ROYAL INSTITUTE
OF TECHNOLOGY

Distributed Trust-Aware Recommender Systems

STEFAN MAGUREANU

Master's Thesis at ICT
Supervisors: Nima Dokoohaki, Shahab Mokarizadeh
Examiner: Mihhail Matskin

TRITA xxx yyyy-nn

Abstract

Collaborative filtering(CF) recommender systems are among the most popular approaches to solving the information overload problem in social networks by generating accurate predictions based on the ratings of similar users. Traditional CF recommenders suffer from lack of scalability while decentralized CF recommenders (DHT based, gossip based etc.) have promised to alleviate this problem. Thus, in this thesis we propose a decentralized approach to CF recommender systems that uses the T-Man algorithm to create and maintain an overlay network that in turn would facilitate the generation of recommendations based on local information of a node. We analyze the influence of the number of rounds and neighbors on the accuracy of prediction and item coverage and we propose a new approach to inferring trust values between a user and its neighbors. Our experiments on three important datasets show an improvement of prediction accuracy relative to previous approaches while using a highly scalable, decentralized paradigm. We also analyze item coverage and show that our system is able to generate predictions for significant fraction of the users, which is comparable with the centralized approaches.

Contents

Chapter 1

Introduction

Recommender systems are a popular solution to the ever more present problem of information overload. Information overload consists of users having difficulty identifying information relevant to their interest from a large set of data with which they are presented. The most common use of such systems is in e-commerce applications, where users can receive suggestions of items to purchase based on what similar clients have previously viewed or purchased, thus reducing the need of customers to browse the vast database of items. Recommender systems are also making their way into the domain of social networks, in this case suggesting other members of the network that they might be interested in.

Two of the most popular types of recommender systems are collaborative filtering(CF) and those based on matrix factorization(MF) [33]. CF relies on the presumption that, in real life, users have more trust in recommendations from friends and people with similar interests. Thus, CF strategies put higher weight on proposals received from more similar users or friends, in the case of social networks. The drawback of this approach is the necessity of having information about users in order to determine the relation between one another so prediction quality is dependent on how much information we have on the participants. Matrix factorization-based algorithms are often employed when the data on users is very scarce and can generate more predictions than CF approaches. In practice, information on users is not usually abundant, this aspect contributing to the growth in popularity of matrix factorization-based algorithms. Most of the current implementations of CF and matrix factorization-based recommenders are centralized, meaning that generating recommendations and calculating trust requires computational efforts proportional to the size of the social network they are using. Modern social networking sites can have an extremely large number of users, deeming centralized recommender systems impractical and costly to run. Thus, decentralized recommenders are needed in practice.

Decentralized approaches use techniques borrowed from P2P systems such as DHTs or gossip-based algorithms. Gossip-based recommender systems rely on epidemic network overlay algorithms to create and maintain a decentralized network

in which nodes can use local information to generate recommendations. The main advantage of this paradigm is its intrinsic scalability. However, because only local information is used, on demand recommendations are generated much faster than in the centralized paradigm where the whole dataset needs to be evaluated. Since the overlay maintenance algorithms used in the case of gossip-based recommender systems generally aim to group similar users together, using local information will not result in significant loss of accuracy, and in many such cases proves to have a positive influence on recommendation accuracy. To further reduce the errors, as in conventional recommender systems, trust-awareness can be used to complement this method.

In this thesis, we will address the scalability problem of the most popular centralized trust-based recommenders by integrating methods widely used in P2P systems. We propose grouping users in an overlay network maintained using the T-Man algorithm [15] and we use a novel trust inference model to improve the accuracy of prediction over previous trust metrics. We show the improvements our approach achieves compared to other methods and analyze its performance over two datasets. We also investigate how a user's prediction is affected by the growth of its direct neighborhood and how the system performs in terms of item coverage.

In the following chapter we will describe previous work in the field of trust-aware recommender systems with a focus on decentralized CF. In chapter 3 we will present our approach to creating the network overlay and to the computation of trusts between neighboring nodes. Chapter 4 will be dedicated to accommodating our experimental setup while chapter 5 will contain the results of our experiments and evaluation of the performance of the system. The final chapter will be dedicated to the conclusions and future work.

Chapter 2

Background

In this section we will be briefly describing previous work in the domain of recommender systems with a focus on decentralized approaches and trust inference techniques.

2.1 Matrix Factorization-Based Techniques

(Non-negative)Matrix factorization(MF) [31] is a class of methods employed for filling cells in a data matrix with approximated values. MF does not refer to a single algorithm, many different methods have been developed over time that rely on the same idea (singular value decomposition, principal component analysis, etc.). MF relies on the idea that, given a matrix M of size $m \times n$ containing a number of filled(non-null) cells, we can make educated guesses on the empty cells by considering M the result of the product of two other matrices U and V of sizes $m \times l$ and $l \times n$ respectively. The main idea behind these approaches is that iteratively adjusting the values contained by U and V until:

$$for\ all\ (i,j) \qquad M'[i][j] \approx M[i][j] \tag{2.1}$$

where $M' = U \times V$ and (i, j) are the coordinates of the filled data cells in the M matrix, will result in M' being a good approximation of the completed matrix M. The vectors V_i and U_j are called $latent\ feature\ vectors$.

If we choose to look at a recommender system as a system that fills in the empty data cells in a $user \times item$ matrix with educated guesses, we can immediately see how MF is a very good match for this type of problems. The main factor that determines the performance of MF methods however is the density of known values in the M matrix. The higher the density, the harder it would be to adjust the values in U and V so that the product would give a good approximation of the known values. The difficulty would imply both higher computation costs but also the resulting matrix M' is likely to be a worse approximation as we would have to satisfy more constraints. This is the main reason why MF is employed in recommender systems only in the case of very sparse datasets, with limited information on users. In this

context however, this problem of MF techniques does not have serious negative consequences since most real life datasets are very sparse.

Probabilistic Matrix Factorization

Probabilistic Matrix Factorization(PMF) has been proposed by Salakhutdinov and Mnih in [33] as a feasible application of MF to recommender systems. The following are presented as described by Salakhutdinov in his work. PMF is a probabilistic linear model with Gaussian observation noise. Given a recommender system dataset containing n users and m items, let R being the rating matrix $n \times m$, with $R_{i,j}$ being rating user i assigned to item j. In this case, U_i and V_j are the l-dimensional user specific latent feature vector and item specific latent feature vector respectively. The conditional distribution over the known ratings $R \in \mathbb{R}^{m \times n}$ and the prior distributions $U \in \mathbb{R}^{m \times l}$ and $V \in \mathbb{R}^{l \times n}$ is given by the following expression:

$$p(R|U,V,\alpha) = \prod_{1 \leq i \leq n} \prod_{1 \leq j \leq m} [\aleph(R_{i,j}|U_i V_j, \alpha^{-1})]^{I_{ij}} \tag{2.2}$$

$$p(U|\alpha_U) = \prod_{1 \leq i \leq n} \aleph(U_i^T|0, \alpha_U^{-1} I) \tag{2.3}$$

$$p(V|\alpha_V) = \prod_{1 \leq j \leq m} \aleph(V_j|0, \alpha_V^{-1} I) \tag{2.4}$$

where $\aleph(X|y, \alpha^{-1})$ denotes the Gaussian distribution with mean y and precision α and I_{ij} is equal to 1 is user i rated items j and 0 otherwise.

Adjusting U and V is done by learning the model, meaning we need to maximize the log-posterior function over the item and user feature vectors with fixed hyperparameters, meaning the observation noise variance and the prior variances. Thus, we need to maximize the following function:

$$ln\ p(U,V|R,\alpha_U,\alpha_V,\alpha) = ln\ p(R|U,V,\alpha) + ln\ p(U|\alpha_U) + ln\ p(V|\alpha_V) + C \tag{2.5}$$

where C is a constant that does not depend on any of the parameters.

Maximizing the equation at (2.5) is equivalent to minimizing the following expression:

$$\begin{aligned} E = \frac{1}{2} \times \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} I_{ij} \times (R_{i,j} - U_i \times V_j)^2 \\ + \frac{\lambda_U}{2} \times \sum_{i \in 1\ to\ n} \|U_i\|_{Fro}^2 \\ + \frac{\lambda_V}{2} \times \sum_{j \in 1\ to\ m} \|V_j\|_{Fro}^2 \end{aligned} \tag{2.6}$$

where $\|X\|_{Fro}^2$ denotes the Frobenius norm and $\lambda_U = \alpha_U/\alpha$ and $\lambda_V = \alpha_V/\alpha$. This expression represents the *sum-of-squares error function with quadratic regularization*

terms. The minimum of the above expression can be determined by performing gradient descent on U and V.

The biggest problem with this approach is the need to select λ_U and λ_V before the algorithm is started, in order to control the complexity and allow the model to generalize well. A way to overcome this drawback is to attempt to maximize the expression in equation (2.5) over *both* parameters and hyperparameters. Although this method proves to work in practice, it is not yet well-grounded theoretically, according to Salakhutdinov [33].

2.2 Decentralized CF

Using peer-to-peer techniques in the context of distributed recommender systems has been considered in several other works. This paradigm shift is common when dealing with very large databases such as the case of social networks, due to its intrinsic scalability. Research into the field of recommenders has also shown interest in decentralized approaches as a replacement for the more popular centralized techniques. In [42], Ziegler presents a detailed analysis of the challenges related to decentralized recommenders and proposes a framework for implementing such systems. The two main types of CF recommenders are *memory-based* and *model-based*. Memory-based CF recommenders use the ratings of a subset of users to generate recommendations, usually the most similar users to a user or its neighbors in the network. This method is usually referred to as user-based CF. A variation of this method uses the same principle but it predicts the rating of an item based on similar items rather than users. This method is referred to as item-based CF(or content-based CF). Model-based CF recommenders rely on creating a model from a collection of users and items. The resulting model can then be used to make recommendations without the need of storing the collection from which it was inferred. The drawback of this approach is that creating a model is usually more complicated and harder to implement than simply using ratings expressed by other users directly, as in the case of memory-based recommenders.

2.2.1 Model-based Approaches

Random walk models, such as those presented by M. Gori in [11] have been suggested as a solution to sparsity in CF recommender systems. Random walkers work by exploring the social network, starting from a user and taking a probabilistic path outwards into the network until it reaches a certain depth. The probability of a path being chosen is usually dependent on trust values between nodes. During this exploration, the walker uses the ratings of encountered users to create a model that can later be used for recommendations. In [26], B. N. Miller proposes a gossip-based recommender system in which nodes exchange ratings with a neighbor at each step in order to construct an item to item similarity matrix which can then be used to make offline predictions. The choice of neighbors as well as determining the neighbors of a user are implementation-dependent in this approach. Unlike our approach, in [26],

Miller does not maintain an overlay network. This is understandable since his proposed system does not need to keep similar profiles easily accessible and only needs a profile for a one-time computation, after which it can be discarded. In our approach, the most similar profiles must be consulted for every recommendation, meaning an overlay network is needed in order to keep those profiles easily accessible.

ItemRank

ItemRank [11] is an example of a random walk algorithm that relies on the model expressed by a *Correlation Graph* to predict user preferences. A *Correlation Graph* (henceforth noted C_G) is a graph in which nodes correspond to items and an edge between two nodes exists only if $C_{i,j} > 0$. Where

$$C_{i,j} = \frac{\overline{C_{i,j}}}{\omega_j} \tag{2.7}$$

$\overline{C}$ being the *Correlation Matrix* of size $nr\_items \times nr\_items$ and $\overline{C_{i,j}}$ contains the number of users that have rated both items i and j and ω_j is the sum of the j-th column of $\overline{C}$. It is important to note that in the resulting graph, the weight of the edges between nodes corresponding to items i and j will be equal to the value stored in $C_{i,j}$. The C_G can therefor be used to determine the relative correlation between the items of the dataset. The algorithm can then spread a user's preferences through the C_G given a training set of rated items by that individual. This phenomenon is referred to by the Gori et al. as a *preference flow*. The *preference flow* should be controlled in terms of attenuation and propagation in order to generate high score values to items that are similar to highly rated items in the training set.

There are two main assumptions that are made in order to determine a good spreading algorithm for this model. First, if an item i_n is similar to multiple items that have received high ratings from a user, it should be attributed with a similar high rating for the same user. This assumption corresponds to what we referred to as propagation in the above paragraph. The second effect that the algorithm needs to consider is attenuation. Items with high ratings have to propagate their positive influence through the graph, however, its influence should be reduced as it propagates outwards into the network. When propagating, the node's influence should also take into account the weight of edge it is traversing.

The PageRank algorithm [32] fulfills both requirements stated above and can thus be applied in this context. Aside from having the desired qualities, PageRank is also very efficiently computed due to extensive research on the algorithm [17]. Given a graph $G = \{N, E\}$ where N represents the set of nodes and E represents the set of edges, PageRank can compute the importance score of each node in $n \in N$, $PR(n)$. The approach takes into account graph connectivity, a node having a high PR score if it is connected to important nodes that have a low out-degree. As presented in [11], the following formula is used to compute the PR score:

$$PR(n) = \alpha \times \sum_{q:(q,n)\in E} \frac{PR(q)}{\omega_q} + (1-\alpha) \times \frac{1}{N} \tag{2.8}$$

where ω_q is the out degree of node q and α is the decay factor. A popular value for α is 0.85. Equation (2.8) can be re-written in matrix form to offer a more intuitive view on how the PageRank formula can be ported to ItemRank context [11]:

$$PR(n) = \alpha \times M \times PR + (1 - \alpha) \times d \tag{2.9}$$

where M is a stochastic matrix of non-negative values that meets the requirement that values of every column sum up to 1. d is a vector of non-negative values which also sum up to 1. The equation of this form can then easily be translated into the terms of the ItemRank recommendation algorithm:

$$IR_{u(i)} = \alpha \times C \times IR_{u(i)} + (1 - \alpha) \times d_{u(i)} \tag{2.10}$$

where $d_{u(i)}$ is chosen differently for each user $u(i)$ in order to achieve custom $IR_{u(i)}$ scores for each user. Extensive research has been done into the effect of tuning the d vector in order to create bias in the PageRank algorithm in order for the algorithm to better fit recommendation models. ItemRank also uses bias in PageRank to obtain better performance. $d_{u(i)}$ is defined by the formula $d_{u(i)} = \frac{\overline{d_{u(i)}}}{|\overline{d_{u(i)}|}}$, meaning $d_{u(i)}$ is the normalized version of $\overline{d_{u(i)}}$ with respect to the j-th item, defined in [11] by:

$$\overline{d_{u(i)}} = \begin{cases} 0 \quad if \quad t_{i,j} \notin TS \\ r_{i,j} \quad if \quad t_{i,j} \in TS \wedge t_{i,j} = (u(i), item_j, r_{i,j}) \end{cases} \tag{2.11}$$

The ItemRank algorithm can also be computed iteratively. According to Gori et al. it would only require around 20 iterations to converge and is would use the following formula:

$$\begin{cases} IR_{u(i)}(0) = \frac{1}{|M|} \times 1_{|M|} \\ IR_{u(i)}(t+1) = \alpha \times C \times IR_{u(i)}(t) + (1 - \alpha) \times d_{u(i)} \end{cases} \tag{2.12}$$

The interpretation of IR score is intuitive, the higher the IR score for a given item, the higher the probability that the user computing it would give said item a higher rating and would prefer it to an item with a lower IR score.

PocketLens

In [26], B. N. Miller proposes a peer-to-peer recommender system in which nodes periodically exchange ratings with a neighbor in order to construct an item to item similarity matrix which can then be used to make offline predictions. This approach is presented in our thesis as described by Miller in [26]. The choice of neighbors as well as determining the neighbors of a user are implementation-dependent in this approach. Unlike our approach, in [26], Miller does not maintain an overlay network. This is understandable since his proposed system does not need to keep similar profiles easily accessible and only needs a profile for a one-time computation, after which it can be discarded. In our approach, the most similar profiles must

be consulted for every recommendation, meaning an overlay network is needed in order to keep those profiles easily accessible.

The main idea behind PocketLens is to bring recommender systems to hand-held devices, meaning reducing the system to a small model that can operate on the limited resources of pocket-size devices. Creating an item-to-item similarity matrix fits this purpose since it would make recommendations based only on items that are similar to other items the user has rated. Not being dependent on the ratings of other users after the matrix has been created would make offline predictions easy to generate.

At the heart of the PocketLens algorithm lies a similarity model that comprises of several similarity metrics between items. These metrics rely on the rating history of users for the specified items. These metrics are implementation-dependent, so any popular similarity metric can be used, including correlation, cosine similarity and conditional probability. Miller determined that cosine similarity offers the best performance out of the popular approaches listed above, so the algorithm implemented in [26] is based on this measure. The formula for cosine similarity between items i and j is given by:

$$sim(i,j) = \frac{\vec{i} \times \vec{j}}{\|\vec{i}\| \times \|\vec{j}\|} \tag{2.13}$$

where $\vec{x}$ represents the vector of ratings in the user space for item x.

The similarity model can be represented as a matrix M of size $nr\_items \times nr\_items$ where the cell at coordinates (i, j) represents the similarity between items i and j. To make better use of the reduced resources on mobile devices, the rows of items that have not been rated by the local user can be removed from the matrix, leaving the matrix of size $nr\_rated\_items \times nr\_items$. This will greatly reduce the load on the memory and could be combined with a method of sparse matrix storage to further improve performance, since M would be very sparse in a real-life scenario. Each cell in M would comprise of 4 values: the dot product of the rating vectors involved(denoted as $PartialDot$, the vector lengths of the column vectors of the two items(denoted as $PtLen_i$ and $PTLen_j$) and a counter $Cooccur$ representing the number of neighbors with which the node agrees strongly.

Presuming C is the set of rated items by a neighbor and $\vec{c}$ is the vector of corresponding ratings, in order to ensure that all neighbors have relatively the same influence on the mode, the ratings vector $\vec{c}$ needs to be normalized so that $\|c\| = 1$. At each round, rating information is exchanged with another peer in the network(which we refer to as a *neighbor*, in the case of this algorithm), the goal being to update the M matrix. At each exchange, a node is interested in updating the cells in a given set C such that $C \cap O \neq \emptyset$, where O is the set of items rated by the owner and C is the set of items rated by the neighbor. For each cell (O_i, N_j) in M, where $N = C - O$, the values contained at the specific coordinates in the matrix are updated as follows:

$$PartialDot_{i,j} += r_i \times r_j$$
$$PtLen_i += r_i^2$$
$$PtLen_j += r_j^2$$
$$Cooccur += 1$$

At any point during the network exploration, the recommendation list can be computed by calculating the similarity scores for all the items using the following formula:

$$sim(i,j) = k \times \frac{PartialDot_{i,j}}{\sqrt{PtLen_i} \times \sqrt{PtLen_j}} \tag{2.14}$$

where k represents the *significance weighting factor* and has the purpose of reducing the influence of neighbors with which a node agrees strongly yet share a reduced number of rated items. This factor can be computed given the $Cooccur$ component of M's cells as follows:

$$k = \begin{cases} 1 & if\ Cooccur \geq 50 \\ Cooccur/50 & otherwise \end{cases} \tag{2.15}$$

Once the computation is finished, the sum of all columns corresponding to unrated items are calculated. The items can then be ranked by their total column score, higher score meaning a higher possibility that the user would highly rate the item corresponding to the column. Predictions for particular items can be generated as weighted average of the scores in the corresponding column.

2.2.2 Memory-based Approaches

In [12], a DHT-based (Distributed Hash Table) approach is suggested, where the central dataset is organized into "buckets" of users which can be saved on individual nodes, each user using his most suitable "bucket" to chose neighbors with which to generate predictions. In [34], user clustering is suggested as a solution for solving scalability problems as well as a means of improving accuracy. Unlike our approach, Sarwar et. al. [34] presents clusters as groups of users where all the users in a cluster are each other's neighbors, whereas in our case, the "neighbor" relation is directional. A directional "neighbor" relation is desirable since, while a user's neighbors will be the most similar users to it, there might be others that are more similar to a neighbor than said user.

M. Jamali in [14] introduces a memory-based random walk algorithm that uses several random walks to gather information about the rating of an item in a nodes vicinity in a trust network. Ormandi et. al. [30] determines that using gossip based algorithms to cluster a network in the context of recommender systems offers potential for increasing accuracy of prediction. This is particularly interesting for

our work since in [30] the main algorithms being tested are variations of the T-Man algorithm. However the aforementioned work does not analyze item coverage and does not cover trust-awareness in recommender systems, instead focusing on load-balancing.

PipeCF

The PipeCF algorithm [12] proposed by Han et al. attempts to use techniques borrowed from P2P to model a decentralized DHT-based CF recommender system. DHT overlay networks are networks in which nodes are responsible for storing a certain range of key-value pairs in such a way that the value associated with a key can be retrieved transparently by any node in the network. Thus, any DHT algorithm requires splitting its database among a number of peers. In the case of PipeCF, the database is split into units denoted *buckets*.

In order for the DHT to work, each bucket requires an identifier that allows other nodes to determine its location. The way Han et al. split the database is designed to have every bucket contain a group of user entries which have rated at least one item the same. The resulting buckets would then be identified by tuples of <item_id><rating>. Since PipeCF is a Collaborative Filtering recommender, each node will choose neighbors from based on *similarity*. *Similarity* in this case is defined as users who share at least one bucket with the node. Thus, when a user requires a prediction, it will receive suggestions from any node that has at least one item rated the same as the user. This would reduce the number of users involved in a prediction process by about 50% according to Han et al. compared to traditional CF algorithms, without sacrificing item coverage. While this is still an improvement over centralized CF approaches, in [12], Han et al. also presents two optimizations for the PipeCF.

1) Significance Refinement

Since in the basic approach described above all the nodes in a bucket are used by a user trying to make a prediction, the scalability of the algorithm would be impeded by nodes that have very few items in common rated the same. This would lead to both an increase in complexity and also a likely loss of accuracy as some users may have some items rated the same by pure coincidence and otherwise have widely different preferences. Also, Han et al. point to the fact that universally liked/disliked items are not as relevant as less common items in capturing the taste of a user. The obvious optimization in this case is to limit the number of users returned from a bucket. This change results in not only a dramatic reduction in returned users, which leads to reduced complexity, but also to improved accuracy.

2) Unanimous Amplification

Inspired from Breese et al. [4], Han et al. conclude that an increase in performance can be obtained by amplifying the weights of more similar users(those that are close to 1). In the case of PipeCF, Han [12] proposes the following transformation of the

estimation weights:

$$weight_{i,j} = \begin{cases} weight_{i,j} & if\ N_{i,j} = 0 \\ weight_{i,j} \times \alpha & if\ 0 < N_{i,j} \leq \gamma \\ weight_{i,j} \times \beta & if\ \gamma < N_{i,j} \end{cases} \tag{2.16}$$

where $N_{i,j}$ is the number of buckets that users i and j share. Han et al. mention that in their experiments they used $\alpha = 2$, $\beta = 4$ and $\gamma = 4$. The weight are used intuitively: once the neighbors are selected from the buckets, to generate a prediction, the weighted sum of the rating of each neighbor of the requested item will represent the system's guess on the item's rating for the soliciting user. This optimization results in higher prediction accuracy compared to the basic PipeCF algorithm.

TrustWalker

Walkers, such as TrustWalker [14] are usually employed as a solution for making predictions to cold start users (users with very few, or no rated items). Usually, these types of recommenders imply a trade-off between accuracy and coverage, as the further the walker explores the network, the less reliable the ratings will be and the higher the chances of discovering the desired item rated by one of the peers.

TrustWalker proposes a more balanced approach where it also considers the ratings of similar items by strongly trusted friends. This allows the walker to gather more information from trusted peers and thus offer higher coverage without requiring information from untrusted users. Traditional random walkers only hold data about the item of interest, while this approach also uses a list of similar items to aid in gathering information. The model consists of two main components: the random walk on the trust network and a probabilistic item selection. The item selection component prevents the walker from going too deep into the network by including information on similar items into the recommendation model. This allows TrustWalker to gather sufficient information without adventuring too deeply in the network of untrusted users.

Considering the walker starts from a node u and after a n steps reaches node u_n, in search of information on the target item i. If u_n does not have item i rated, the walker needs to decide whether or not to continue its exploration, or gather knowledge on a similar item. The probabilities of choosing one of the options is denoted as $\phi_{u,i,n}$ for the latter and $1 - \phi_{u,i,n}$ for the former, with Jelasity proposing the following formula for its computation:

$$\phi_{u,i,n} = \max_{j \in RI_u} sim(i,j) \times \frac{1}{1 + e^{-\frac{n}{2}}} \tag{2.17}$$

where RI_u represents the set of items that are similar to the target item and :

$$sim(i,j) = corr(i,j) \times \frac{1}{1 + e^{-\frac{|UC_{i,j}|}{2}}} \tag{2.18}$$

with $corr(i,j)$ being the Pearson correlation of ratings expressed for both items:

$$corr(i,j) = \frac{\sum_{u \in UC_{i,j}} (r_{u,i} - \overline{r_u}) \times (r_{u,j} - \overline{r_u})}{\sqrt{\sum_{u \in UC_{i,j}} (r_{u,i} - \overline{r_u})^2} \times \sqrt{\sum_{u \in UC_{i,j}} (r_{u,j} - \overline{r_u})^2}} \tag{2.19}$$

and $CU_{i,j}$ is the subset of user who have rated both items.

A single walk returns when one of three conditions are met:

1) If it has reached a node that has the target item rated.

2) If at some node u_n it decided to use one of the similar items RI_u as the source of the requested rating instead of the target item i.

3) To prevent walking to infinity, the last condition is limiting the number of steps it can take. In their paper, Jamali et. al. choses the maximum depth to be 6, based on the "six-degrees of separation" principle in [25].

To increase accuracy, the algorithm performs several walks and therefor needs to determine when it has gathered sufficient information in order to make a reliable decision. For this, the variance of the results of all walks needs to be computed:

$$\sigma^2 = \frac{\sum_{i \in [1,n]} (r_i - \overline{r})^2}{n} \tag{2.20}$$

where r_i is the result of the i-th random walk and $\overline{r}$ represents the mean of the results of all the walks with n being the number of executed walks. σ^2 is defined as *the variance in the results of the first i random walks*. The termination condition in [14] is:

$$|\sigma^2_{i+1} - \sigma^2_i| \leq \epsilon. \tag{2.21}$$

TrustWalker [14] also introduces a measure of confidence in its results, as a function of σ. It correlates the variance to the confidence such that the confidence is inversely proportional to the variance.

$$confidence = 1 - \frac{\sigma^2}{max(\sigma^2)}. \tag{2.22}$$

where $max(\sigma^2)$ represents the maximum possible variance for the results.

2.3 Trust Inference

Trust has been the focus of much research since it emerged as a reliable means of improving recommendation accuracy. Its effects on and correlation with the privacy preferences of users have also been studied (e.g. [8], [18]), as inter-user relations are an important factor in the performance of recommender systems. Many definitions for trust have been proposed throughout the literature related to recommender systems. Trust is presented by Mui et. al. in [27] as "a subjective expectation an agent has about another's future behavior based on the history of their encounters". Several other definitions are presented by Zhou et. al. in [41]. Zhou also

presents a more thorough presentation of a wide range of approaches to trust-aware recommender systems.

Throughout this thesis we use the term *trust* to denote the confidence a user has in the recommendations of another. As discussed in [23], trust complements CF recommenders by addressing such problems as the reduced computability of similarity between users, improving accuracy of prediction and limiting the negative effect of *malicious* users. In [39], Yuan et al. describes trust networks as being social networks with user defined trust networks. The authors determine that this type of networks hold the property of small-worldness, which involves having closely clustered users and small average path lengths between any two users. They then use this finding to define a model for recommender systems that takes advantage of the small-worldness of social networks in order to increase both accuracy and item coverage. Several approaches, such as Golbeck [10], Kuter et al. [19], Avesani et al. [2], DuBois et al. [9] and Zarghami et al. [40], also exploit underlying mechanism in a network that allows for explicitly stated trust statements between users. However, not all systems support such features and the ability of users to express confidence in others is limited due to the time and effort required to evaluate other members of the network in order to form an opinion. Therefor, the ability of recommender systems to infer trusts from limited knowledge is still a desired feature.

P. Victor et. al. in [37] addresses the problem of distinguishing between *malicious* users and *unknown* users (i.e. users which we know nothing about). Victor proposes a model that uses distrust to complement trust, such that distrust will be assigned to users that are known to have *malicious* behavior while no trust will simply represent *unknown* users. This approach helps deal more effectively with users that have undesired behavior. The concept of distrust is also used in [35] by N. Verbiest et al. In their work, Verbiest et al. analyses the effect of path length on trust and accuracy. This is particularly interesting to our work since we also observe the effects of using further neighbors on the accuracy and item coverage of our recommender system.

The technique used to infer trust between users is critical to the accuracy of a trust-based CF recommender. Pearson similarity is a popular weight metric, however, as shown in [24], using a more complex weighing measure than just similarity has the potential to offer more accurate results, especially in sparse datasets. Approaches such as those proposed by J. Golbeck et al. [10], [19] take advantage of trust ratings explicitly stated by the users themselves to infer trusts between nearby members of the network through trust propagation. In [40], Fazeli et al. proposes the use of a local trust metric together with a global trust metric computed using the in-degree of a node in relation with the trust on each incoming edge. The global trusts are then used to create a global repository of top trusted users for each item which can then be referenced by other nodes in order to find new neighbors in the network. It is important to note that our trust inference technique does not require any user-defined trust between nodes and it computes trust knowing only user ratings.

The trust metric proposed by O'Donovan and Smyth in [29] is similar to ours

in this respect. O'Donovan uses the known ratings to create an artificial history of predictions for each user. By *predicting* the known ratings of users using all the other users and counting the amount of *correct* predictions that each user makes, O'Donovan and Smyth establish a global trust for each user as the ratio of correct predictions to total predictions of a user(in [29], they also propose item level trust which is similar to user level trust only applied on items; both trust models can be used concurrently to offer better results). Our approach follows the same path of using artificial predictions of known ratings to adjust trust values for users. However, we compute the trusts differently, solely within a neighborhood, resulting in *local trusts* rather than *global trusts*. O'Donovan's method requires the analysis of the whole database when computing a user's trust rating which will greatly impede scalability and will become more computationally demanding as the number of users grows. Our technique can calculate a user's trust towards any subset of users in the network, making it easily implementable in a decentralized paradigm.

Unlike us, O'Donovan computes trusts as an absolute feature of a user, all users in the network have the same trust in a given user. We refer to our trusts *local* because, as we will show in section III, the values computed are relevant only in the context of a neighborhood. Thus, the trust between two users depends on the profiles of the other neighbors as well as the profiles of the two users.

TidalTrust

In her work [10], J. Golbeck introduces the TidalTrust trust inference algorithm. This method addresses the problem of computing trust between users not connected by a direct edge in a trust network. It is important to note that this approach is dependent on a pre-existing trust graph between the users of a social network. TidalTrust is built on the assumption that the further apart two nodes are in the trust network, the higher the chances are that their tastes differ. In her work, Golbeck presents the following two principles are features integrated into TidalTrust:

1) For a fixed trust value, shorter paths have lower error.
2) For a fixed path length, higher trusts have lower error.

According to the two statements above, an intuitive way of integrating path length into trust computation would be to limit the depth of search in order to achieve higher accuracy. This however would limit the number of accessible nodes. Golbeck proposes a compromise in which the allowed path length can vary from computation to computation. By limited the depth to the shortest path length required to connect the two nodes, the algorithm will preserve the advantage of having shorter path lengths while maintaining the capacity to generate the same number of inferences.

To take into account the fact that most relevant information rests with the closest neighbors, TidalTrust only considers connections between two users that have a trust value above a certain threshold. This limit can not however be determined pre-execution as it is impossible to know what the trust values would be along the

possible paths. This threshold needs to be set at a value that allows the existence of a path between the two participating nodes. If the value was set beforehand, there is a chance it is too high and there will be no path determined between the two nodes. Also, if it were too low, it would allow the algorithm to take into account unreliable information coming from untrusted users. This threshold is set during execution in order to ensure the two nodes will be connected.

TidalTrust works by sweeping outwards into the network while polling the nodes along the way about the destination node and afterwards returning the final value to the source. If a neighbor has a direct edge to the destination, the trust value between them is returned, otherwise, it queries its neighbors. This behavior is done recursively, each node keeping track of the depth of the query. This will essentially behave as a Breadth First Search algorithm on the trust graph, having the source node as the root. The *strength* of a path to each neighbor is the minimum of the strength of the path to the node and the node's trust of its neighbor. This will essentially be the minimum trust encountered between two neighboring nodes on the path until that moment. Each node is required to keep track of the path leading up to it that has the highest strength.

Once the destination node is reached, the maximum depth is set to the length of the path between the destination and the source. Since TidalTrust parses the graph in a depth first search manner, this will be the guaranteed shortest path. The other paths continue until reaching the depth limit, essentially detecting all paths of this length to the destination node. Once all the paths reached the limit, the trust threshold is set as the maximum strength of all the paths leading to the destination node. Once this step is complete, each node can complete its computation of the following formula:

$$t_{i,s} = \frac{\sum_{j \in adj(j) | t_{i,j} \geq max} t_{i,j} t_{j,s}}{\sum_{j \in adj(j) | t_{i,j} \geq max} t_{i,j}} \tag{2.23}$$

It is important to realize that, because all the paths depends on the detection of the shortest path to the destination node, this approach is problematic to implement in a decentralized fashion. The complexity required to announce all the paths of the reached minimum would be a definite hurdle in the way of a decentralized implementation of TidalTrust. Our approach, presented in chapter 3, is completely decentralized and also does not require a pre-existing trust network between the users of a social network.

MoleTrust

In [22] P. Massa et al. propose a *local* trust metric for propagating trust through a network of users in order to predict the trust between a *source* user and a *target* user. In the context presented by Massa et al., *local* trust metrics refer to metrics that are computed from the perspective of an active user, rather than a central authority that attaches a trust value on each user and other users consider it absolute. The local trust metrics present the advantage of viewing prediction partners from the point of view of each user, deeming them trustworthy or not for that particular

individual. This will result in better tailoring of predictions for each individual rather than relying on general metric that appeal to the generic user of the system. In this reference frame, our approach presented later, is a *local* metric of trust as it is computed from a specific user's point of view.

The idea behind MoleTrust is that the trust of the *source* towards the *target* is dependent on the trust values of edges connecting the two users. This will obviously result in a *target* user being trusted differently by different *source* users. The algorithm for propagating trust has two stages. In the first stage, all the cycles are removed from the trust graph. It is important to realize that the trust network is directional, so a cycle between three users A, B and C would be of the form A trusts B with n, B trusts C with m and C trusts A with p. If A would trust C and C would not trust A, the three users would not compose a cycle. If the cycles were not be removed, the algorithm would need to be adjusted to accommodate for repeated passes through the same nodes while walking the trust network. This would require iterative adjustments of values and waiting for convergence. Eliminating the cycles would allow us to compute the trust values in one try and is much more efficient. The procedure to achieve this is described below.

The first step orders nodes by the path length between the *source* and themselves, keeping only those that are within N hops of the *source*. The N parameter is called the *trust propagation horizon*. This parameter determines the maximum distance from the *source* that the walker can reach. This limitation is required because, intuitively, the reliability of the propagation deteriorates as the walker advances further into the network. Having changed the network from the perspective of the *source*, we can no proceed to create the desired acyclic graph by removing all edges with the exception of those connecting nodes at distance k to nodes at distance $k + 1$. This is a rather destructive way of eliminating cycles as it removes edges that could not be involved in cycles and could prove informative, however it increases time efficiency and according to the authors, it is an acceptable trade-off.

The second step of the algorithm will walk the trust graph obtained at step 1, starting from the root node (the *source*). The initial trust score of the *source* is 1. The algorithm then computes the trust scores of all the nodes at distance 1, then 2 and so on. The trust scores of users at distance k will only depend on the scores of users at step $k - 1$ (which have already been computed), thus there is no need for multiple passes through the same node, thanks to remodeling the network at the previous step. To predict the trust score of a node, MoleTrust assesses its incoming edges and considers only those coming from nodes with a trust score above a certain threshold. The trust score is computed using the following formula:

$$trust(n) = \frac{\sum_{i \in predecessors} trust(i) \times edge(n, i)}{\sum_{i \in predecessors} trust(i)} \tag{2.24}$$

This algorithm is in some ways similar to TidalTrust [10], since both approaches use a BFS(Breadth First Search) exploration of the trust graph up to a particular point. The main difference between the two is how MoleTrust remodels the social

network to obtain a more favorable abstraction of the trust graph, from a computational standpoint. Also, TidalTrust computes the *propagationhorizon* dynamically during exploration, while MoleTrust uses a predefined limit.

T-Index

In [40], Zarghami et al. introduce a trust metric based on the popular H-Index [13] indicator used in ranking scientific contribution. H-Index characterizes scientific contribution of researchers as the number of papers with the citation number higher than H. Zarghami et al. propose a similar indicator that would characterize the trustworthiness of users of a social network recommender system. The authors also characterize T-Index as $Indegree^2$, noting that it is similar to the basic indegree characteristic of nodes in a network. The main difference between the two is that T-Index considers the weight of the incoming edges and not only their number.

This approach deals with *clusters* as groups of users connected to the same user in the trust graph, referred to as the *centric user*. The T-Index values are limited to a pre-defined interval. Let us presume this interval is between 0 and 10, for simplicity. The trust values of edges are considered to be within the interval $[0, 1]$. The first step of computing the T-Index for the centric user is multiplying all the trust values of incoming edges by the maximum T-Index value, which is in our case 10. The resulting values are then rounded mathematically and sorted in descending order. Let us presume that the obtained values would be a list of the form $[8, 6, 5, 5, 2, 2, 1]$. To compute the T-Index, we parse the list from left to right until the value stored at a position is lower than the value of its index. The T-Index would be the highest index that stores a value greater than itself, meaning that for our example, the result would be 4. Note that the T-Index of the node would remain the same no matter how many values would follow the first 2 in our example list. Therefor, the nodes with higher indegrees could still achieve a lower T-Index if they are weakly trusted by more nodes. The trust computation between two users used by Zarghami et al. is proposed by Lathia et al. in [20] and is given by the formula:

$$T(u_1, u_2) = 1 - \frac{\sum_{i \in [1,n]} r_{u_1,i} - r_{u_2,i}}{r_{max} \times n} \tag{2.25}$$

where n is the number of items in common, $r_{x,i}$ is the rating of user x for item i and r_{max} is the maximum possible rating.

The T-Index values are then used to create a $Top - Trustee$ list for every available item. This would represent a global repository which nodes could access in order to retrieve top-trusted users(in terms of T-Index score) that rated a specific item. This mechanism would allow nodes all over the network to discover trusted users that may too far out in the trust graph for them to be able to discover them otherwise. The repository is maintained as a distributed hash-table(DHT) to conserve scalability.

Each node maintains a list of $top - n$ trusted neighbors. Whenever it rates a new item, it recomputes its trust edges toward its neighbors and refreshes the $top - n$ neighbor list. Afterwards, it consults the T-Index repository to check if the top trustees for the item in question is a fitting neighbors. It computes the trust between itself and the $top - trustees$ and if it is high enough, they are added to the $top - n$ neighbor list and the corresponding edges are added to the trust graph.

O'Donovan and Smyth's Trust Metric

O'Donovan and Smyth [29] propose a machine learning approach for computing trust values in a social network. This section described O'Donovan's approach as presented in [29]. These trust values are not between two users but between the whole network and one user, essentially being how trustworthy are the predictions of a user in the eyes of all other users. In his work, O'Donovan refers to user requesting a prediction as *consumers* and users whose profiles are used to generate those predictions as *producers*.

The main idea behind the metric proposed in [29] is to generate a history of predictions for all users in a network and analyze how accurate these *previous* predictions are. Judging by the number of *acceptable* predictions in which a user was a *producer*, O'Donovan and Smyth can derive how trustworthy that user's recommendations are. To generate the above mentioned history, for each rated item of each user in the social network, we generate a prediction using all the other users as *producers*, *one by one*. Since to begin with there is no trust information available, these predictions are done using the standard Resnick prediction formula:

$$c(i) = \overline{c} + \frac{\sum_{p \in P_i} (p(i) - \overline{p}) sim(c,p)}{\sum_{p \in P_i} |sim(c,p)|} \tag{2.26}$$

where $c(i)$ is the predicted rating for consumer c and item i, $p(i)$ is the rating of producer p for item i, $\overline{x}$ is the average of the ratings in user x's profile and $sim(c,p)$ is the Pearson similarity between users c and p. It is important to remember these predictions will be done using only one producer at the time, and not a set of producers as in the above formula. This is because it is not desired for the predictions to be influenced by other users as it would reduce the relevance of the current user's contribution. In consequence, when computing users u's trust, we will be generating predictions for all the members of the training set using only u as the producer for every prediction.

After generating the above mentioned prediction history, we can compute the trust of a user as the ratio of *correct* predictions it has made to the number of total predictions it was a part of:

$$Trust(p) = \frac{|CorrectSet(p)|}{|LearningSet(p)|} \tag{2.27}$$

where $CorrectSet(p)$ is the set of correct predictions made by p and $LearningSet(p)$ is the total learning set used by p. A *correct* prediction is defined as a prediction

that is within a certain threshold apart from the user's actual rating i.e. :

$$Correct(p, u, i) \Leftrightarrow |p(i) - u(i)| < \varepsilon \tag{2.28}$$

This metric however does not offer us information about how much a user u_1 trusts a user u_2 but only how much user u_2 is trusted collectively by the users that were part of its training set.

This first metric is referred to as *profile-level trust* since it characterizes the trust other users have for the user in question. This is a rather course estimation so O'Donovan and Smyth developed a more detailed version called *item-level trust*. *Item-level* trust is computed very similarly to *profile-level* trust, the only difference is that the trust values are associated with each item separately. Meaning the *item-level* trust for item i and user u would be the percentage of correct predictions made by user u for item i of all the users in the training set. This offers a more detailed use of the profile information of each user.

As an additional advantage, the two metrics can be used to complement each-other. In their work, O'Donovan and Smyth present prediction techniques using filtering and weighing users based on these two metrics. The best result was given by item weighing combined with profile filtering. This approach involves considering for prediction only users that have a profile-level trust above a certain threshold and then using the item-level trust when computing the prediction formula.

In the context of our distributed approach, this trust metric however proves problematic to use as the training sets would be significantly less than in centralized methods. Also, this metric does not offer the trust value towards a user from the perspective of a single user(i.e. the one requesting a prediction), but rather from the collective network, which is likely to be more general than we would like. Another disadvantage would be the profile-level trust filtering that would reduce the already limited neighborhood even more and could lead to significant loss in item coverage.

Chapter 3

Approach

In this section we will present our proposed approach. First, we will describe how we use the T-Man algorithm in the context of a recommender system for social networks, and then we describe the trust inference technique we use to compute the trust between two users based on their known ratings.

3.1 Network Overlay

A network overlay is a network constructed on top of another network, by reorganizing the logical links between nodes in order to make it more suitable for the application logic. In our case, the overlay network will be built on top of the social network and a user's resulting neighbors will not necessarily be his friends in the social network.

3.1.1 Cyclon

Cyclon [16] is a decentralized random peer sampling algorithm developed in 2005 initially as an inexpensive membership management service for unstructured P2P systems [38]. Cyclon requires nodes to maintain a *partial view* of the network(a list of node identifiers and their related age) which would eventually represent random entries of nodes from throughout the network. The role of the age is to prevent entries of users that have left the network to propagate as easily as the ones of the active nodes. This give a high degree of fault tolerance to the protocol. The purpose of periodic gossiping in this algorithm is to ensure that diversity is maintained over time in the partial view and also to keep the *partial view* consistent with the dynamics of the system, in case of nodes leaving or joining the network. Essentially, the *partial view* will represent a set of random nodes from the network and the random peer sampling service can simply select one of the nodes.

The algorithm works by executing rounds consisting of 4 steps. Even though the algorithm is not synchronous, we refer to the repeated execution of the 4 steps as rounds because they are executed by each node at certain intervals of T time

measurement units. The protocol is influenced by 3 parameters: the size of the *partial view* - c, H and S. Below we will discuss the meaning of the H and S parameters. The steps composing each rounds are detailed below:

Step 1:

The first step consists of choosing a node from the current partial view. The partial view is initialized with the nodes current neighbors in the network graph. There are different policies for selecting a peer to gossip with at a certain round. Two simple and effective examples would be to choose a node at random from the view and to choose the entry with the highest age. Note that choosing the node with lowest age would result in view exchanges with peers with which there was an exchange the previous round. This would lead to a rather static evolution of the view, with not many updates and low entry diversity.

Step 2:

The second steps consists of initializing a buffer with an empty entry of the current node, consisting of its identifier and age 0. The buffer is then completed with $c/2 - 1$ randomly selected entries from the view. The entries are selected by shuffling the view and selecting the first $c/2 - 1$ entries and ignoring the oldest H entries. If the view does not contain enough entries to fill the buffer, as it is expected to happen during the initial rounds of the protocol, the oldest H entries are also considered for selection. The buffer is then sent to the peer selected at Step 1 and a reply is awaited.

Step 3:

When a buffer is received, it is appended to the receiving node's view. The entries corresponding to the same node are deleted, with the exception of the one with the lowest age. The view now contains a maximum of one entry per node and is guaranteed to be at least the same size as before the append operation was executed. Further eliminations are done, without decreasing the size of the view below c and beginning with the oldest entries. The role of the H parameter is to help eliminate outdated entries, which have higher chances of having failed. This control the *healing* of the network as the higher H is, the faster entries that are not updated regularly are removed from the views.

Step 4:

At step 2 we mention the view is shuffled and the first $c/2 - 1$ entries are added to the buffer. During step 2, the first S entries in the view are removed. S represents the importance a node grants to incoming entries versus existing entries in its view, prior to the gossip round. In essence, S reflects the number of items that are exchanged between nodes during a round, as all nodes run the protocol with the same parameters.

In this thesis, we are not concerned with the fault-tolerance of the system, instead we focus on the behaviour of the network in terms of recommendation performance.

3.1.2 T-Man

The T-Man algorithm [15] is widely used in P2P systems for obtaining overlay networks for a very diverse range of purposes. T-Man is a gossip algorithm that works by having each node maintain a set of most suitable neighbors that it accumulates over time from gossiping with its peers. The main idea behind T-Man is to reorganize a network(however large) into a stable, desired state in a reduced number of iterations. This is done by using a utility function to determine how useful a node is to another node. Usefulness will generally reflect whether or not a peer would be a desirable neighbor for the node in the final, desired state that the network converges to. The aim of each node is to acquire as its neighbors the nodes that have highest utility. The main parameters to consider for this protocol are the distance function, size of neighborhood (n) and number of entries exchanged at each gossip round(c).

This is done by having neighboring nodes exchange information regarding their current neighbors and retain the most *useful* entries they come across. At the start of each gossip round, a node selects a peer from its neighbors. The selection policy can vary across implementations however usually the more *useful* nodes are favored. The two nodes then select their most *useful* neighbors, replace their peer's entry with their own, if it is necessary, and exchange the resulting subsets of nodes between them. After the exchange takes place, each node replaces its least *useful* neighbors with more *useful* entries received from its peer. *Usefulness* is usually defined by a distance function between two nodes, lower values of the distance function representing higher *usefulness*. The implementation of the distance function has a vital role in what the resulting overlay network will look like. The fact that the algorithm allows for a very large range of distance metrics is a key part of what makes T-Man such a versatile tool for creating network overlays.

To further improve convergence time, T-Man is usually implemented together with the Cyclon random peer sampling algorithm [38]. Using random peer sampling together with T-Man offers the benefit of being able to find useful nodes that are not close to a node in the overlay graph. To combine the two algorithms, each node is required to maintain a set of neighbors and a set of randomly selected nodes generated by Cyclon. After a gossip round occurs and two nodes exchange neighborhood information, each node combines their random set with their set of neighbors and the entries received during the gossip step and sorts the resulting set using the distance metric. It then keeps the "closest" nodes in its neighborhood and discards the rest of the entries.

3.1.3 Distance Metric

Our goal is to fill each user's neighborhood with its most similar peers. An intuitive distance metric for our case would be using the Pearson similarity coefficient. However, Pearson similarity has a negative impact on the number of relevant predictions the system is able to make, since it disregards the number of items in common

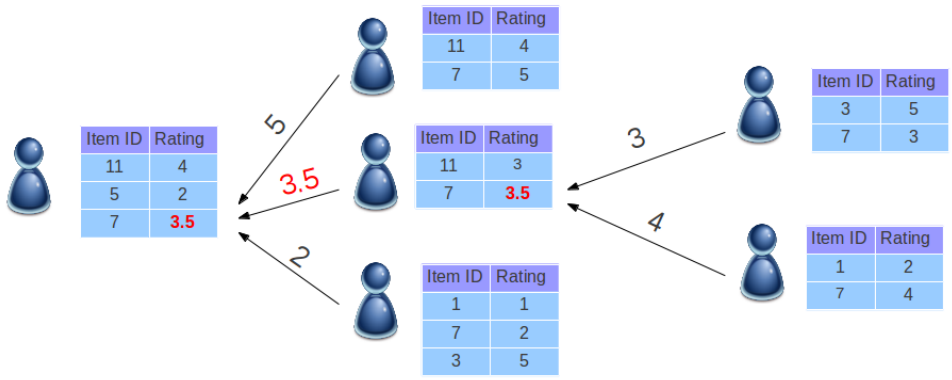

Figure 3.1. Example of recursive prediction in which a user who has not rated an item yet asks for a prediction from its neighbors and passes the result as his own rating. For simplicity, the rating prediction formula is the simple average of the received suggestions.

between two users. Thus, we will use a slightly different version :

$$Similarity(u_1, u_2) = \frac{\sum_i r_{u_1,i} \times r_{u_2,i}}{\sqrt{\sum_i C \times r_{u_1,i}^2} \times \sqrt{\sum_i r_{u_2,i}^2}} \tag{3.1}$$

where $r_{u,i}$ is the rating user u assigned to item i and C is 0.5 if u_2 has not rated item i and 1 if u_2 has item i rated; also, $r_{u_2,i}$ is defaulted to 0 if u_2 hasn't rated item i. This new metric offers a balance between how similar two users are in terms of ratings for common items as well as in terms of the number of items in common. We can now form neighborhoods based on the similarity of ratings in users profiles as well as the number of items they have in common, meaning we are more likely to be able to make predictions for items more relevant to users. It is important to note that in the trust inference step and recommendation step, only the nodes in the neighborhood will be used.

3.1.4 Dealing With Sparsity

In order to deal with potential coverage issues, we propose using *recursive rating prediction* requests. This way, if a neighbor does not have the desired item rated, it can ask its neighbors for a prediction for the item in question, and pass it to the user asking for a prediction as its own(see figure 3.1). This will greatly reduce situations in which none of the neighbors of a user have the item the user is interested in. However, this behavior can not scale very well, the number of involved neighbors potentially increases exponentially. Fortunately, as we will present in the section V, having recursive calls with a depth of maximum 2 is sufficient for even a very sparse database. To reduce complexity for higher range values, recursive calls can be made only when none of the neighbors have the desired item rated.

3.2 Trust Inference

In our experiment, trust is computed after a certain number of T-Man rounds determines the neighborhood of each user. In real scenarios, the T-Man algorithm can be run continuously and the trusts can be computed on *stable* neighborhoods, where a *stable* neighborhood is one that has not changed in a set number of rounds. Our approach for computing trust is inspired by machine learning [21], in the sense that we use a user's known ratings as a training set, based on which we tune the trusts so that we obtain sufficiently accurate predictions for the known ratings. We could view our method as a variation of boosting method in machine learning, where a node's neighbors represent the weak classifiers and we use expectation maximization and iterative proportional fitting to determine the weights that are to be associated with each of them, in order to form a high accuracy classifier.

3.2.1 Expectation Maximization and Iterative Proportional Fitting

In this subsection we will describe the two main algorithms used by the convex linear system solver we chose to use [5]. Describing the two techniques is relevant as such probabilistic methods are used routinely in the context of recommender systems, especially Expectation Maximization.

Iterative Proportional Fitting

Iterative proportional fitting [6] [28], sometimes referred to as "Ranking", is a procedure initially proposed for generating un-biased guesses on cells of data based on samples of data and other information or estimates about the totals of the data cells. Its most common use is estimating values of matrix cells given the row and column totals (in the case of a two-dimensional matrix but can be generalized for other dimensions). It works as a weighting system adjusting the cells in the table repeatedly in order to fit the row and column constraints. The resulting data will 'constitute a joint probability distribution of maximum likelihood estimates' [28] converging when the probabilities are within a pre-defined limit.

For simplicity, we are focusing on IPF applied to a two-dimensional matrices, however we can easily generalize for higher dimensions. The mathematical definition of IPF presented in [28] is as follows:

$$p_{ij(k+1)} = \frac{p_{ij(k)}}{\sum_j p_{ij(k)}} \times Q_i \tag{3.2}$$

$$p_{ij(k+2)} = \frac{p_{ij(k+1)}}{\sum_i p_{ij(k+1)}} \times Q_j \tag{3.3}$$

where $p_{ij(k)}$ is the matrix element at row i, column j and iteration k, and Q_i and Q_j are the desired row and column totals. The two equations are employed repeatedly, at each iteration step, until the following condition is met:

$$\sum_j p_{ij(k)} == Q_i \;\; \&\& \;\; \sum_i p_{ij(k)} == Q_j \tag{3.4}$$

The IPF algorithm's results can also be achieved by specialized versions of more general iterative algorithms, for example EM. The line between the two is somewhat blurred, as IPF is a more specialized approach while EM is a whole class of algorithms. IPF however usually has lower complexity since it only requires simple arithmetical operations, while EM algorithms can require more complex operations, depending from one instance to another.

Expectation Maximization

Expectation Maximization [7] [3] is an iterative method used for computing maximum likelihood estimates of parameters in statistical models from observations that can be viewed as incomplete data. By "incomplete" we refer to the fact that the parameters can not be calculated by solving the corresponding equations. Given a data set X, obtained from a parameterized family, the algorithm's goal is to maximize the maximum likelihood estimate for θ: $P(X|\theta)$, where θ is the vector of parameters. To add the set of unobserved data Z to the formula, we use the following artifice:

$$P(X|\theta) = \sum_{z \in Z} P(X, z|\theta) \times P(z|\theta). \tag{3.5}$$

EM consists of two steps which are applied iteratively. During the *Expectation step*(or *E-step*) the algorithm computes the conditional expectation of the missing data Z with regard to the observed data X and the values of the parameters vector θ_n at iteration n:

$$E_{Z|X,\theta_n}\{lnP(X, z|\theta)\} = \sum_{z \in Z} P(z|X, \theta_n) \times P(X, z|\theta). \tag{3.6}$$

The second step is called the *maximization step*(or *M-step*) and consists of determining the next value of the vector of parameters. This is done by computing θ so that it maximizes the expression obtained in the *Expectation step*. Ultimately, the repeated application of the two steps presented above will lead to an approximation of the solution we are trying to find for θ. For the proof of convergence and an in-depth presentation of the method, we refer you to [7] [3].

3.2.2 Modeling the system

We chose to use the formula described by (3.7) to calculate predictions of an item's rating for a user. The formula is one of the most popular ones for predicting ratings and is also used by Resnick prediction and O'Donovan et al in [29].

$$\frac{\sum_n (r_{n,i} - \overline{r_n}) \times w_n}{\sum_n w_n} = r_i - \overline{r} \tag{3.7}$$

where $r_{n,i}$ is the rating of neighbor n for item i, $\overline{r_n}$ is the average of user n's ratings, w_n is the weight the user assigns to neighbor n(or the trust of the user

for n; we will be using the terms "trust" and "weight" interchangeably), r_i is the user's rating for item i and $\overline{r}$ is the user's average rating. It should be noted that the denominator sum is not of weights in absolute value, meaning we only consider weights to be positive. This makes perfect sense since we are interested in the proportions between them and not their actual values.

Given the fact that we know all the ratings involved, since we are applying it for items a user has already rated, we can interpret (3.7) as an equation with the trusts representing the variables. If we use the formula for every item a user has rated, we can form a linear system of I equations and N variables, where I is the number of rated items the user currently has, and N is the number of neighbors, of the form:

$$\begin{cases} \sum_n (r_{n,0} - \overline{r_n} - r_0 + \overline{r}) \times w_n = 0 \\ \dots \\ \sum_n (r_{n,i} - \overline{r_n} - r_i + \overline{r}) \times w_n = 0 \end{cases} \tag{3.8}$$

This system most likely will not have positive solutions, however we can try to approximate them. For this step, we have chosen to use the algorithm proposed by D. Cartwright in [5], which uses expectation maximization [21] and iterative proportional fitting to gradually converge to an approximation of the solution of systems similar to the one above. However, in order to be able to use this method, we must first overcome the constraint that the system can only have positive coefficients. To ensure that the coefficients remain positive no matter what ratings are involved in the calculation, we will add an extra equation to our system, as follows:

$$\begin{cases} \sum_n (r_{n,0} - \overline{r_n} - r_0 + \overline{r}) \times w_n = 0 \\ \dots \\ \sum_n (r_{n,i} - \overline{r_n} - r_i + \overline{r}) \times w_n = 0 \\ \sum_n w_n = N \times Trust_{mean} \end{cases} \tag{3.9}$$

where $Trust_{mean}$ is the average between what we set as a maximum limit and a minimum limit for trust values. We chose the maximum limit to be 1 and the minimum limit to be 0.1, since we don't want to disregard any ratings as we have already filtered the representative users into the neighborhood. However we observed that using values closer to 0 (such as 0.001) do not have a significant effect on accuracy or on the distribution of trust. It is also worth mentioning that the added equation that limits the sum of the weights to a constant value will not have any influence on the quality of predictions as it does not affect the ratio between the resulting weights. The added equation also gives us the benefit of having trust values of the same order of size throughout the network.

However, adding the equation above is not sufficient for satisfying the constraint required for [5]. We must now add a constant $2 \times R_{max}$ to each coefficient in the initial equations, resulting in the following system:

$$\begin{cases} \sum_n (2 \times R_{max} + r_{n,0} - \overline{r_n} - r_0 + \overline{r}) \times w_n = C \\ \dots \\ \sum_n (2 \times R_{max} + r_{n,i} - \overline{r_n} - r_i + \overline{r}) \times w_n = C \\ \sum_n w_n = N \times Trust_{mean} \end{cases} \tag{3.10}$$

where $C = 2 \times R_{max} \times \sum_n w_n$ which, considering the last equation in the system, will actually be a constant $C = 2 \times R_{max} \times N \times Trust_{mean}$. In the above formulae, R_{max} is the maximum rating available in the profile database. It is now obvious that all the coefficients will become positive.

3.2.3 Approximating the Trusts

As we stated earlier, we will use D. Cartwright's approach to approximate solutions for (3.10). The algorithm assumes there are no negative coefficients in the system which are represented as a matrix of coefficients A. Another assumption is the existence of the non-negative matrix of exponents G, of size $s \times n$, where n is the number of variables and s is the largest number of different exponents that appear for any variable. Also, it assumes the existence of the positive real number d_j, with $j \in [1, s]$ such that for each exponent $\alpha \in S$ and $j \leq s$ the following statement holds true:

$$\sum_{i \in [1,n]} G_{ji} \times \alpha_i = 0 \ or \ d_j. \tag{3.11}$$

The algorithm starts with randomly initializing the solution vector and iteratively converging towards an approximation of the solutions by applying the EM steps described earlier.

```
while(x not stable)
    for all a in exponents S
```

$$\varepsilon_a = \sum_i R_i \frac{A_{ia} x^a}{\sum_b A_{ib} x^b}$$

```
    while(x not stable)
        for j = 1 to s do
            for all variables x
```

$$x_i = x_i \times \frac{\sum_a \varepsilon_a}{\sum_a A_a x^a}^{G_{ji}/d_j}$$

where $A_a = \sum_i A_{ia}$ and R_i is the free term in each of the equations.

In the above algorithm, we can distinguish the two steps of the expectation maximization algorithm. The E-step consists of computing the ε as a measure of how we need to adjust the monomial x^a so that the sum of monomials in an equation i better match the desired result R_i. The M-step consists of applying the IPF algorithm in order to obtain the values for the variables that would best fit

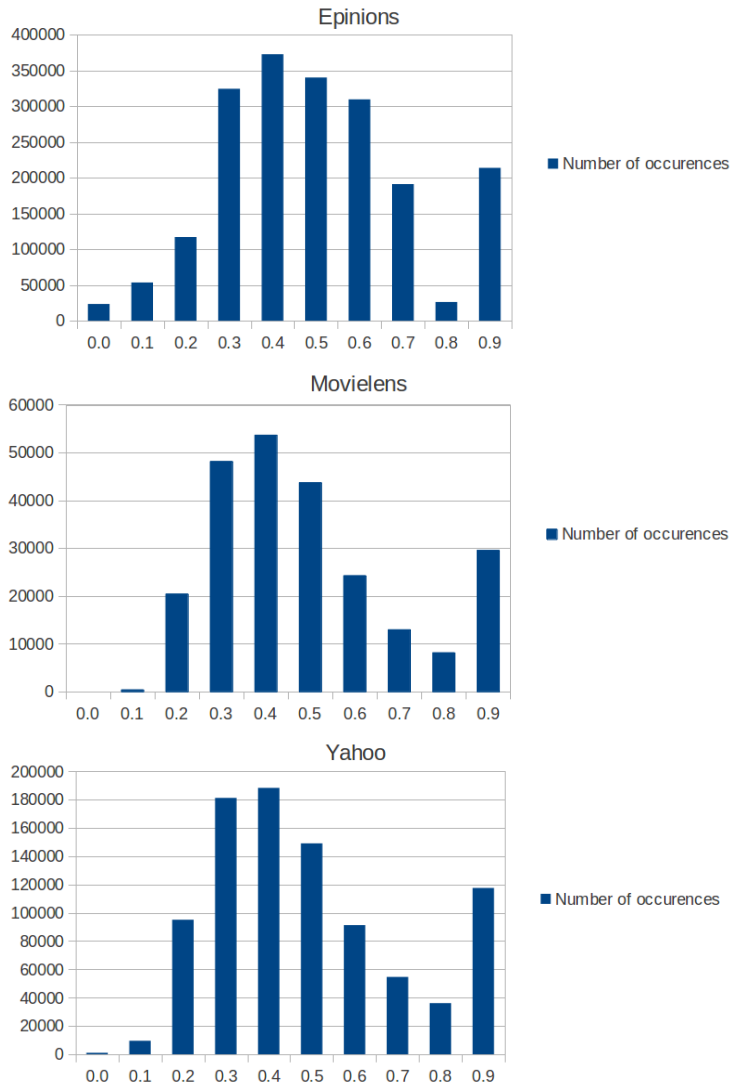

Figure 3.2. Trust distribution obtained for Epinions(top), Movielens(middle) datasets and Yahoo(bottom). Each bracket represents an interval of size 0.1 in the allowed trust range (e.g. Bracket 0.2 represents the interval [0.2, 0.3)). The allowed trust interval for this experiment is [0.1, 1].

the measure computed before. The next iteration would recompute ε taking into account the values obtained by the IPF algorithm in the M-step and attempt to determine new values that would fit the new model.

Our case is a particularly simplified example of the types of systems the algorithm can solve, meaning the algorithm can be reduced to a much simpler form. Since the monomials are simply the variables of our system, all x^a will become x_i. Thus, the second *while* statement would only have 1 iteration, meaning the algorithm can be re-written in the following form:

```
while(!converged)
    for each n in neighbors
```

$$w_n = w_n \times \frac{\sum_{i \in Items} \frac{C}{C_i} \times coef_{i,n}}{\sum_{i \in Items} coef_{i,n}} \tag{3.12}$$

where C_i is the value of the left term of *equation* i of the system with the current values for weights, or $\sum_n (2 \times R_{max} + r_{n,i} - \overline{r_n} - r_i + \overline{r}) \times w_n$, $coef_{i,n}$ is the coefficient of w_n in *equation* i, or $2 \times R_{max} + r_{n,i} - \overline{r_n} - r_i + \overline{r}$, and C is the constant defined in equation (3.10) . Also we must mention that we initialize the weights with random values in the trust interval we chose to use. It is important to state that the trust values are updated simultaneously. The terminating condition is described in more detail in the following section.

We can observe from the algorithm above that, at each iteration, the trusts are multiplied with the weighted sum of the error of the equations weighed by the value of the coefficient of the variable in the respective equation. We can thus express the importance of each equation by multiplying all their terms with a constant k(the lower k, the lower the equation's importance). Since the last equation in our system is just an artifice to satisfy the algorithm's constraints, we can multiply all its terms with a low constant(we chose $k = 0.1$):

$$\begin{cases} \sum_n (2 \times R_{max} + r_{n,0} - \overline{r_n} - r_0 + \overline{r}) \times w_n = C \\ \dots \\ \sum_n (2 \times R_{max} + r_{n,i} - \overline{r_n} - r_i + \overline{r}) \times w_n = C \\ \sum_n w_n \times k = N \times Trust_{mean} \times k \end{cases} \tag{3.13}$$

Furthermore, we can, in theory, use the same principle to highlight the importance of more *controversial* [36] items over items voted similarly across a vast majority of users. However, we will leave this to future work and will not tackle this problem in this thesis.

3.2.4 Convergence

In Cartwright's algorithm [5], the convergence of the algorithm is given by the condition that the Kullback-Leibner divergence between the solutions vector before and after a step should be below a certain threshold. However, we want to achieve a balanced trust distribution and we prefer to prevent some users from receiving an imbalanced high trust value. For example, if one of the neighbors had exactly the same ratings as the user, to solve the system, an obvious solution is to give all the other neighbors 0 trust and use the value required to satisfy the last equation of (3.10) , i.e. $N \times Trust_{mean}$. Such situations are often encountered in sparse

datasets, when the user has a small number of rated items but a significantly higher number of neighbors, resulting in far more variables than equations.

To deal with this setback, we will limit the values for trust to a certain interval. We used a minimum value of 0.1 and a maximum value of 1. If at a step, a trust value exceeds either, it is rounded to the appropriate end of the interval after each iteration. Even so, trust distribution tends to clutter around the edges of the interval, so we interrupt the iterations when the number of values that have reached either end of the interval exceeds 10% of the size of the neighborhood. This value can be changed if needed. However, we observed that it gave a good trust distribution across the allowed interval as shown in figure 3.2. Also, in order to prevent values reaching the upper and lower bounds prematurely, we initialize the trusts within the interval

$$\begin{aligned}&[Trust_{min} + 0.25 \times (Trust_{max} - Trust_{min}),\\ &Trust_{min} + 0.75 \times (Trust_{max} - Trust_{min})].\end{aligned} \tag{3.14}$$

It is worth noting that trusts are relevant only in the context of the neighborhood they have been calculated in. The trust value between two neighbors can be compared only if they have been computed in the same neighborhood. Thus, each user can choose any positive interval for trust values without influencing the system's performance.

Chapter 4

Experiment Results

In this section we will discuss the performance of our system in terms of accuracy of prediction and coverage. We will be measuring accuracy in terms of MAE (Mean Average Error) and coverage as the number of users for which the system can generate a prediction for the requested item i.e. the item hidden from the profile of the user.

We have structured our findings in three sections. The first section is dedicated to the experimental setup and a brief description of the data sets used. The second will present the performance of our trust metric and compare the obtained MAE by different approaches against the chosen baselines. The third will present the influence of the network overlay algorithm and the results of our attempts to maximize coverage in our decentralized system.

4.1 Experimental Setup

To test the performance of our system and trust inference method, we used three datasets. The first dataset is obtained from the Movielens website, which represents databases with an abundance of ratings. The second is from Epinions.com, which reflects the sparse databases one would expect to encounter in a real scenario. Movielens.com is a website that allows its users to rate various movies they have watched The Movielens[1] dataset consists of 6040 users and 1 million ratings, with a minimum of 20 ratings per user. This set also presents a balanced distribution of ratings across the allowed scale (1 to 5) and has an average of 165 ratings per user. The Epinions [2] dataset consists of 49,290 users, 139,738 items and 664,824 ratings, some of the users(9127 of them) having empty profiles. Unlike the Movielens dataset, this sample has a more disproportionate distribution of ratings across the scale 1 to 5, a large majority of the ratings being either 4 or 5. Also, in the case of the Epinions dataset, the average number of ratings per user is only 13. The third dataset, obtained from the Yahoo! Webscope [1] website, represents relatively

[1]Movielens dataset available at http://www.grouplens.org/node/73

[2]Epinions dataset available at http://www.trustlet.org/wiki/Epinions_datasets

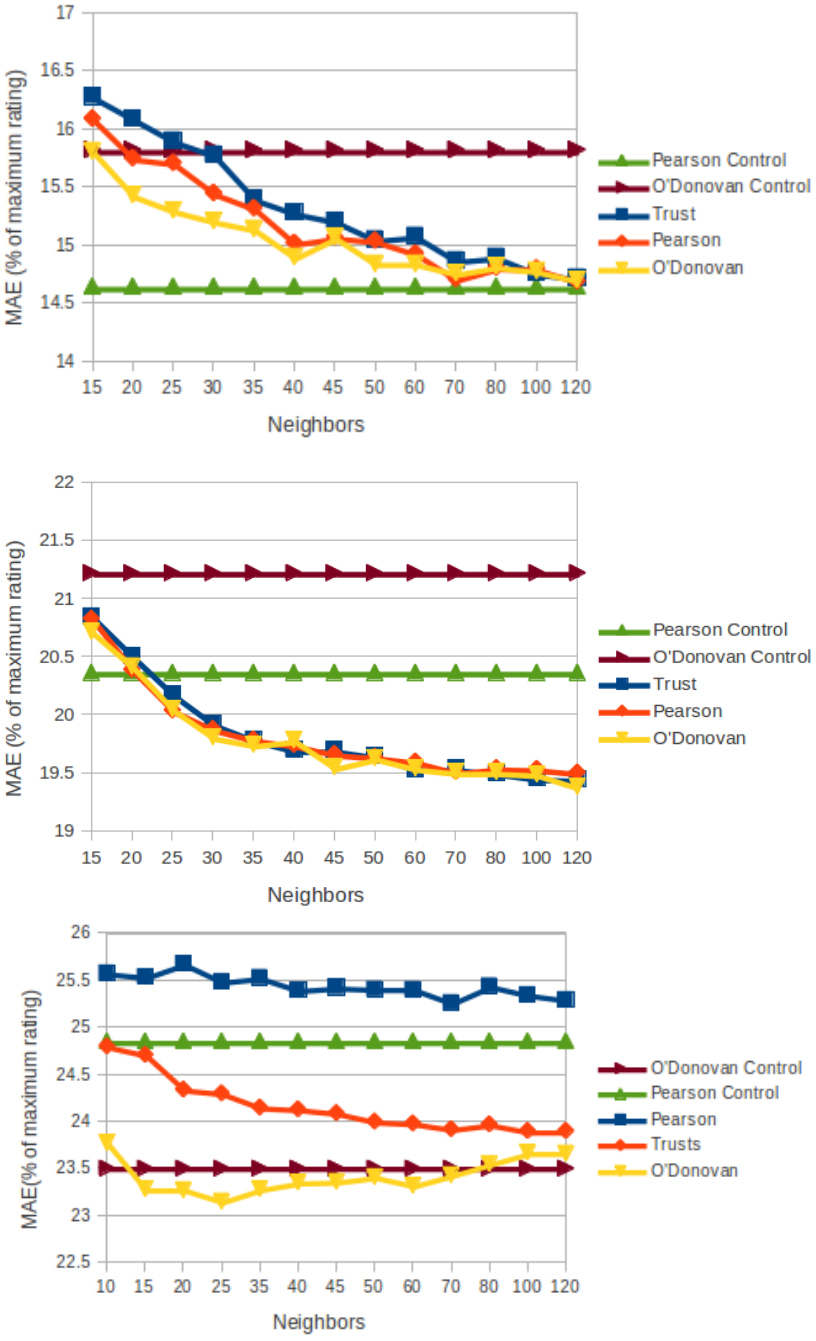

Figure 4.1. The evolution of MAE over neighborhood size for Movielens(top), Yahoo! Webscope(middle) and Epinions(bottom) datasets. We use our decentralized approach in combination with Pearson similarity, O'Donovan's trust metric and our trust metric and compare with the baselines set by O'Donovan's trust metric and Pearson similarity over the whole dataset.

less sparse databases. It consists of 15400 users totaling 300,000 ratings with a minimum number of 10 ratings for any user.

To evaluate our methods, we will use the leave-one-out methodology in which we hide one rating for every user and then run the T-Man algorithm and try to predict the hidden ratings. We initialize the neighbors with a set of randomly picked nodes. In a real scenario, this initialization could be replaced by choosing to use the friends of a user in a social network for example. As a measure of item coverage, we evaluate the number of users for which the system can generate a prediction for the hidden rating i.e. at least one of the neighbors can provide a rating for the item in question. We evaluate the performance on the above mentioned datasets using 3 trust metrics: Pearson correlation, the metric proposed by O'Donovan in [29] and our approach, presented earlier.

4.2 Trust Metric and MAE

In this section we will discuss the results of our trust metric compared to previous approaches and evaluate the evolution of accuracy and coverage function of the neighborhood size. The chosen baselines for our system are regular prediction taking all the users into account with trust represented by the Pearson similarity and by the trust metric proposed by O'Donovan in [29]. These baselines are referred to as *Pearson control* and *O'Donovan control* in the figures in this section. We compare the performance of these two methods against our system using the overlay algorithm in combination with the trust metrics used in the baselines as well as with our own trust metric, described in the section III. To calculate MAE we used the following formula:

$$MAE = \frac{\sum_N |r_{n,i} - r^p_{n,i}|}{|N|} \tag{4.1}$$

where $r_{n,i}$ is the rating of item i for user n, $r^p_{n,i}$ is the predicted rating of item i for user n and N is the subset of users which have successful prediction requests for item i i.e. the users which can make a prediction for item i. Also, i of node n is the item we hid from node n in the leave-one-out step.

Figure 4.1 shows the performance of the different trust metrics over the size of the neighborhood in the case of the Movielens dataset. We can observe that all the metrics increase accuracy with the size of the neighborhood and all show very similar performance, with our trust metric performing slightly worse than O'Donovan and Pearson similarity. The chosen baselines are the variations of the Resnick prediction technique, presented in [29], (using Pearson correlation as similarity and O'Donovan trusts) over the whole set of users.

We notice that our decentralized approach yields very close results to the baseline(0.739 compared to the baseline of 0.732, with the decentralized approach using O'Donovan trusts and Pearson similarity producing MAEs between the two values), while presenting the added advantage of scalability. On the Epinions dataset how-

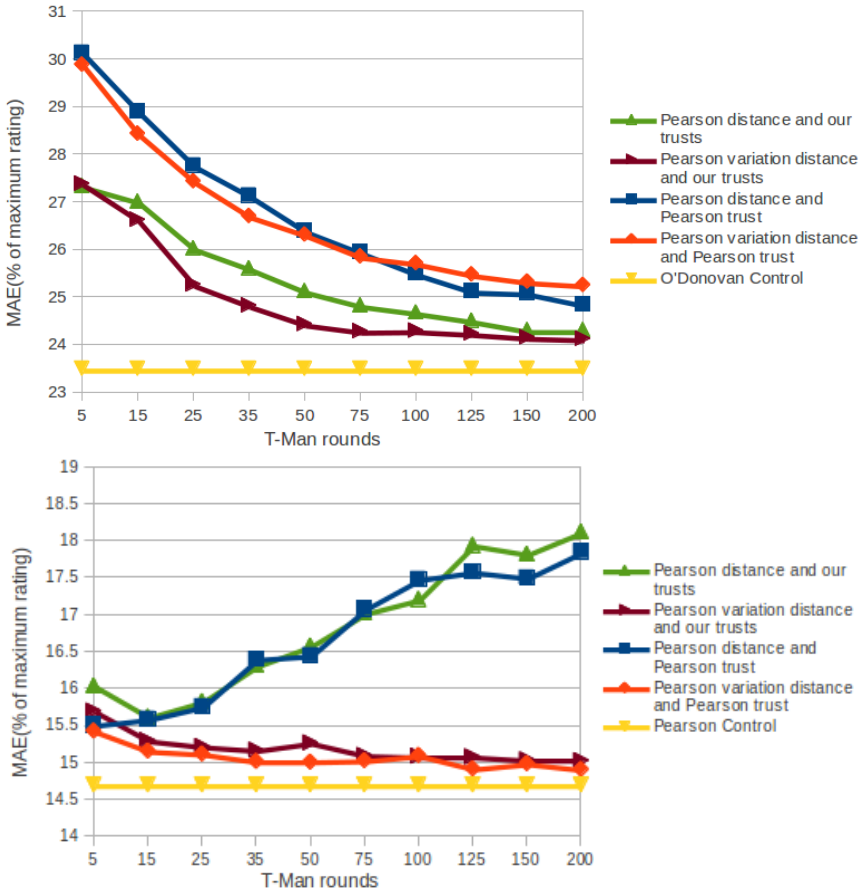

Figure 4.2. Evolution of MAE over T-Man rounds for different trust and distance metrics on Epinions(top) and Movielens(Bottom) datasets.

ever, our trust metric performs better than Pearson similarity when used both in a centralized fashion or decentralized. Using the decentralized approach in combination with O'Donovan trusts yields the best results, O'Donovan baseline being second and our trust metric a close third and performing better as the number of neighbors increases. We must note that O'Donovan trusts were computed using the whole database as the training set(best performance scenario). It would be significantly harder to implement the metric in a distributed fashion in a way that would preserve these results in terms of MAE and coverage. Our trust metric can be computed with a very limited training set and can therefor be part of a fully distributed approach. In these experiments, the distance metric chosen for forming the overlay is the variation of Pearson similarity presented in formula (3.1). In the case of the Yahoo! dataset, our decentralized approach shows little influence of the trust metric used inside the neighborhoods, all three metrics showing very close results. However, the decentralized method yields better accuracy than both

centralized baselines.

In figure 4.2 we can clearly notice how accuracy increases as the network overlay converges, especially when using our proposed distance metric. In this experiment we compare the evolution of accuracy with the number of T-Man rounds and also compare the two earlier mentioned distance metrics used in the T-Man implementation: regular Pearson similarity and its variation presented at formula (3.1). In the case of the Epinions dataset, from the point of view of prediction accuracy, both distance metrics give similar results, with the proposed variation yielding slightly better results when combined with our proposed trust. In the case of using Pearson similarity as trust, the variation only gives lower errors in the earlier rounds while using the default Pearson similarity as a distance metric offers slightly better accuracy near the convergence of the network. On the Movielens dataset however, using regular Pearson similarity hurts accuracy significantly regardless of the trust metric we use. This is because nodes will choose neighbors that have rated only a few items in common, but with very similar values. Therefore, if a two nodes have rated only one item in common but, coincidently, with the same rating, they will be considered very similar. Our distance function also takes into account the number of items in common and therefore reduces such occurrences.

4.3 Network Overlay and Coverage

We will refer to a request from a node to a neighbor for the rating of an item as a *prediction request*. A *prediction request* is successful if the neighbor can provide a rating for the desired item (either his own rating or a prediction that itself obtains from its neighbors). In section III we discussed ways in which we can overcome sparsity and we proposed two methods which can be used concurrently. The first method is using a slight variation of the Pearson similarity as the distance metric in the implementation of the T-Man overlay management algorithm. The specified function is presented in equation (3.1). The second is using recursive *prediction requests* which will allow neighbors that do not have a requested item rated to subsequently ask its neighbors for a prediction of the rating for the item in question and pass the prediction as its own rating to the initiating node. To avoid such requests going on forever, we limit such calls by a time-to-live(TTL), which we refer to as *search range*.

Figure 4.3 presents the results of varying the *search range* for the Epinions dataset. It is worth noting that in terms of coverage, our proposed decentralized approach yields very similar results for Pearson similarity and our proposed trust metric, analyzed in the previous subsection. Also, we only present these experiments for the Epinions dataset because the coverage in the case of Movielens is very close to 100% and our methods for dealing with sparsity do not show any significant improvement of coverage. In the top image of figure 4.3, we can clearly observe that using regular prediction techniques, the coverage is very poor for sparse datasets like the one analyzed in this experiment. For 120 neighbors, less than 25% of the nodes

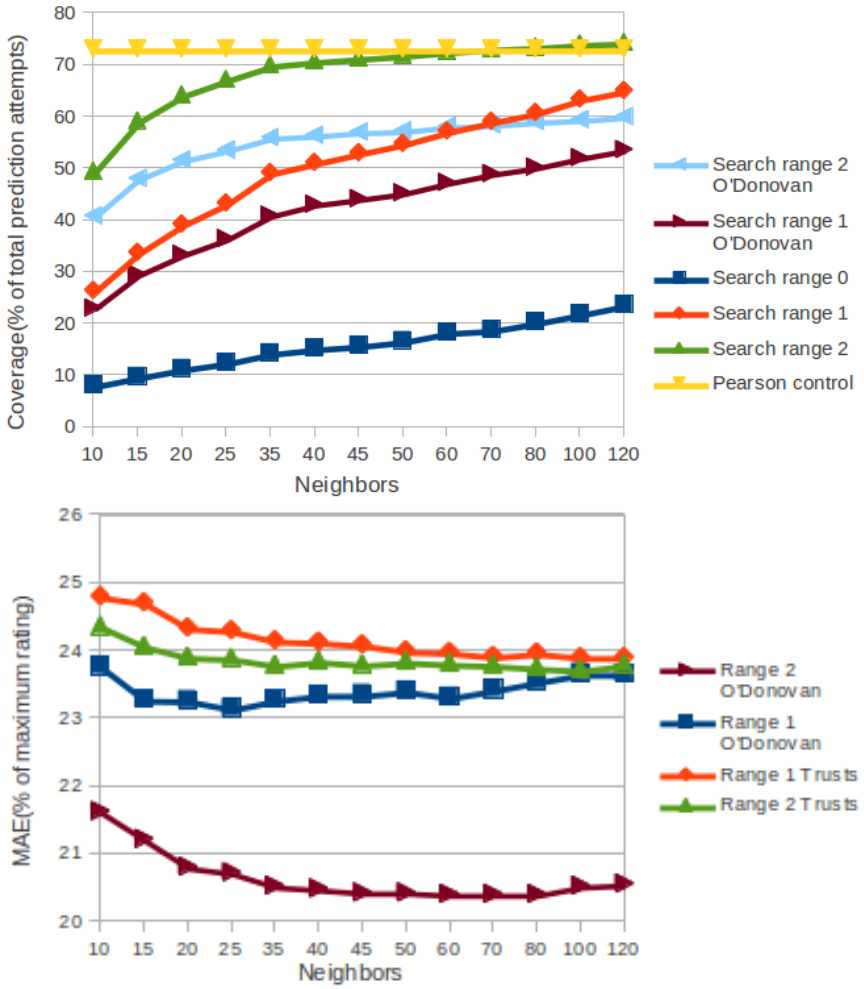

Figure 4.3. Influence of search range on item coverage(top) and MAE(bottom) for Epinions dataset.

manage to obtain a prediction from their direct neighbors, meaning that in 75% of the cases, none of the 120 neighbors had the item in question rated. Since the Epinions dataset is a good representation of a real-life scenario, such low coverage is unacceptable.

We notice that increasing the range to 1 (i.e. allowing immediate neighbors to ask their neighbors for a prediction for that item) greatly increases coverage. In this case coverage in the case of 120 neighbors approaches the baseline set by using Resnick prediction with Pearson similarity over the whole dataset (73% coverage). The baseline is only 73% because 9127 of the 49290 in the network have no items rated meaning we can not verify prediction results against their real rating and we consider these requests as unsuccessful. On top of that, we have hidden 1 item from each user, meaning that sparsity is even worse than in the initial dataset. Despite

being computed in a centralized fashion, O'Donovan trust offers a significant loss of 10% item coverage across the whole range of neighbors(15% for search range 2). If we increase the error margin ε in equation (2.28) to increase the coverage to match that of our metric, the MAE in the case of Epinions would increase to levels comparable to that of Pearson similarity.

Once we increase the search range to 2, we notice coverage is very close to the baseline starting from a neighborhood size of only 25. Even so, using a range of 2 could potentially involve 25^3 nodes in a single request, for the worse case scenario, which might become computationally intensive. However, it is still less demanding and more scalable than having to consult the ratings of every user in the network. Given the sparsity of the dataset used for this experiment, we believe that using a search range higher than this is unnecessary for most real-life cases.

In the plot on the bottom of figure 4.3, we notice the effect of increasing the search range on prediction accuracy. We notice that both in the case of our trusts and O'Donovan trusts, using a range of 2 yields better results than using a range of 1. This is to be expected since there will be significantly more ratings available for the prediction phase. The advantages of using a search range of 2 are particularly obvious in the case of the O'Donovan metric where the accuracy is increased dramatically, significantly surpassing the centralized approach using the same trust metric(1.02 MAE compared to the centralized approach of 1.17). From this experiment we can infer that choosing a range for requests will represent a compromise between accuracy and complexity.

Figure 4.4 depicts how coverage is affected by the convergence of the network and by the distance metric used by the T-Man algorithm. We can observe that as neighbors become more similar to a user, coverage increases significantly. The experiment was run on the Epinions, Movielens and Yahoo! datasets, for a neighborhood size of 60 and using a search range of 1. We notice how coverage is increased when using the variation of the Pearson similarity mentioned in equation (3.1). This makes perfect sense since taking the number of items in common between two users into account when calculating similarity increases the probability that highly similar users will also be interested in similar items. In the case of the Yahoo! dataset, the improvement is not as significant since it is less sparse. However, we observe that using regular Pearson similarity has a negative impact on item coverage as the network converges. The negative impact of this function is even more obvious on the more populated Movielens dataset. This is caused by nodes choosing items with very few items in common but with a high degree of similarity between the shared ratings.

Taking into account the results presented in figure 4.2, we notice that our proposed variation offers better coverage and better accuracy, as in the case of using our proposed trust metric for rating prediction. Thus, we can assume that our metric is overall a more desirable distance metric to be used in the implementation of the T-Man algorithm.

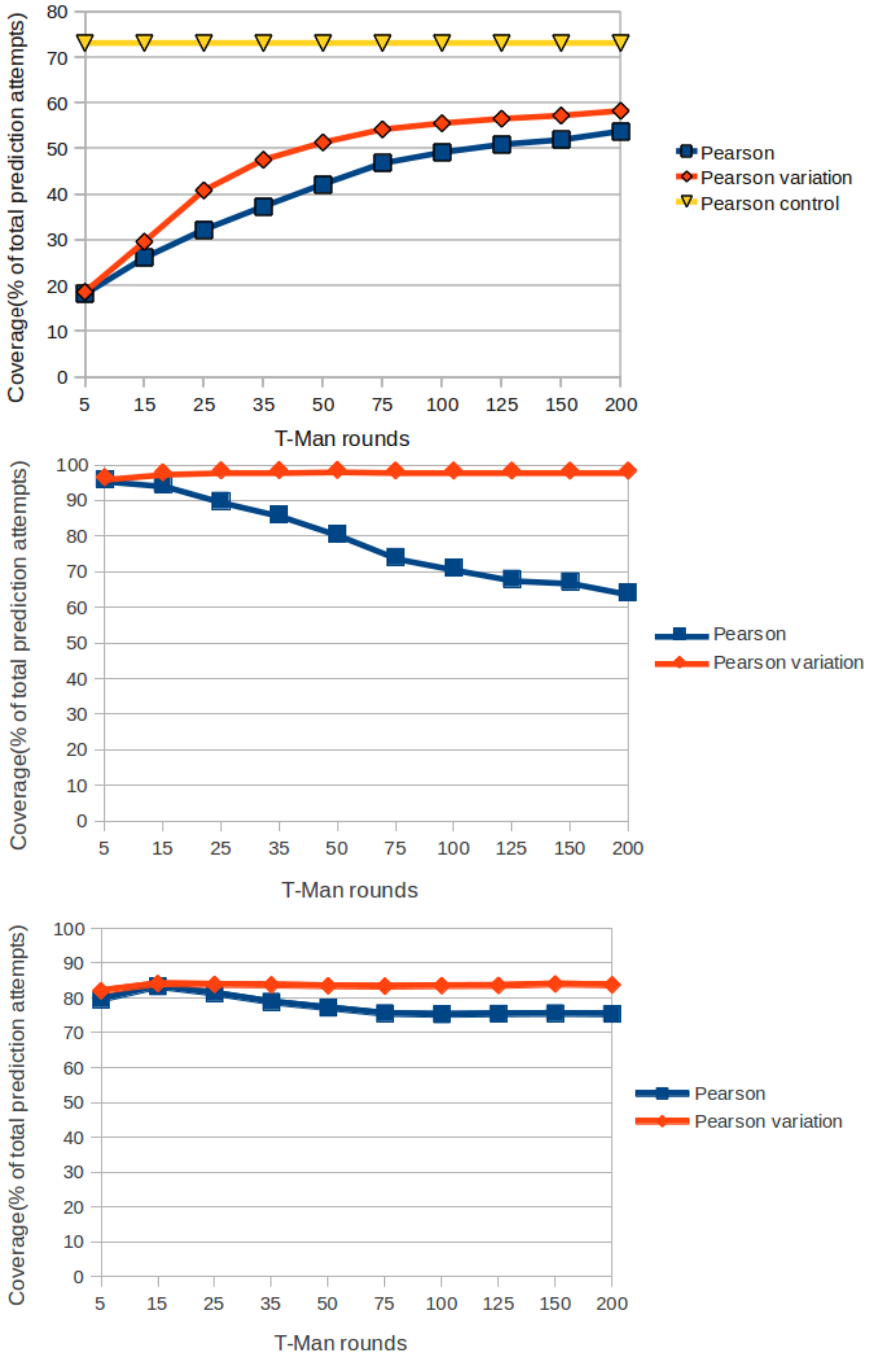

Figure 4.4. Influence of distance metric on coverage as network converges for Epinions(top), Movielens(middle) and Yahoo! Webscope(bottom) datasets.

Chapter 5

Conclusion and Future Work

In this work, we have used techniques inspired from P2P applications to create a scalable recommender system model. We have used the T-Man algorithm to cluster similar users together using a variation of the Pearson similarity as a distance metric. We have shown the improvements in item coverage and accuracy the proposed variation achieves and implemented a new trust inference method to obtain directional trust values between a user and its neighbors while only knowing their ratings. Our inference method can compute trusts between a user and any given subset of neighbors by having them predict the user's known ratings and modifying trusts values until predictions are close to the known ratings. We showed that our decentralized approach achieves better accuracy than two popular centralized models while maintaining comparable item coverage. Also, our trust inference method in the context of our decentralized approach performs better than using Pearson similarity and is comparable to the popular trust metric proposed by O'Donovan and Smyth in [29], while being, easily applicable in a decentralized model, in contrast to the latter metric. We also showed our trust metric allows for significantly higher item coverage than O'Donovan and Smyth's trusts, even though their metric is computed centrally and ours only requires a limited neighborhood.

Future work will include observing the influence of different privacy settings of users on trust values and implementing a policy for punishing users that share too few items through lowering their trust. Also, it is worth exploring the potential of other distance metrics for T-Man and/or prediction formulae and the effectiveness of implementing our trust metric in a centralized fashion. We are also interested in pursuing gradient-based boosting algorithms as a possible replacement for our trust inference method.

Bibliography

[1] Yahoo! Webscope dataset ydata-ymusic-user-artist-ratings-v1_0. http://research.yahoo.com/Academic_Relations.

[2] P. Avesani, P. Massa, and R. Tiella. A trust-enhanced recommender system application: Moleskiing. In *In SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1589–1593. ACM Press, 2004.

[3] S. Borman. The expectation maximization algorithm: A short tutorial. unpublished paper available at http://www.seanborman.com/publications. Technical report, 2004.

[4] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In G. F. Cooper and S. Moral, editors, *UAI*, pages 43–52. Morgan Kaufmann, 1998.

[5] D. Cartwright. An iterative method converging to a positive solution of certain systems of polynomial equations. *Journal of algebric statistics Vol. 2, No. 1*, pages 1–13, 2011.

[6] J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. In *The Annals of Mathematical Statistics*, volume 43, pages 1470–1480, 1972.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[8] N. Dokoohaki, C. Kaleli, H. Polat, and M. Matskin. Achieving optimal privacy in trust-aware collaborative filtering recommender systems. In *The Second International Conference on Social Informatics (SocInfo'10)*. Springer, October 2010.

[9] T. DuBois, J. Golbeck, and A. Srinivasan. Predicting trust and distrust in social networks. In *SocialCom/PASSAT*, pages 418–424. IEEE, 2011.

[10] J. Golbeck. Combining provenance with trust in social networks for semantic web content filtering. In L. Moreau and I. T. Foster, editors, *International*

Provenance and Annotation Workshop, volume 4145 of *Lecture Notes in Computer Science*, pages 101–108. Springer, 2006.

[11] M. Gori and A. Pucci. Itemrank: A random-walk based scoring algorithm for recommender engines. In M. M. Veloso, editor, *IJCAI*, pages 2766–2771, 2007.

[12] P. Han, B. Xie, F. Yang, and R. Shen. A scalable p2p recommender system based on distributed collaborative filtering. *Expert Syst. Appl.*, 27(2):203–210, 2004.

[13] J. E. Hirsch. An index to quantify an individual's scientific research output that takes into account the effect of multiple coauthorship. *Scientometrics*, 85(3):741–754, Dec. 2010.

[14] M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. Zaki, editors, *KDD*, pages 397–406. ACM, 2009.

[15] M. Jelasity and O. Babaoglu. T-man: Gossip-based overlay topology management. In *The Fourth International Workshop on Engineering Self-Organizing Applications (ESOA'06)*, Hakodate, Japan, May 2006.

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

[17] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Extrapolation methods for accelerating pagerank computations. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 261–270, New York, NY, USA, 2003. ACM.

[18] J. A. Konstan and J. Riedl. Recommender systems: from algorithms to user experience. *User Model. User-Adapt. Interact.*, 22(1-2):101–123, 2012.

[19] U. Kuter and J. Golbeck. Sunny: A new algorithm for trust inference in social networks using probabilistic confidence models. In *AAAI*, pages 1377–1382. AAAI Press, 2007.

[20] N. Lathia, S. Hailes, and L. Capra. Trust-based collaborative filtering. *Computer*, 263:299–300, 2008.

[21] B. Marlin. Collaborative Filtering: A Machine Learning Perspective. Master's thesis, University of Toronto, 2004.

[22] P. Massa and P. Avesani. Trust metrics on controversial users: balancing between tyranny of the majority and echo chambers. *International Journal on Semantic Web and Information Systems*.

[23] P. Massa and P. Avesani. Trust-aware collaborative filtering for recommender systems. In *In Proc. of Federated Int. Conference On The Move to Meaningful Internet: CoopIS, DOA, ODBASE*, pages 492–508, 2004.

[24] P. Massa and P. Avesani. Trust-aware recommender systems. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24, New York, NY, USA, 2007. ACM.

[25] S. Milgram. The small world problem. *Psychology Today*, 61:60–67, 1967.

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

[27] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation. In *35th Hawaii International Conference on System Science (HICSS)*, 2002.

[28] P. Norman and L. L. Jt. Putting iterative proportional fitting on the researchers desk, 1999.

[29] J. O Donovan and B. Smyth. Trust in recommender systems. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174. ACM Press, 2005.

[30] R. Ormandi, I. Hegedüs, and M. Jelasity. Overlay management for fully distributed user-based collaborative filtering. In P. D'Ambra, M. R. Guarracino, and D. Talia, editors, *Euro-Par (1)*, volume 6271 of *Lecture Notes in Computer Science*, pages 446–457. Springer, 2010.

[31] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[32] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[33] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

[34] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large–scale e–commerce: Scalable neighborhood formation using clustering. *In Proceedings of the Fifth International Conference on Computer and Information Technology (ICCIT'02)*, December 27–28 2002.

[35] N. Verbiest, C. Cornelis, P. Victor, and E. Herrera-Viedma. Trust and distrust aggregation enhanced with path length incorporation. *Fuzzy Sets and Systems*, (0):–, 2012.

[36] P. Victor, C. Cornelis, M. D. Cock, and A. Teredesai. A comparative analysis of trust-enhanced recommenders for controversial items. In *ICWSM*, 2009.

[37] P. Victor, C. Cornelis, M. De Cock, and P. Pinheiro da Silva. Gradual trust and distrust in recommender systems. *Fuzzy Sets Syst.*, 160(10):1367–1382, May 2009.

[38] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2):197–217, 2005.

[39] W. Yuan, D. Guan, Y.-K. Lee, S. Lee, and S. J. Hur. Improved trust-aware recommender system using small-worldness of trust networks. *Knowledge-Based Systems*, 23(3):232 – 238, 2010.

[40] A. Zarghami, S. Fazeli, N. Dokoohaki, and M. Matskin. Social trust-aware recommendation system: A t-index approach. In *Web Intelligence/IAT Workshops*, pages 85–90. IEEE, 2009.

[41] X. Zhou, Y. Xu, Y. Li, A. Josang, and C. Cox. The state-of-the-art in personalized recommender systems for social networking. *Artificial Intelligence Review*, 37(2):119–132, May 2011.

[42] C. N. Ziegler. *Towards Decentralized Recommender Systems.* PhD thesis, Albert-Ludwigs-Universität Freiburg, Germany, 2005.