Addressing Sparsity in Decentralized Recommender Systems through Random Walks

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

Addressing Sparsity in Decentralized Recommender Systems through Random Walks

Anne-Marie Kermarrec[1], Vincent Leroy[2], Afshin Moin[1], and Christopher Thraves[1]

[1] INRIA Rennes - Bretagne Atlantique, Rennes, France
[2] INSA de Rennes, UEB, Rennes, France

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 existing 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.

1 Introduction

Recommender systems are crucial to the success of e-commerce websites like Amazon[3], eBay[4] or Netflix[5]. They assist users in finding relevant content among thousands of items, increasing both the satisfaction of the users and the activity of the website. Typically, recommender systems keep track of the past behaviour of the users and leverage this information to predict the interest of a user for a given item. The research on this field was accelerated by the Netflix prize [2] on October 2006. The competition was about improving the precision of the

[3] `http:\\www.amazon.com`
[4] `http:\\www.ebay.com`
[5] `http:\\www.netflix.com`

Netflix existing recommender algorithm Cinematch by 10%. The prize was finally awarded in September 2009.

Different theoretical [8, 4, 16, 5] or empirical [10, 7, 17, 9, 14] 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, while *Content-based* recommenders as another important recommendation strategy require precise meta-data describing each of the items. In addition, users are mostly interested in products reported as good items by taste-wise users rather than simply receiving a list of items with similar content to their past purchases.

There are two major models of CF: *latent factor model* [14, 15] and *neighborhood model* [9, 20]. Latent factor models make users and items directly comparable by projecting them on the same space. The relevance of an item to a user is then inferred from user feedback and the latent factor space. Although factor models slightly outperform neighborhood models, this latter is the most widely used approach in 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. Such explanations are hard to give for latent factor models because the predictions are made by leveraging the whole information in the system, and not a restricted neighborhood.

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. Pearson correlation is traditionally the best known similarity measure used to judge the similarity between items or users while forming the neighborhood. Yet, it does not consider the number of common ratings. As a result, some users having a few ratings in common may be considered similar. To overcome this issue, some *significance weighting* term can be integrated into Pearson correlation to account for the number of common ratings, improving on the precision [9]. The neighborhood is consequently used with some prediction function in the rating estimation phase to estimate the score of unseen items for clients. Almost all the works in early days of CF were user-based algorithms. However, the item-based approach has recently received more attention in the domain of centralized recommender algorithms for its better scalability. More specifically, in most datasets, the number of users is larger and grows faster than the number of items. These schemes also benefit from better explainability than user-based algorithms 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 [6], which are out of the scope of this paper, can be deployed in decentralized environments to preserve the users' privacy by communicating 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. These decentralized networks have become very popular media for users to share their content. They meet the very same problem as e-commerce websites: the large amount of available content makes it very difficult for a user to find items meeting her needs. Hence, P2P networks would greatly benefit from recommender systems.

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 (user, 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 existing work on recommendation algorithms has been focused on centralized 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.* The P2P nature of the system we consider makes it challenging to develop recommender systems. Each user can only leverage her own information and data provided by a small (wrt the size of the system) number of other peers[6]. 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 quality and the coherence of meta-data. Hence, content-based recommenders can not be easily applied. The contributions of this paper are as follows:

First, decentralized user-based and item-based CF algorithms are implemented using different similarity measures, and their performance are compared in a P2P context. 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 in comparison with centralized settings. In our algorithm, each peer of the system is provided with a neighborhood composed of a small (wrt the size of the system) set of other 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

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

formed their neighborhood, i.e. the epidemic protocol has converged, each peer is thoroughly independent from other peers in generating her recommendations. This algorithm delivers 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 becomes specifically important in P2P context as it strongly affects the precision and complexity of the algorithm. The parameter values for which our algorithm gives the best performance are 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 behaviour of random walk is also analyzed in this section. Section 7 concludes the 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.

User-based CF recommender systems provide recommendations based on the similarity between users, while item-based systems compute the similarity between items. Item-based schemes have recently been the dominant neighborhood model in centralized CF. The most important advantage of centralized item-based methods is their scalability. Namely, the number of users is usually much larger than the number of items in recommender systems. Hence, user-based schemes become prohibitively computational intensive for a central server. It was shown in [20] that item-based schemes may even outperform user-based schemes.

In this section, we provide some background on the user-based [9] and item-based [20] CF methods. Moreover, epidemic protocols [12] are briefly discussed as the decentralization method we use to form a neighborhood of similar users.

2.1 User-based Collaborative Filtering

User-based CF is presented in [9]. In this approach, a neighborhood of similar users is assigned to each user using some *similarity measure*. One popular coefficient is *Cosine similarity*. Cosine similarity between two users u and v is defined as:

$$cos(u,v) = \frac{\sum_{i \in I_u \cap I_v} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u \cap I_v} r_{ui}^2} \sqrt{\sum_{i \in I_u \cap I_v} r_{vi}^2}} \tag{1}$$

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 behaviours. 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, a user 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* [9]. This is achieved by multiplying the Pearson correlation by a term reflecting the number of common items. In [9], 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. The logarithmic function is a damping function whose steep decreases constantly. Hence, it is more discriminating for smaller numbers of common items. We call this *modified Pearson coefficient* and define it as:

$$corr(u,v) = \rho_{uv} \log(|I_v \cap I_u|). \tag{2}$$

It is shown in Section 6 that significance weighting tends to *popularize* the users' neighborhood, i.e. although adding this factor improves the total precision of the system, this leads to omitting users having few but valuable ratings because over-active (having rated a lot of items) users are favored over the users with more similarity but less items in common.

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

$$\hat{r}_{ui} = \overline{r}_u + \frac{\sum_{v \in N(u,i)} \omega_{uv} (r_{vi} - \overline{r}_v)}{\sum_{v \in N(u,i)} \omega_{uv}} \tag{3}$$

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.

2.2 Item-based Collaborative Filtering

The item-based approach is presented in [20]. Similar to user-based CF, the item-based CF starts with assigning a neighborhood of similar items to each item. The item-based Pearson correlation is defined as:

$$\rho_{ij} = \frac{\sum_{u \in U_i \cap U_j}(r_{ui} - \overline{r}_i)(r_{uj} - \overline{r}_j)}{\sqrt{\sum_{u \in U_i \cap U_j}(r_{ui} - \overline{r}_i)^2}\sqrt{\sum_{u \in U_i \cap U_j}(r_{uj} - \overline{r}_j)^2}}$$

where $\overline{r}_i$ is the average rating for item i, and U_i and U_j the set of users having rated i and j respectively. Item-based Modified Pearson correlation is then defined as:

$$corr(i,j) = \rho_{ij} \log(|U_i \cap U_j|).$$

In [20] the prediction is done using the weighted sum function:

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i,u)} \omega_{ij} * r_{uj}}{\sum_{j \in N(i,u)} \omega_{ij}} \tag{4}$$

where $N(i,u)$ is the set of items user u has rated. This can eventually be improved in order to capture the offset corresponding to different user attitudes:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{j \in N(i,u)} \omega_{ij}(r_{uj} - \bar{r}_u)}{\sum_{j \in N(i,u)} \omega_{ij}} \tag{5}$$

ω_{ij} is the similarity weight between items i and j. In this paper, it is either of the similarity measures presented so far.

2.3 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) [12] 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 behaviour. 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* [21]. 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: they converge from any connected P2P network.

3 Related Work

In this section, we review the previous work on decentralized recommender systems and suggested solutions to sparsity. The need for effective decentralized recommender systems has been realized for some time. However, the research about them has remained modest with respect to the centralized algorithms. Notable works on the context are as follows: Tribler [3], 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. The rating vectors of the users are binary, i.e. the entry is 1 if the user has ever downloaded the corresponding item, and 0 otherwise. Tribler uses epidemic protocols to form the neighborhood. Cosine similarity (Equation (1)) with significance weighting is used to compute the similarity between users. The significance weighting term is defined as $(|I_v|/40)$, where v can be any of the neighbors. A non-normalized score is then computed for each item through user-based CF approach. These scores are consequently used to generate an ordered recommendation list. 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 [3], a decentralized search engine using BitTorrent protocol, is capable of giving personnalized 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 (Equation (1)) as similarity measure. The significance weighting term is defined as $(|I_v|/40)$, 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 [18] is a decentralized recommender algorithm developed by GroupeLens research group. In [18], different architectures from centralized to fully decentralized are suggested for neighborhood formation. PocketLens uses the

Cosine similarity to estimate the similarity between users. 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 the weighted sum in Equation (4). PocketLens [18] is a decentralized recommender algorithm developed by GroupeLens research group. In [18], different architectures from centralized to fully decentralized are suggested for neighborhood formation. PocketLens uses the Cosine similarity to estimate the similarity between users. 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 [13] 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. In addition, providing demographic data endangers the privacy of the users. *Default rating* [19] 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 that it creates very dense input data matrix, hugely increasing the complexity of the computations. Hence, this is not a proper solution for P2P either, because the complexity of the algorithm must remain affordable for personal computers and laptops, being in general less powerful 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 [11] 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 [22] 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. Namely, contrary to the central recommender algorithms, each user can only access to 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 2.3 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 computed locally at each user.

The decentralization of the item-based CF algorithm is more of a challenge because the algorithm needs the rating vector of the items in order 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. On the other hand, the scalability advantage of centralized item-based CF over centralized user-based CF algorithms is not valid in P2P networks. In fact, 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 complexity of CF algorithms is mostly due to the computation of similarity between users or items. For decentralized user-based CF algorithms, a vector containing the similarities between the central user and all other peers in the neighborhood must be calculated. The complexity of each similarity calculation depends on the number of items in common between two users. This number can 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 all unknown items of the neighborhood and the items the user has rated should be calculated to form a similarity matrix. We observed during simulations on MovieLens dataset, that provided the neighborhood is big enough, the number of items in the neighborhood can get close to the total number of items in the system. This issue rises because the neighborhood almost always contains some users having rated the majority of items. Hence, user u needs to compute $L(N - L)$ similarities where L equals $|I_u|$. The complexity of each similarity calculation is related to the size of the rating vector of the items. It can be up to the neighborhood size. Hence, the complexity of the decentralized item-based approach is $O(N^2S)$ for each peer. The worst case happens when all items reside in the neighborhood, and the user has already rated half of them, that is $L = N/2$. Comparing the complexity of the two approaches, it is clear 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 weight is computed through random walks.

Some centralized approaches [22] have used random walks to improve the precision of their algorithm. In general, the recommendation problem is modeled by a weighted and directed graph where vertices represent the entity of interest. This entity can be items in item-based recommenders or webpages in PageRank algorithm for example. The application of random walks to centralized recommenders is relatively obvious. More precisely, the topology of the whole graph is known to the central recommender algorithm. Therefore, the algorithm 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 aiming at neighborhood formation. In P2P however, random walks can not be used for clustering because the knowledge of each peer about the P2P network is limited to the peers of its neighborhood and the items they have rated.

In our decentralized algorithm, each peer first locally executes a neighborhood formation phase through gossip protocols. The users in the neighborhood are then modeled as a Markov Chain graph, and a random walk is applied on this graph. 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.

We now present in detail both components, neighborhood formation and random walk method, of our approach.

Neighborhood Formation Each peer of the P2P network maintains a view of peers sharing their rating behaviour. To this end, they rely on a clustering gossip protocol, as described in Section 2.3. 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 partial 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.

In a P2P system, for scalability reasons and network costs, c is typically small with respect to the size of the network. 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. We evaluate the size of the neighborhood on the MovieLens dataset in Section 6.3.

Random Walk Algorithm In order 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 in the neighborhood.

Each peer computes (or simulates) the random walk on its neighborhood with a Markov chain. 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. 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:

$$p^a_{uv} = (1 - \beta)\frac{s'_{uv}}{\sum_{z \in K(a)} s'_{uz}} + \frac{\beta}{m}, \tag{6}$$

where

$$s'_{uv} = \begin{cases} s_{uv} & \text{if } s_{uv} \geq 0 \text{ and } u \neq v \\ \gamma_u & \text{if } u = v \\ 0 & otherwise \end{cases}$$

$K(a)$ 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 the Pearson correlation or the 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 $\boldsymbol{d}_a$. Each entry of $\boldsymbol{d}_a$ is defined as:

$$\boldsymbol{d}_a(v) = \frac{s''_{av}}{\sum_{z \in K(a)} s''_{az}} \tag{7}$$

where

$$s''_{av} = \begin{cases} s_{av} & \text{if } v \in \text{clustering view of } a \\ 0 & otherwise. \end{cases}$$

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:

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

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:

$$\Pr(X_k = u) = \alpha^k (\boldsymbol{d} \cdot \boldsymbol{P}_{*u}^k).$$

The overall probability of being in state u is then:

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

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

$$\hat{R} = \sum_{k=1}^{\infty} \alpha^k \boldsymbol{d} P^k = \boldsymbol{d} \alpha P (I - \alpha P)^{-1}. \tag{8}$$

One way to obtain the final probability distribution vector is to launch several random walks on the graph, and take the average of all results. In our algorithm however, we use Equation (8) to estimate the final distribution vector. Parameter α is optimized empirically. Note that even in the real implementation of the algorithm, Equation (8) may still be used instead of launching 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 (3) in order to generate the recommendations.

The computation of transition similarity matrix and the matrix inversion of Equation (8) 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 user-based random walk algorithm with other decentralized CF algorithms. The behavior of random walk is also analyzed in Section 6.3.

6.1 Evaluation Methodology

In order to run valid experiments we first need to choose proper input data. 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]. This dataset was made available by GroupLens research group in January 2009 for research purposes about recommendations. 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 2.3 to find and retrieve the profiles of similar peers. This strategy enables us to simulate a P2P network composed of $71,567$ users residing in MovieLens dataset, 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. In this method, the profile of each user is partitioned into two disjoint subsets: *training profile* and *test profile*. The data in the training profile is input to the algorithm as learning data. The predictions are then made on the test profile, and compared against real ratings. To generate proper training and test profiles, we split the profile of each MovieLens user into 20 regular random slices, where 20 comes from the minimum number of ratings per user in the MovieLens dataset. Consequently, each user has at least one rating in each of her profile slices. The training profile of a user contains a given number of slices and the remaining slices (out of 20) form the test profile. Since our algorithm targets situations where the input data is sparse, we tried different levels of sparsity 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:

$$\mathrm{RMSE(u)} = \sqrt{\frac{\sum_{r_{ui} \in I_{T_u}} (\hat{r}_{ui} - r_{ui})^2}{|\mathrm{I_{T_u}}|}}$$

Where $|\mathrm{I_{T_u}}|$ is the number of items in the test profile of u. Since, our recommender system is decentralized, each peer computes its own RMSE, and the total RMSE of the system is defined as the mean of RMSEs.

While precision assesses the quality of recommendations, *coverage* is another important measure of usefulness for recommender systems. It is defined as the proportion of items in the database for which the recommender algorithm is capable of predicting a rating. Using the traditional definition of coverage is challenging for decentralized recommender systems because the total number of items in a P2P network is not known to the users. Hence, we define the coverage for user u as:

$$coverage(u) = \frac{\left|\hat{I}_{T_u}\right|}{|I_{T_u}|}$$

Where $\hat{I}_{T_u}$ is the set of predictable items in test profile of u. The total coverage of the system is then defined as the mean coverage of all peers.

6.2 Simulation Results

training set	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
α	0.7	0.9	0.8	0.7	0.8	0.6	0.5	0.5	0.5	0.3

Table 1. The value of optimal α for different levels of sparsity

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. The value of optimal α is given in Table 1. In general, the sparser the input data, the larger α. This implies that the length of the random walk should increase as the data becomes sparser. Despite the fact that the similarity between users is most often transitive, it happens in few cases that users in a two hop distance have negative similarity with the active user. 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 2. The best results, obtained with a view size of 30, is reported in Tables 3 and 4. The results for view sizes of 20 and 30 are summerized in Tables 5, 6, 7, and 8. The item scores computed by Tribler are not scaled. Hence, we generated a score for each item using Equation (3) to be able to compare it with other algorithms. In P-itembased and MP-itembased, both neighborhood formation

P-randomwalk	decentralized user-based random walk algorithm described in Section 5
MP-userbased	decentralized version of the user-based algorithm in [9] using Modified Pearson correlation
P-userbased	decentralized version of the user-based algorithm in [9] using Pearson correlation
Tribler[3]	decentralized user-based approach using Cosine with significance weighting
PocketLens [18]	decentralized item-based approach using Cosine
MP-itembased	decentralized version of the item-based algorithm in [20] using Modified Pearson correlation
P-itembased	decentralized version of the item-based algorithm in [20] using Pearson correlation

Table 2. Short description of decentralized CF algorithms with their abbreviations

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	1.0719	1.0147	0.9869	0.9693	0.9575	0.9513	0.9423	0.9327	0.9196	0.8842
MP-userbased	1.1164	1.0481	1.0081	0.9841	0.9717	0.9662	0.9522	0.9408	0.9168	0.8752
P-userbased	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-itembased	1.2218	1.1410	1.0867	1.0493	1.0211	1.0011	0.9732	0.9559	0.9338	0.8985
P-itembased	1.2508	1.1601	1.0984	1.0524	1.0255	1.0062	0.9805	0.9656	0.9441	0.9038

Table 3. RMSE in different levels of sparsity, view = 30

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	0.8429	0.9272	0.9370	0.9487	0.9492	0.9506	0.9560	0.9540	0.9511	0.9474
MP-userbased	0.8324	0.9642	0.9854	0.9917	0.9943	0.9956	0.9969	0.9979	0.9983	0.9986
P-userbased	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-itembased	0.8265	0.9612	0.9853	0.9924	0.9951	0.9963	0.9974	0.9979	0.9985	0.9989
P-itembased	0.6872	0.8844	0.9192	0.9406	0.9434	0.9470	0.9543	0.9521	0.9488	0.9458

Table 4. Coverage in different levels of sparsity, view = 30

and item-based prediction are done using the same type of similarity measure, and the predictions are made using Equation (5).

As seen in Table 3, P-randomwalk algorithm outperforms all other decentralized algorithms when the sparsity is less than 70%. P-userbased and MP-userbased 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 than Pearson correlation in item-based approaches. Moreover, comparing MP-userbased and MP-itembased with P-userbased and P-itembased approaches 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-userbased, MP-itembased and Tribler, is slightly better than other methods. The P-randomwalk algorithm improves the coverage over P-userbased although they use the same neighborhood. This is because P-randomwalk can also use the ratings of users having negative direct similarity.

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	1.1200	1.0519	1.0216	0.9988	0.9890	0.9806	0.9684	0.9608	0.9462	0.9097
MP-userbased	1.1567	1.0842	1.0368	1.0102	0.9957	0.9851	0.9706	0.9553	0.9313	0.8865
P-userbased	1.1670	1.1092	1.0679	1.0409	1.0243	1.0123	0.9953	0.9851	0.9659	0.9228
Tribler	1.2521	1.1362	1.0793	1.0563	1.0426	1.0392	1.0316	1.0249	1.0095	0.9658
PocketLens	1.2188	1.1455	1.0879	1.0518	1.0290	1.0133	0.9912	0.9786	0.9575	0.9174
MP-itembased	1.2375	1.1529	1.0991	1.0625	1.0352	1.0140	0.9852	0.9658	0.9401	0.9012
P-itembased	1.2658	1.1899	1.1281	1.0802	1.0495	1.0276	0.9969	0.9790	0.9538	0.9095

Table 5. RMSE in different levels of sparsity, view = 20

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	0.7454	0.8617	0.8771	0.8999	0.9004	0.9053	0.9157	0.9133	0.9072	0.9037
MP-userbased	0.7540	0.9449	0.9768	0.9857	0.9896	0.9919	0.9942	0.9955	0.9968	0.9975
P-userbased	0.5428	0.7543	0.8074	0.8512	0.8597	0.8708	0.8896	0.8899	0.8870	0.8857
Tribler	0.6662	0.9487	0.9811	0.9894	0.9927	0.9942	0.9963	0.9974	0.9985	0.9991
PocketLens	0.5401	0.7968	0.8623	0.8877	0.9006	0.9087	0.9169	0.9207	0.9194	0.9166
MP-itembased	0.7656	0.9478	0.9784	0.9876	0.9913	0.9933	0.9952	0.9963	0.9973	0.9979
P-itembased	0.5120	0.7897	0.8444	0.8833	0.8896	0.8983	0.9118	0.9096	0.9057	0.9016

Table 6. Coverage in different levels of sparsity, view = 20

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	1.2086	1.1473	1.1100	1.0826	1.0692	1.0587	1.0435	1.0337	1.0152	0.9686
MP-userbased	1.2160	1.1676	1.1108	1.0717	1.0516	1.0351	1.0127	0.9934	0.9643	0.9146
P-userbased	1.2035	1.1839	1.1529	1.1291	1.1111	1.0976	1.0800	1.0647	1.0423	0.9867
Tribler	1.2581	1.2076	1.1505	1.1128	1.0913	1.0763	1.0617	1.0507	1.0275	0.9790
PocketLens	1.2115	1.1957	1.1475	1.1065	1.0740	1.0507	1.0173	0.9972	0.9674	0.9166
MP-itembased	1.2564	1.1780	1.1197	1.0812	1.0549	1.0329	1.0037	0.9830	0.9545	0.9102
P-itembased	1.2538	1.2263	1.1783	1.1314	1.0982	1.0719	1.0335	1.0081	0.9710	0.9190

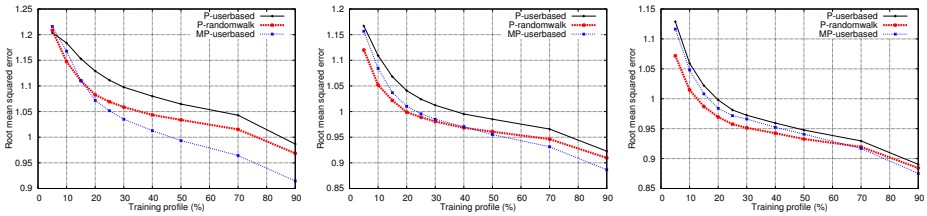
Table 7. RMSE in different levels of sparsity, view = 10

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	0.4312	0.6254	0.6681	0.7181	0.7273	0.7420	0.7717	0.7717	0.7676	0.7595
MP-userbased	0.4946	0.8470	0.9307	0.9557	0.9676	0.9747	0.9815	0.9851	0.9889	0.9911
P-userbased	0.2235	0.4392	0.5361	0.6156	0.6433	0.6698	0.7153	0.7221	0.7263	0.7227
Tribler	0.4587	0.8490	0.9344	0.9631	0.9748	0.9833	0.9901	0.9937	0.9970	0.9983
PocketLens	0.2126	0.4749	0.6115	0.6774	0.7152	0.7368	0.7641	0.7667	0.7830	0.7808
MP-itembased	0.5477	0.8808	0.9453	0.9645	0.9739	0.9793	0.9845	0.9874	0.9905	0.9921
P-itembased	0.2445	0.4985	0.6022	0.6795	0.7016	0.7251	0.7612	0.7664	0.7640	0.7596

Table 8. Coverage in different levels of sparsity, view = 10

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 [14]. 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 improvement is due to the fact that algorithms rely on more users for making predictions. We observed during simulations that increasing the view size over 30 does not yield any significant improvement. Note that there is no advantage 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



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

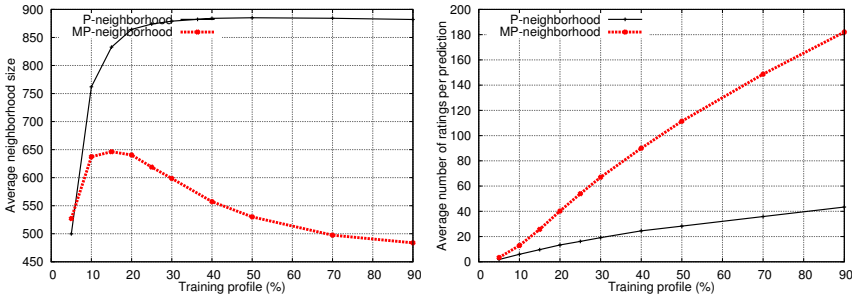
Fig. 1. RMSE

computation time quite affordable for the users. This value was computed for the MovieLens dataset, and may be different for other datasets.

6.3 Analysis of the Behaviour 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 are able to deliver better performance than random walk algorithm. For the view size of 30, P-randomwalk 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.



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

Fig. 2. MP-neighborhood vs. P-neighborhood

Random Walk vs. Neighborhood Size The precision of the three approaches having the best precision is plot in Figure 1. It is observed that before the training

profile arrives at a threshold, P-randomwalk delivers the best performance in terms of precision by outperforming the MP-userbased algorithm as the second best approach. This threshold increases rapidly with an increment in the size of the neighborhood. It is 15% for a view size of 10, where the neighborhood is very small 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 the improvement that the P-randomwalk has over other approaches increases with incrementing the neighborhood size. In fact, random walk reevaluates the similarity weight between users by mining longer paths in the neighborhood to find implicit transitive similarity between users. Classic similarity measures do not have such capability since they can only capture direct similarity. The chance for detecting such implicit relations is naturally higher for larger neighborhoods having more users. It is why P-randomwalk outperforms MP-userbased, but P-userbased has poorer precision than the latter, while both P-userbased and P-randomwalk use the same type of neighborhood.

Random Walk vs. Similarity Measure In this paragraph the effect of similarity measure is studied by applying the random walk algorithm on two types of neighborhood formed either through Pearson correlation or Modified Pearson correlation.

To see how significance weighting of Modified Pearson correlation can influence the quality of the neighborhood, and in turn the performance of the random walk, we compared the average neighborhood size and the average number of ratings per prediction for the two types of neighborhood (see Figure 2).

It can be 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 in average rated a large number of items. Note that P-randomwalk has better precision than MP-userbased although it uses less ratings. It means that P-randomwalk *learns* much faster than MP-userbased. With increasing the training profile, the P-neighborhood approaches very soon its maximum size, being about 900. Recall that the neighborhood contains peers within one and two hop distance on the P2P network. 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 directly-connected peers will have many common neighbors in their views. In fact, with incrementing the training profile proportion, the chance of over-active users for being put in the neighborhood becomes more than moderate users, because such users have more ratings in each slice of their profile, and the significance weighting term grows faster for them. Consequently, the indegree of such users increases with a rise in the training profile. As a result, they will exist more or less in the neighborhood of all users. This is what we call the *popularizing effect* of significance weighting, briefly introduced in Section 2.1.

Although the popularizing effect leads to better coverage, it avoids the random walk algorithm from working in two ways: first, it decreases the chance of the random walk for precise reevaluation of similarities by decreasing the neighborhood size and omitting users with few ratings but more *implicit* similarity

Training Profile	5%	10%	15%	20%	25%	30%	40%	50%	70%	90%
P-randomwalk	1.0719	1.0147	0.9869	0.9693	0.9575	0.9513	0.9423	0.9327	0.9196	0.8842
MP-randomwalk	1.0462	1.0170	0.9979	0.9832	0.9773	0.9745	0.9640	0.9544	0.9326	0.8900
MPP-randomwalk	1.0410	1.0109	0.9953	0.9816	0.9753	0.9700	0.9624	0.9517	0.9299	0.8856

Table 9. RMSE of three variants of the random walk algorithm, view = 30

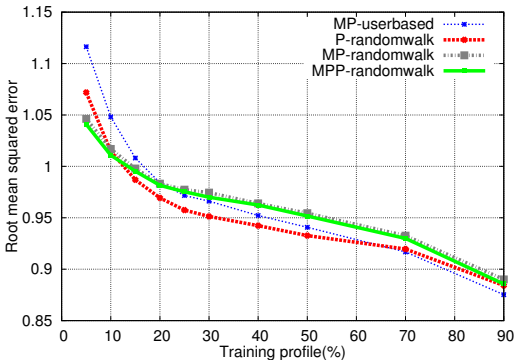


Fig. 3. Error v/s proportion of data set used

to the central user. Second, over-active users act as a *sink* in the Markov Chain model of the neighborhood during the random walk because a lot of peers point towards them with large probability. Hence, the final probability of stopping on them at the end of the random walk is higher than other peers in the neighborhood. In other words, random walk intensifies the influence of over-active users in making predictions with respect to the users with less ratings. This significantly decreases the quality of random walk predictions in the MP-neighborhood. The neighborhood 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. When the training profile is less than 15%, 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 decentralized user-based random walk algorithm. The first variant is MP-randomwalk. Being quite similar to P-randomwalk, 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 (6) and (7)). The second one is MPP-randomwalk where the neighborhood is formed by Modified Pearson correlation, while user similarity weight in the Markov Chain model is assigned using Pearson correlation. The precision of these approaches is compared with P-randomwalk in Table 9. The results are also plot in Figure 3.

It is seen in Figure 3 that when the training profile is more than 15%, MP-randomwalk starts to show much poorer results than P-randomwalk. Its performance is even worse than MP-userbased when the training set is more than 20%. The reason of this phenomenon hides behind the popularizing effect of sig-

nificance weighting. The precision of MPP-randomwalk is slightly better than MP-randomwalk. This is due to the fact that the transition probability of the edges pointing towards over-active users decreases when no significance weighting is used while assigning the user similarities. Hence, the sink role of such users is partly alleviated. It is also observed that MPP-randomwalk can outperform P-randomwalk 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 proposed 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.

Decentralized item-based and user-based approaches were implemented using different similarity measures and compared with our algorithm. Our algorithm showed the best precision over a wide range of sparsity. Decentralized user-based CF algorithms showed better precision and less complexity than their item-based counterpart. Moreover, Cosine similarity performed better in decentralized item-based algorithms, while Pearson correlation was a better choice 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 for 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 of 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. *Netflix Prize*, 2010. `www.netflixprize.com/`.
3. *Tribler*, 2010. `http://www.tribler.org`.
4. B. Awerbuch, Y. Azar, Z. Lotker, and B. Patt-Shamir. Collaborate with strangers to find own preferences. *Theory of Computing Systems*, 42(1), 2008.

5. Y. Azar, A. Fiat, A. R. Karlin, F. Mcsherry, and J. Saia. Spectral analysis of data. In *Proc. of the thirty-third annual ACM symposium on Theory of computing*, pages 619 – 626, 2001.
6. J. Canny and S. Sorkin. Practical large-scale distributed key generation. In *Advances in Cryptology*, pages 138–152, 2004.
7. M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.
8. P. Drineas, I. Kerenidis, and P. Raghavan. Competitive recommendation systems. In *Proc. of the thiry-fourth annual ACM symposium on Theory of computing*, pages 82–90, 2002.
9. J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. of the 22nd annual Int. ACM SIGIR*, pages 230–237, 1999.
10. T. Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, 2004.
11. M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proc. of the 15th ACM SIGKDD*, pages 397–406, 2009.
12. 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.
13. M. J.Pazzani. A framework for collaborative, content-based and demographic filtering. In *Artificial Intelligence Review*, pages 393–408, 1999.
14. Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of the 14th ACM SIGKDD*, pages 426–434, 2008.
15. Y. Koren. Collaborative filtering with temporal dynamics. In *Proc. of the 15th ACM SIGKDD*, pages 447–456, 2009.
16. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Recommendation systems: A probabilistic analysis. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 664–673, 1998.
17. G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. In *Internet Computing, IEEE*, pages 76–80, 2003.
18. B. N. Miller, J. A. Konstan, and J. Riedl. Pocketlens: Toward a personal recommender system. *ACM Trans. Inf. Syst.*, 22(3):437–476, 2004.
19. J. S. Breeze, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, 1998.
20. B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proc. of the 10th Int. Conf. on World Wide Web*, pages 285–295, 2001.
21. S. Voulgaris and M. V. Steen. Epidemic-style management of semantic overlays for content-based searching. In *EuroPar*, pages 1143–1152, 2005.
22. 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.