidemic protocols (also known as *gossip protocols*), peers have access to a Random Peer Sampling service (RPS) [10] providing them with a continuously changing random subset of the peers of the network. When a peer joins the network, her view is initialized at random through the RPS. Each peer also maintains a *view* of the network. This view contains information about the c peers that maximize a clustering function. In this paper, this clustering function reflects how much the peers exhibit a similar rating behavior. It can be either of the similarity measures presented in previous sections depending on the context. In order to converge to the ideal view, each peer runs a *clustering protocol* [20, 9]. A peer periodically selects a gossip target from her view and exchanges her view information with her. Upon reception of new information, the peer compares the new candidates with her actual view, and a set of random peers suggested by RPS. Then, keeping only the c most similar entries, she updates her view in order to improve its quality. While the clustering algorithm increases the risks of network partition, the RPS ensures connectivity with high probability. Gossip clustering protocols are known for converging quickly to high quality views. By regularly communicating with the peers in the view, gossip protocols also ensure their liveness and eliminate disconnected nodes. Gossip protocols are fully decentralized, can handle high churn rates, and do not require any specific protocol to recover from massive failures.

3 Related Work

In this section, we review the previous work on decentralized recommender systems and suggested solutions to sparsity. The research on decentralized recommender systems has remained modest although the need for them grows rapidly. Notable works on the context are as follows: Tribler [2], a decentralized search engine using BitTorrent protocol, is capable of recognizing the user's taste and give recommendations after a few search queries by the user. Each entry of the binary rating vector is 1 if the user has ever downloaded the corresponding item, and 0 otherwise. Tribler uses epidemic protocols to form the neighborhood, and Cosine function as similarity measure. The significance weighting term is defined as $min(1, |I_v|/40)$, where v is the corresponding neighbor. A non-normalized score is computed for each item through user-based CF approach, being consequently used to generate an ordered recommendation list.

PocketLens [16] is a decentralized recommender algorithm developed by GroupeLens research group. In [16], different architectures from centralized to fully decentralized are suggested for neighborhood formation. PocketLens uses the Cosine similarity to estimate the neighborhood quality. Once a neighborhood of similar users is formed, an item-based algorithm is applied on the ratings existing in the neighborhood. The Cosine similarity is used as the similarity weight between items, and predictions are made using a normalized weighted sum.

All of these works use classic similarity measures to predict the ratings. The distinctive point of our work is to apply a *model* to introduce a decentralized model-based CF algorithm.

Several solutions have been suggested to alleviate the problem of sparsity. Some works exploit content information of items or *demographic* information [11] of user's profiles like age, gender or code area to improve the recommendations when the data is not dense enough. Such information is not easy to collect in P2P applications. Furthermore, providing demographic data endangers the users' privacy. *Default rating* [17] is another method for dealing with sparsity. This solution slightly improves the precision of the recommendations by assuming some default value for missing ratings. The disadvantage of this method is in creation of dense input data matrix, hugely increasing the complexity of computations. Hence, this is not a proper solution for P2P either, because the computational power of P2P processors is in general much less than central servers.

Hence, a lot of effort has been made to develop models to mine further the existing data in order to detect potential hidden links between items or users. In [8] trust-based and item-based approaches are combined by means of a random walk model. The algorithm is centralized and the trust is explicitly expressed by the users. The information about trust does not exist in the majority of datasets including MovieLens. Authors in [21] suggest a random walk model as a solution to sparsity in a centralized item-based CF approach. Their algorithm is to some extent similar to an item-based version of our random walk algorithm, but does not lend itself well to decentralized environments.

4 Decentralization of CF Algorithms

The main difficulty in decentralization of the user-based CF algorithm is the neighborhood formation phase. Contrary to the central recommender algorithms, each user of a P2P network can only access the data related to a limited number of other users. It is therefore critical to devise a protocol able to efficiently navigate through the P2P system and gather the most similar peers. Epidemic protocols described in Section 2.2 are very suitable for this task, and converge to a view of the most similar users in only a few cycles. Once the neighborhood is formed, the rating estimation is done locally at each user. While scalability is an issue in centralized user-based recommender systems, decentralized approaches do not suffer from this drawback as each user computes her own recommendations.

The decentralization of the item-based CF algorithm is more of a challenge because the algorithm needs the rating vector of the items to find the similarity between them. This vector can not be known by P2P users as they do not know the ratings of the majority of other peers. Consequently, similar to the user-based approach, each peer should find a neighborhood of similar users as a first step. A *partial* rating vector is then constructed for each item based on the ratings available in the neighborhood, and the item-based CF algorithm is applied.

The complexity of CF algorithms is mostly due to the similarity computation between users or items. For decentralized user-based algorithms, a similarity *vector* between the central user and all peers in the neighborhood is calculated. The complexity of each similarity calculation depends on the number of common items between two users, which may go up to a considerable fraction of all items in the system. The complexity of the operation is then $O(SN)$ for each user, where S is the neighborhood size, and N the number of items in the whole P2P system. In decentralized item-based algorithms, the similarity between unknown items of the neighborhood and the items rated by the user is calculated to form a similarity *matrix*. Provided the neighborhood is big enough, it often contains some users having rated the majority of items. Then, user u needs to compute $L(N - L)$ similarities where L equals $|I_u|$. The complexity of each similarity calculation is proportional to the size of the item rating vectors , being up to the neighborhood size. Hence, the complexity of the decentralized item-based approach is at most $O(N^2S)$ for each peer, where the worst case happens when $L = N/2$. Therefore, it is seen that the decentralized user-based approach is much less complex than the decentralized item-based approach.

For the above reasons, user-based approaches seem to match better a P2P setting. In Section 6, it is empirically shown that decentralized user-based approaches also have better precision than decentralized item-based schemes.

5 Decentralized Prediction through Random Walk

In CF recommender algorithms, the similarity weight (ω_{uv} or ω_{ij}) is usually the same as similarity measure. In our algorithm, this is computed through random walks. Random walk has been used to design decentralized search engines [19]. In the context of recommender systems, some centralized approaches [21] have used random walks to improve the precision of recommendations. In general, the recommendation problem is modeled by a weighted and directed graph where vertices represent the entity of interest. This entity is items in item-based recommenders or webpages in PageRank algorithm for example. The application of random walks to centralized recommenders is relatively obvious. Since the whole graph topology is known to the central algorithm, this latter can launch random walks from a vertex and output a similarity score for each of the other vertices. In other words, the random walk acts as a clustering mechanism on its own to form the neighborhood. In P2P however, this can not be done because the knowledge of each peer about the P2P network is limited to its neighborhood.

In our decentralized algorithm, each peer first locally executes a neighborhood formation phase through clustering gossip protocols as described in Section 2.2. Once the protocol has converged, each peer holds in its view the rating information of the c closest peers according to the similarity measure used for clustering. Note that only peers that get a strictly positive similarity score are inserted in the view. When all the peers have selected their views, we define the *P2P network* (or the topology of it) as the network created by the peers connected via edges to the peers in their views. c is typically small with respect to the size of the network for scalability reasons. Gathering information from only c peers is not enough to achieve good precision and high coverage due to data sparsity. In order to obtain more data at a low network cost, each peer also uses information of the peers in the view of her neighbors. Therefore, we define the *neighborhood* of each user as the peers directly connected and the peers connected within a distance of two hops in the P2P network. Depending on the clustering function, the size of the neighborhood can be up to $c^2 + c$. We evaluate the size of the neighborhood on the MovieLens dataset in Section 6.2.

To compute a personalized score prediction for an item, a user a leverages all the scores that users in her neighborhood have assigned to that item. Each contribution is weighted to reflect the similarity between a and the corresponding user. The users in the neighborhood are modeled as Markov Chain graph vertices, and a random walk is applied on this graph. A Markov chain can be represented by a directed graph where vertices are the states of the chain and edges represent the transition probabilities from one state to another. In our case, the states symbolize the users in the neighborhood of a peer, let us say peer a. Since the vertices represent the users of the neighborhood, we call our algorithm *user-based random walk algorithm*. The neighborhood size will consequently be an important parameter of the algorithm, while in centralized algorithms it is always fixed to the size of the complete graph, i.e. the graph containing all users or items of the system. We will see in Section 6 that increasing

the neighborhood size until some threshold raises the precision of recommendations while keeping the execution time in a reasonable level. Intuitively, the benefit of random walks is to consider the whole graph topology when estimating the similarity between users, while classic similarity meaures may only take advantage of the explicit intersection between the rating vectors of each two users.

Let P^a be the transition probability matrix corresponding to the graph of user a's neighborhood. Each element p^a_{uv} of P^a represents the probability that u would ask v for recommendations. This probability is defined as the normalized similarity of the tail peer to the head. Another parameter $\beta \in (0, 1)$ is also added to the equation to consider the case where each peer jumps randomly to any other peer in the neighborhood during the random walk. Choosing very high values of β leads to assignment of equal transition probability towards all users in the neighborhood regardless of their similarity. It means that the ratings of all users will have the same weight in predictions. The value p^a_{uv} is computed using the following equation:

$$p^a_{uv} = (1-\beta)\frac{s'_{uv}}{\sum_{z \in K(a)} s'_{uz}} + \frac{\beta}{m}, \qquad s'_{uv} = \begin{cases} s_{uv} & \text{if } s_{uv} \geq 0 \text{ and } u \neq v \\ \gamma_u & \text{if } u = v \\ 0 & otherwise \end{cases} \tag{3}$$

$K(a)$ is the list of all users in the neighborhood of a, s_{uv} is the similarity between two users u and v. In the experiments presented in Section 6, s_{uv} is either Pearson correlation or Modified Pearson correlation. γ_u is the self loop parameter, modeling the case where a user answers the recommendation query before forwarding it to other users of the neighborhood. Since each user is logically more confident in her own opinion than that of any other user, we fixed γ_u as twice the similarity measure between u and the most similar user in her view.

The random walk starts from the users directly connected to the active user a, that is, peers having a one hop distance with the active peer in the network. The vector of initial probability distribution over the neighborhood is represented by $\vec{d_a}$. Each entry of $\vec{d_a}$ is defined as:

$$\vec{d_a}(v) = \frac{s''_{av}}{\sum_{z \in K(a)} s''_{az}}, \qquad s''_{av} = \begin{cases} s_{av} & \text{if } v \in \text{clustering view of } a \\ 0 & otherwise. \end{cases} \tag{4}$$

Since each user computes her own predictions, we omit the index of a for the sake of simplicity. We use a finite length random walk where each peer decides to continue the walk with probability α. In Markov chains, the probability of being in a state at step k depends only on its previous state. Therefore, the probability of being in state u at step k is:

$$\Pr(X_k = u) = \alpha \sum_{v=1}^{m} \Pr(X_{k-1} = v) p_{vu} = \alpha^k \sum_{v=1}^{m} \vec{d}(v) P^k_{vu}$$

where m is the size of the active peer's neighborhood, and P^k the power k of the transition probability matrix. This is equal to the inner product of the initial distribution vector by column u of the P^k matrix. The overall probability of being in state u is then:

$$\Pr(X = u) = \sum_{k=1}^{\infty} \alpha^k \vec{d} \cdot \vec{P}^k_{*u}.$$

At last, the final probability distribution vector over the neighborhood is:

$$\hat{R} = \sum_{k=1}^{\infty} \alpha^k \vec{d} P^k = \vec{d}\alpha P(I - \alpha P)^{-1}. \quad (5)$$

We use Equation (5) to estimate the final distribution vector, and α is optimized empirically. Note that even in the real implementation of the algorithm, Equation (5) may still be used instead of launching real random walks. Once the final distribution vector is output by the random walk model, its entries are used as similarity weights ω_{uv} in Equation (2) in order to generate the recommendations.

The computation of transition similarity matrix and the matrix inversion of Equation (5) are the main sources of complexity of the algorithm. The similarity must be calculated between each two users. The complexity of matrix inversion is $O(S^3)$, where S is the neighborhood size. Each similarity computation depends on the number of items in the neighborhood. If the set of items of the neighborhood gets close to the set of items in the whole system, the complexity of both operations becomes $O(S^2N + S^3)$. With a correct selection of neighborhood size, the algorithm gives excellent performance with reasonable execution time.

In the same way, we can also imagine applying the same algorithm on the graph of items, then having an item-based random walk algorithm. The complexity of this algorithm will be $O(N^2S + N^3)$. Unfortunately, the execution time of item-based random walk algorithm is far from being affordable for the peers in real settings. The lack of efficiency of item-based random walk algorithm pushed us through suggesting the user-based random walk as a better approach for P2P applications. Furthermore, we will see in next section that item-based approaches have in general poor results in P2P systems.

6 Experiments and Results

In this section we compare our algorithm with other decentralized CF algorithms. Besides, the behavior of the random walk is analyzed.

6.1 Evaluation Methodology and Results

In P2P systems the users do not report any feedback to a central server. As a result, no trace of real P2P data is available. In our experiments we use the

MovieLens 10,000,000 ratings dataset [1]. It consists of $10,000,054$ ratings on $10,681$ movies, rated by $71,567$ real users of the MovieLens website, where each user has rated at least 20 movies. A 5-star scale is used to ask for ratings. To the best of our knowledge, this is the second biggest dataset available after the Netflix dataset for research on recommender systems.

Since MovieLens is a central database, we adopt the following strategy to adjust it for our P2P experiments: For each user in the database, a peer object is instanced. This peer is attributed with the profile of the corresponding user in the database. This profile contains the list of films and corresponding ratings of the user. Consequently, each peer can access directly only her own ratings, and needs to rely on the epidemic protocol described in Section 2.2 to find and retrieve the profiles of similar peers. This strategy enables us to simulate a P2P network of MovieLens users, as if each of them had registered her ratings on her own computer instead of reporting them to the website.

We evaluate different recommender algorithms by cross validation. Namely, each MovieLens user profile is split into 20 regular random slices. 20 comes from the minimum number of ratings per user in the MovieLens dataset. Consequently, each profile slice contains at least one rating. A number of slices form the *training profile* input to the algorithm as the learning data. The predictions are made on the *test profile* composed of the remaining slices. Different levels of sparsity are modeled by changing the proportion of the test and training profiles.

We use Root Mean Squared Error (RMSE) to measure the *precision* of the recommendations. For user u it is defined as $\sqrt{(\sum_{r_{ui} \in I_{T_u}} (\hat{r}_{ui} - r_{ui})^2) / |\mathrm{I_{T_u}}|}$, where $|\mathrm{I_{T_u}}|$ is the size of the test profile of u. Each peer computes its own RMSE, and the total RMSE of the system is defined as the mean of RMSEs.

Coverage is another important measure of usefulness for recommender systems. It shows the proportion of items for which the algorithm can predict a rating. Since the total number of items of a P2P network is not known to the users, we define the coverage for user u as $(\left|\hat{I}_{T_u}\right| / |I_{T_u}|)$, where $\hat{I}_{T_u}$ is the set of predictable items in the test profile of u. The total coverage of the system is then defined as the mean coverage of all peers.

P-RW	decentralized user-based random walk algorithm described in Section 5
MP-U	decentralized version of the user-based algorithm in [7] with Modified Pearson correlation
P-U	decentralized version of the user-based algorithm in [7] with Pearson correlation
Tribler[2]	decentralized user-based approach with Cosine similarity and significance weighting
PocketLens [16]	decentralized item-based approach using Cosine similarity
MP-I	decentralized version of the item-based algorithm in [18] with Modified Pearson correlation
P-I	decentralized version of the item-based algorithm in [18] with Pearson correlation

Table 1: Short description of decentralized CF algorithms with their abbreviations

The simulations are run for three view sizes: 10, 20 and 30. All results were obtained after 30 cycles of gossip, and the epidemic protocol had converged. β was fixed to 0.15 in the random walk algorithm. α was optimized by trying

values in $(0, 1)$ with a step of 0.1 in different levels of sparsity. In general, we observe that the optimal length of the random walk increases (larger α) as the data becomes sparser. Even though the similarity between users is most often transitive, it happens in few cases that users in a two hop distance have negative similarity. We do not take such users into account when making predictions in user-based approaches, although they exist in the neighborhood. This problem never happens in the random walk algorithm because the similarity weights generated by the algorithm are non-negative probabilities. In the same way, only items with positive similarity are used for prediction in item-based methods.

We compare our algorithm with 6 decentralized recommender algorithms. The description of these algorithms and corresponding abbreviations are listed in Table 1. The best results, obtained with a view size of 30, is reported in Tables 2 and 3. The results for view sizes of 10 and 20 is found in the technical report [12]. The item scores computed by Tribler are not scaled. Hence, we generated a score for each item using Equation (2) to be able to compare it with other algorithms. In P-I and MP-I, both neighborhood formation and item-based prediction are done using the same type of similarity measure, and the predictions are made using the item-based version of Equation (2).

As seen in Table 2, P-RW algorithm outperforms all other decentralized algorithms when the sparsity is less than 70%. P-U and MP-U approaches significantly outperform all item-based approaches. Tribler shows the poorest performance among user-based approaches, but still improves over item-based approaches when sparsity is more than 5% and less than 25%. This shows that Pearson correlation is a better choice than Cosine similarity in user-based approaches. PocketLens shows the best performance among item-based approaches. Therefore, Cosine similarity seems to perform better in item-based approaches. Moreover, comparing MP-U and MP-I with P-U and P-I proves that significance weighting is efficient for both item-based and user-based approaches. As a general term, we can state that provided the right similarity measure is used, user-based approach is preferable to item-based approach in P2P recommenders. All methods have good coverage when the training profile is more than 5%. However, the coverage of the methods using significance weighting, that is MP-U, MP-I and Tribler, is slightly better than others. P-RW improves the coverage over P-U although they use the same neighborhood. This is because P-RW can also use the ratings of users with negative direct similarity.

In most recommender systems, the predicted scores are used to propose a recommendation list of top-N items to the user. The quality of this list strongly depends on the RMSE of the system. The achievable RMSE lies in a very restricted range in available datasets, but it is proven that only slight improvement in RMSE yields much more satisfactory recommendation lists [13]. Hence, the improvement of our algorithm over the best of previous algorithms is absolutely valuable specifically because we are very close to the limit of achievable RMSE.

The precision and coverage of all approaches increase with the size of the neighborhood. This is due to the fact that algorithms rely on more users for making predictions. We observed in simulations that increasing the view size over 30 does not yield any significant improvement. Note there is no advantage

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-RW	1.0719	1.0147	0.9869	0.9693	0.9575	0.9513	0.9423	0.9327	0.9196	0.8842
MP-U	1.1164	1.0481	1.0081	0.9841	0.9717	0.9662	0.9522	0.9408	0.9168	0.8752
P-U	1.1288	1.0594	1.0220	0.9980	0.9812	0.9725	0.9594	0.9477	0.9294	0.8903
Tribler	1.2301	1.0946	1.0439	1.0234	1.0166	1.0119	1.0050	0.9988	0.9892	0.9489
PocketLens	1.2036	1.1110	1.0595	1.0296	1.0119	0.9998	0.9833	0.9721	0.9553	0.9174
MP-I	1.2218	1.1410	1.0867	1.0493	1.0211	1.0011	0.9732	0.9559	0.9338	0.8985
P-I	1.2508	1.1601	1.0984	1.0524	1.0255	1.0062	0.9805	0.9656	0.9441	0.9038

Table 2: RMSE in different levels of sparsity, view = 30

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-RW	0.8429	0.9272	0.9370	0.9487	0.9492	0.9506	0.9560	0.9540	0.9511	0.9474
MP-U	0.8324	0.9642	0.9854	0.9917	0.9943	0.9956	0.9969	0.9979	0.9983	0.9986
P-U	0.6971	0.8657	0.892	0.9220	0.9264	0.9316	0.9415	0.9407	0.9394	0.9364
Tribler	0.7469	0.9669	0.9881	0.9933	0.9952	0.9966	0.9978	0.9984	0.9990	0.9993
PocketLens	0.7435	0.9023	0.9337	0.9453	0.9515	0.9549	0.9583	0.9598	0.9582	0.9558
MP-I	0.8265	0.9612	0.9853	0.9924	0.9951	0.9963	0.9974	0.9979	0.9985	0.9989
P-I	0.6872	0.8844	0.9192	0.9406	0.9434	0.9470	0.9543	0.9521	0.9488	0.9458

Table 3: Coverage in different levels of sparsity, view = 30

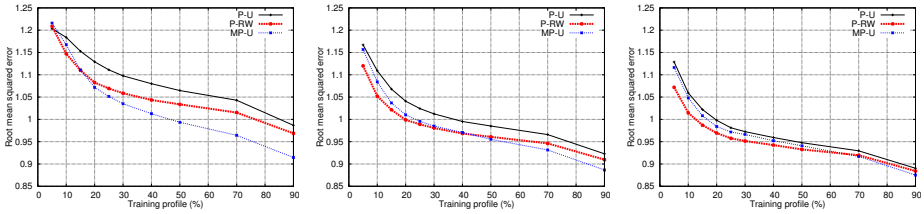
in choosing very large views. Not only does it exponentially increase the execution time, but also renders the recommendations less personalized. A view size about 30, allows for good precision and coverage while keeping the computation time quite affordable. This value may be different for datasets other than MovieLens.

6.2 Analysis of the Behavior of Random Walk

In this section we discover further the behavior of random walk in function of sparsity, neighborhood size and similarity measure.

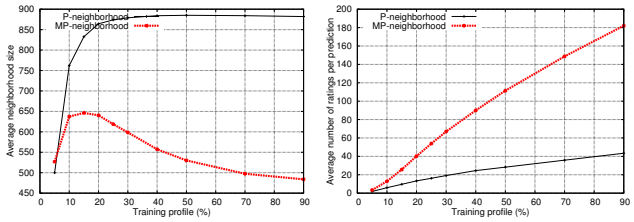
Random Walk vs. Sparsity Random walk works well when the data is so sparse that classic similarity measures fail to detect meaningful relation between users. By increasing the training set proportion, classic similarity measures deliver better performance than the random walk algorithm. For the view size of 30, P-RW gives the best results until when the training set proportion is below 70%. However, when the training set proportion goes beyond 70%, the direct similarities become more reliable than random walk similarities.

Random Walk vs. Neighborhood Size The precision of the three approaches with the best precision is plot in Figure 1. Before the training profile arrives at a threshold, P-RW delivers the best precision outperforming the MP-U algorithm as the second best approach. This threshold increases rapidly with incrementing the size of the neighborhood. It is 15% for a view size of 10, and goes up to 40% for a view size of 20. The threshold reaches 70% for the view size of 30, suggested as the best view size by our experiments. Furthermore, the amount of improvement of P-RW over other approaches increases with the neighborhood size. In fact, random walk reevaluates the similarity weight



(a) view size = 10 (b) view size = 20 (c) view size = 30

Figure 1: RMSE



(a) view size = 30 (b) view size = 30

Figure 2: MP-neighborhood vs. P-neighborhood

between users by mining longer paths in the neighborhood to find implicit transitive similarities. However, classic similarity measures can only capture direct similarity. The chance of detecting the transitive similarities is naturally higher for larger neighborhoods. It is why P-RW outperforms MP-U, but P-U has poorer precision than the latter, while both P-U and P-RW use the same type of neighborhood.

Random Walk vs. Similarity Measure To see how significance weighting of the similarity measure can influence the quality of the neighborhood, we compared the average neighborhood size and the average number of ratings per prediction for two types of neighborhood formed either through Pearson correlation or Modified Pearson correlation (see Figure 2). It is seen that MP-neighborhood has more ratings per prediction than P-neighborhood while its size is smaller. It means that Modified Pearson correlation prefers over-active users having rated a large number of items. Note P-RW has better precision than MP-U although it uses less ratings, showing that P-RW *learns* faster than MP-U. With increasing the training profile, the P-neighborhood approaches its maximum size (about 900) very soon. Unlike P-neighborhood, the size of MP-neighborhood decreases continuously when the training profile goes beyond 15%. This indicates that the P2P network becomes more clustered because the views of directly-connected peers contain many common neighbors. In fact, since over-active users have more ratings in each profile slice the significance

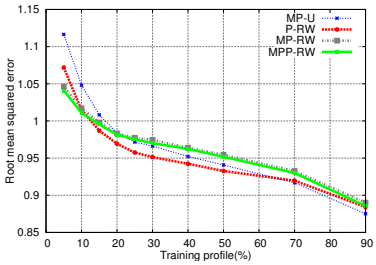


Figure 3: Error v/s proportion of data set used

weighting term grows faster for them with incrementing the training profile. Consequently, their chance being put in the neighborhood becomes more than moderate users, and their indegree increases quickly. We call it the *popularizing effect* of significance weighting.

Although the popularizing effect leads to better coverage, it prevents the random walk algorithm from working in two ways: first, it decreases the ability of the algorithm in similarity estimation by decreasing the neighborhood size and omitting users with few ratings but *implicit* similarity to the central user. Second, over-active users act as a *sink* in the Markov Chain model during the random walk. Then, their state probability at the end of the random walk is higher than other peers. In other words, random walk intensifies the influence of over-active users in predictions with respect to the users with less ratings. This significantly decreases the quality of random walk predictions in the MP-neighborhood. The size of MP-neighborhood has a peak when the training profile is 15%. This shows that the sinking behavior of significance weighting starts at this point. For smaller training profiles, the P2P network is not still well clustered, and peers continue to add new users to their views.

To investigate the performance of random walk on an MP-neighborhood, we implemented two new variants of our algorithm. The first variant is MP-RW. Being quite similar to P-RW, it uses Modified Pearson correlation instead of Pearson correlation for neighborhood formation and also as user similarity weight s_{uv} in Markov chain model (see Equations (3) and (4)). The second one is MPP-RW where the neighborhood is formed through Modified Pearson correlation, while user similarity weight in Markov Chain model is assigned using Pearson correlation. The results are plot in Figure 3. The exact RMSE values can be found in the technical report [12].

When the training profile is more than 15%, MP-RW starts to show poorer results than P-RW. Its performance is even worse than MP-U when the training set is more than 20%. The reason hides behind the popularizing effect of significance weighting. The slightly better precision of MPP-RW than MP-RW is due to the fact that the transition probability of the edges pointing towards over-active users decreases when significance weighting is not used for user similarity assignment. Hence, the sink role of such users is partly alleviated. It is also observed that MPP-RW can outperform P-RW when the training profile is

extremely sparse (below 15%). This is due to the fact that the sinking behavior of Modified Pearson correlation is not still severe in this range.

7 Conclusion

In this paper, we propose a user-based random walk algorithm to enhance the precision of previous decentralized CF recommender systems. We use epidemic protocols to assign each user with a neighborhood of similar peers. Each user locally runs the random walk algorithm on her neighborhood, and computes her recommendations. The algorithm is fully decentralized, and users are totally independent from each other in computing their own recommendations.

We implemented decentralized CF recommenders using different similarity measures and compared them with our algorithm. Our algorithm had the best precision over a wide range of sparsity. Decentralized user-based algorithms showed better precision and less complexity than their item-based counterparts. Moreover, Cosine similarity performed better in decentralized item-based algorithms, while Pearson correlation worked better for decentralized user-based algorithms.

Simulating a P2P network using the MovieLens 10,000,000 ratings dataset, we empirically showed how sparsity, neighborhood size, and similarity measure are determining parameters of the random walk algorithm. This algorithm delivers better precision when the data gets sparser. It works better for larger neighborhood sizes. The view size of 30 was given as a good trade-off between precision and execution time for MovieLens dataset. In the end, the behavior of the random walk was studied for two types of neighborhood formed either through Pearson correlation or Modified Pearson correlation. We showed how popularizing effect related to significance weighting term of Modified Pearson correlation is a barrier against the performance of random walk.

Acknowledgements We are very grateful to GroupLens research group for providing MovieLens datasets.

References

[1] *MovieLens Datasets*, 2010. `http://www.grouplens.org/node/73#attachments`.

[2] *Tribler*, 2010. `http://www.tribler.org`.

[3] Y. Azar, A. Fiat, A. R. Karlin, F. Mcsherry, and J. Saia. Spectral analysis of data. In *ACM symposium on Theory of computing*, pages 619 – 626, 2001.

[4] G. Biau, B. Cadre, and L. Rouviere. A stochastic model for collaborative recommendation. *The Annals of Statistics*, 2009.

[5] J. Canny and S. Sorkin. Practical large-scale distributed key generation. In *Advances in Cryptology*, pages 138–152, 2004.

[6] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.

[7] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *ACM SIGIR*, pages 230–237, 1999.

[8] M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *ACM SIGKDD*, pages 397–406, 2009.

[9] M. Jelasity, A. Montresor, and O. Babaoglu. T-man: Gossip-based fast overlay topology construction. *IJCNC*, 2009.

[10] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.

[11] M. J.Pazzani. A framework for collaborative, content-based and demographic filtering. In *Artificial Intelligence Review*, pages 393–408, 1999.

[12] A.-M. Kermarrec, V. Leroy, A. Moin, and C. Thraves. Addressing sparsity in decentralized recommender systems through random walks. Technical report, INRIA, 2010.

[13] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of the 14th ACM SIGKDD*, pages 426–434, 2008.

[14] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Recommendation systems: A probabilistic analysis. In *Proc. IEEE Symp. on Foundations of Computer Science*, 1998.

[15] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. In *Internet Computing, IEEE*, pages 76–80, 2003.

[16] B. N. Miller, J. A. Konstan, and J. Riedl. Pocketlens: Toward a personal recommender system. *ACM Trans. Inf. Syst.*, 22(3):437–476, 2004.

[17] J. S. Breeze, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Uncertainty in Artificial Intelligence*, 1998.

[18] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *World Wide Web*, pages 285–295, 2001.

[19] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *P2P*, pages 102–109, 2003.

[20] S. Voulgaris and M. V. Steen. Epidemic-style management of semantic overlays for content-based searching. In *EuroPar*, pages 1143–1152, 2005.

[21] H. Yildirim and M. S.Krishnamoorthy. A random walk method for alleviating the sparsity problem in collaborative filtering. In *Proc. of the ACM Conf. on Recommender systems*, pages 131–138, 2008.