Multimed Tools Appl (2008) 36:89–113
DOI 10.1007/s11042-006-0075-6

Personalization on a peer-to-peer television system

Jun Wang · Johan Pouwelse · Jenneke Fokker · Arjen P. de Vries · Marcel J. T. Reinders

Published online: 13 January 2007

Abstract We introduce personalization on *Tribler*, a peer-to-peer (P2P) television system. Personalization allows users to browse programs much more efficiently according to their taste. It also enables to build social networks that can improve the performance of current P2P systems considerably, by increasing content availability, trust and the realization of proper incentives to exchange content. This paper presents a novel scheme, called *BuddyCast*, that builds such a social network for a user by exchanging user interest profiles using exploitation and exploration principles. Additionally, we show how the interest of a user in TV programs can be predicted from the zapping behavior by the introduced user-item relevance models, thereby avoiding the explicit rating of TV programs. Further, we present how the social network of a user can be used to realize a truly distributed recommendation of TV programs. Finally, we demonstrate a novel user interface for the personalized peer-to-peer television system that encompasses a personalized tag-based navigation to browse the available distributed content. The user interface also visualizes the social network of a user, thereby increasing community feeling which increases trust amongst users and within available content and creates incentives of to exchange content within the community.

Keywords *Tribler* · *BuddyCast* · Peer-to-peer (P2P) television system · Personalization · Collaborative filtering · Recommender system

J. Wang (✉) · J. Pouwelse · M. J. T. Reinders
Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, Delft, The Netherlands
e-mail: jun.wang@tudelft.nl

J. Fokker
Faculty of Industrial Design Engineering, Delft University of Technology, Delft, The Netherlands
e-mail: j.e.fokker@tudelft.nl

A. P. de Vries
CWI, Amsterdam, The Netherlands
e-mail: arjen@acm.org

1 Introduction

Television signals have been broadcast around the world for many decades. More flexibility was introduced with the arrival of the VCR. PVR (personal video recorder) devices such as the TiVo further enhanced the television experience. A PVR enables people to watch television programs they like without the restrictions of broadcast schedules. However, a PVR has limited recording capacity and can only record programs that are available on the local cable system or satellite receiver.

This paper presents a prototype system that goes beyond the existing VCR, PVR, and VoD (Video on Demand) solutions. We believe that amongst others broadband, P2P, and recommendation technology will drastically change the television broadcasting as it exists today. Our operational prototype system called *Txribler* [20] gives people access to all television stations in the world. By exploiting P2P technology, we have created a distribution system for live television as well as sharing of programs recorded days or months ago.

The *Tribler* system is illustrated in Fig. 1. The basic idea is that each user will have a small low-cost set-top box attached to his/her TV to record the local programs from the local tuner. This content is stored on a hard disk and shared with other users (friends) through the *Tribler* P2P software. Each user is then both a program consumer as well as a program provider. *Tribler* implicitly learns the interests of users in TV programs by analyzing their zapping behavior. The system automatically recommends, records, or even downloads programs based on the learned user interest. Connecting millions of set-top boxes in a P2P network will unbolt a wealth of programs, television channels and their archives to people. We believe this will tremendously change the way people watch TV.

The architecture of the *Tribler* system is shown in Fig. 2 and a detailed description can be found in [20]. The key idea behind the *Tribler* system is that it exploits the prime social phenomenon "kinship fosters cooperation" [20]. In other words, similar taste for content can form a foundation for an online community with altruistic behavior. This is partly realized by building social groups of users that have similar taste captured in user interest profiles.

The user interest profiles within the social groups can also facilitate the prioritization of content for a user by exploiting recommendation technology. With this information, the

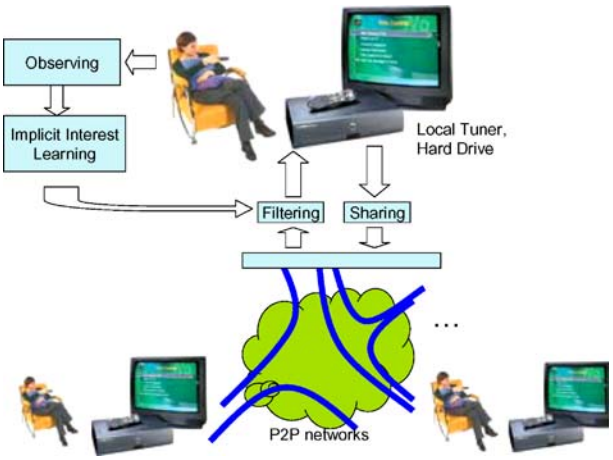

Fig. 1 An illustration of *Tribler*, a personalized P2P television system

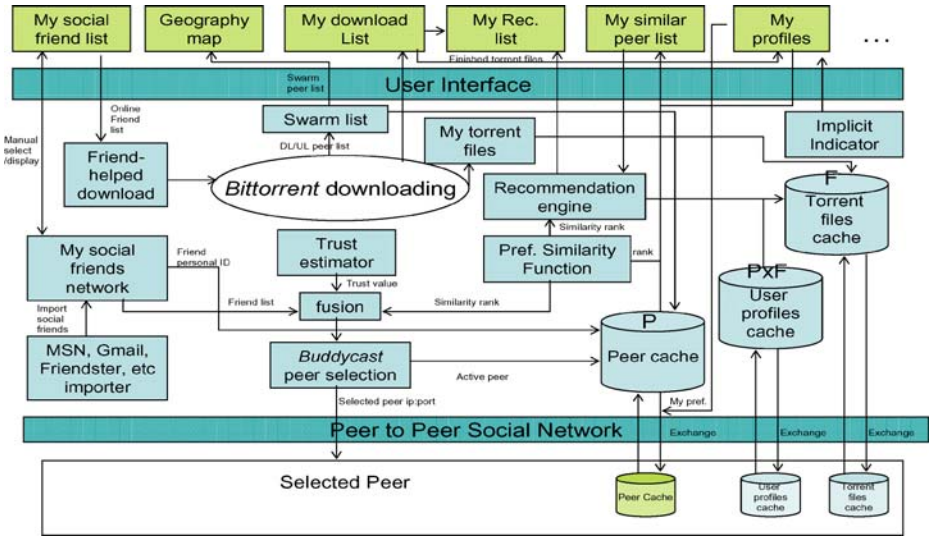

Fig. 2 The system architecture of *Tribler*

available content in the peer-to-peer community can be explored using novel personalized tag-based navigation.

This paper focuses on the personalization aspects of the *Tribler* system. Firstly, we review the related work. Secondly, we describe our system design and the underlying approaches. Finally, we present our experiments to examine the effectiveness of the underlying approaches in the *Tribler* system.

2 Related work

2.1 Recommendation

We adopt recommendations to help users discover available relevant content in a more natural way. Furthermore, it observes and integrates the interests of a user within the discovery process. Recommender systems propose a similarity measure that expresses the relevance between an item (the content) and the profile of a user. Current recommender systems are mostly based on collaborative filtering, which is a filtering technique that analyzes a rating database of user profiles for similarities between users (user-based) or programs (item-based). Others focus on content-based filtering, which, for instance, based on the EPG data [2].

The profile information about programs can either be based on ratings (explicit interest functions) or on log-archives (implicit interest functions). Correspondingly, their differences lead to two different approaches of collaborative filtering: *rating-based* and *log-based*. The majority of the literature addresses rating-based collaborative filtering, which has been studied in depth [16]. The different rating-based approaches are often classified as memory-based [3, 10] or model-based [11].

In the memory-based approach, all rating examples are stored *as-is* into memory (in contrast to learning an abstraction). In the prediction phase, similar users or items are sorted

based on the memorized ratings. Based on the ratings of these similar users or items, a recommendation for the query user can be generated. Examples of memory-based collaborative filtering include item correlation-based methods [21] and locally weighted regression [3]. The advantage of memory-based methods over their model-based alternatives is that they have less parameters to be tuned, while the disadvantage is that the approach cannot deal with data sparsity in a principled manner.

In the model-based approach, training examples are used to generate a model that is able to predict the ratings for items that a query user has not rated before. Examples include decision trees [3], latent class models [11], and factor models [4]. The 'compact' models in these methods could solve the data sparsity problem to a certain extent. However, the requirement of tuning an often significant number of parameters or hidden variables has prevented these methods from practical usage.

Recently, to overcome the drawbacks of these approaches to collaborative filtering, researchers have started to combine both memory-based and model-based approaches [19, 23, 25]. For example, [25] clusters the user data and applies intra-cluster smoothing to reduce sparsity. Wang et al. [23] propose a unified model to combine user-based and item-based approaches for the final prediction, and does not require to cluster the data set a priori.

Few log-based collaborative filtering approaches have been developed thus far. Among them are the item-based top-*N* collaborative filtering approach [6] and Amazon's item-based collaborative filtering [15]. In previous work, we developed a probabilistic framework that gives a probabilistic justification of a log-based collaborative filtering approaches [22] that is also employed in this paper to make TV program recommendation in *Tribler*.

2.2 Distributed recommendation

In P2P TV systems, both the users and the supplied programs are widely distributed and change constantly, which makes it difficult to filter and localize content within the P2P network. Thus, an efficient filtering mechanism is required to be able to find suitable content.

Within the context of P2P networks there is, however, no centralized rating database, thus making it impossible to apply current collaborative filtering approaches. Recently, a few early attempts towards decentralized collaborative filtering have been introduced [1, 17]. In [17], five architectures are proposed to find and store user rating data to facilitate rating-based recommendation: (1) a central server, (2) random discovery similar to Gnutella, (3) transitive traversal, (4) Distributed Hash Tables (DHT), and (5) secure Blackboard. In [1], item-to-item recommendation is applied to TiVo (a Personal Video Recorder system) in a client-server architecture. These solutions aggregate the rating data in order to make a recommendation and are independent of any semantic structures of the networks. This inevitably increases the amount of traffic within the network. To avoid this, a novel item-buddy-table scheme is proposed in [24] to efficiently update the calculation of item-to-item similarity.

Jelasity and van Steen [13] introduced newscast an epidemic (or gossip) protocol [7] that exploits randomness to disseminate information without keeping any static structures or requiring any sort of administration. Although these protocols successfully operate dynamic networks, their lack of structure restricts them to perform these services in an efficient way.

In this paper, we propose a novel algorithm, called *BuddyCast*, that, in contrast to newscast, generates a semantic overlay on the epidemic protocols by implicitly clustering peers into social networks. Since social networks have small-world network characteristics

the user profiles can be disseminated efficiently. Furthermore, the resulting semantic overlays are also important for the membership management and content discovery, especially for highly dynamic environments with nodes joining and leaving frequently.

2.3 Learning user interest

Rating-based collaborative filtering requires users to explicitly indicate what they like or do not like [3, 9]. For TV recommendation, the rated items could be preferred channels, favorite genres, and hated actors. Previous research [5, 18] has shown that users are unlikely to provide an extensive list of explicit ratings which eventually can seriously degrade the performance of the recommendation. Consequently, the interest of a user should be learned in an implicit way.

This paper learns these interests from TV watching habits such as the zapping behavior. For example, zapping away from a program is a hint that the user is not interested, or, alternatively, watching the whole program is an indication that the user liked that show. This mapping, however, is not straightforward. For example, it is also possible that the user likes this program, but another channel is showing an even more interesting program. In that case zapping away is not an indication that the program is not interesting. In this paper we introduce a simple heuristic scheme to learn the user interest implicitly from the zapping behavior.

3 System design

This section describes a heuristic scheme that implicitly learns the interest of a user in TV programs from zapping behavior in that way avoiding the need for explicit ratings. Secondly, we present a distributed profile exchanger, called *BuddyCast*, which enables the formation of social groups as well as distributed content recommendation (ranking of TV programs). We then introduce the user-item relevance model to predict interesting programs for each user. Finally, we demonstrate a user interface incorporating these personalized aspects, i.e., personalized tag-based browsing as well as visualizing your social group.

3.1 User profiling from zapping behavior

We use the zapping behavior of a user to learn the user interest in the watched TV programs. The zapping behavior of all users is recorded and coupled with the EPG (Electronic Program Guide) data to generate program IDs. In the *Tribler* system different TV programs have different IDs. TV series that consists of a set of episodes, like “Friends” or a general “news” program, get one ID (all episodes get the same ID) to bring more relevance among programs.

For each user u_k the interest in TV program i_m can be calculated as follows:

$$x_k^m = \frac{\text{WatchedLength}(m,k)}{\text{OnAirLength}(m) \cdot \text{freq}(m)} \tag{1}$$

WatchedLength(m,k) denotes the duration that the user u_k has watched program i_m in seconds. *OnAirLength*(m) denotes the entire duration in seconds of the program i_m, on air (cumulative with respect to episodes or reruns). *Freq*(m) denotes the number of times program i_m has been broadcast (episodes are considered to be a rerun), in other words *OnAirLength(m)/freq(m)* is the average duration of a ‘single’ broadcast, e.g., average

duration of an episode. This normalization with respect to the number of times a program has been broadcast is taken into consideration since programs that are frequently broadcast also have more chance that a user gets to watch it.

Experiments (see Fig. 10) showed that, due to the frequent zapping behaviors of users, a large number of x_k^m's have very small values (zapping along channels). It is necessary to filter out those small valued x_k^m's in order to: (1) reduce the amounts of user interest profiles that need to be exchanged, and (2) improve recommendation by excluding these noisy data. Therefore, the user interest values x_k^m are thresholded resulting in binary user interest values:

$$y_k^m = 1 \text{ if } x_k^m > T \text{ and } y_k^m = 0 \text{ otherwise} \tag{2}$$

Consequently, y_k^m indicates whether user u_k likes program $i_m\left(y_k^m = 1\right)$ or not $\left(\mathrm{y_k^m} = 0\right)$. The optimal threshold T will be obtained through experimentation.

3.2 BuddyCast profile exchange

BuddyCast generates a semantic overlay on the epidemic protocols by implicitly clustering peers into social networks according to their profiles. It works as follows. Each user maintains a list of top-N most similar users (a.k.a. taste buddies or social network) along with their current profile lists. To be able to discover new users, each user also maintains a *random cache* to record the top-N most fresh "random" IP addresses.

Periodically, as illustrated in Fig. 3a, a user connects either to one of his/her buddies to exchange social networks and current profile list (*exploitation*), or to a new randomly chosen user from the random cache to exchange this information (*exploration*). To prevent reconnecting to a recently visited user in order to maximize the exploration of the social network, every user also maintains a list with the K most recently visited users (excluding the taste buddies).

Different with the gossip-based approaches, which only consider exploration (randomly select a user to connect), *BuddyCast* algorithm considers exploration as well as exploitation. Previous study has shown that a set of user profiles is not a random graph and has a certain clustering coefficient [24]. That is the probability that two of user A's buddies (top-N similar users) will be buddies of one another is greater than the probability that two randomly chosen users will be buddies. Based on this observation, we connect users according to their similarity ranks. The more similar a user, the more chance it gets selected to exchange user profiles. Moreover, to discover new users and prevent network divergence, we also add some randomness to allow other users to have a certain chance to be selected.

To find a good balance between exploitation (making use of small-world characteristics of social networks) and exploration (discovering new worlds), as illustrated in Fig. 3b, the following procedure is adopted. First, δN random users are chosen, where δ denotes the exploration-to-exploitation ratio and $0 \leq \delta N \leq$ number of users in the random cache. Then, these random users are joined with the N buddies, and a ranked list is created based on the similarity of their profile lists with the profile of the user under consideration. Instead of connecting to the random users to get their profile lists, the random users are assigned the lowest ranks. Then, one user is randomly chosen from this ranked list according to a roulette wheel approach (probabilities proportional to the ranks), which gives taste buddies a higher probability to be selected than random users.

Once a user has been selected, the two caches are updated. In random cache, the IP address is updated. In the buddy cache, the buddy list of the selected user is merged. The

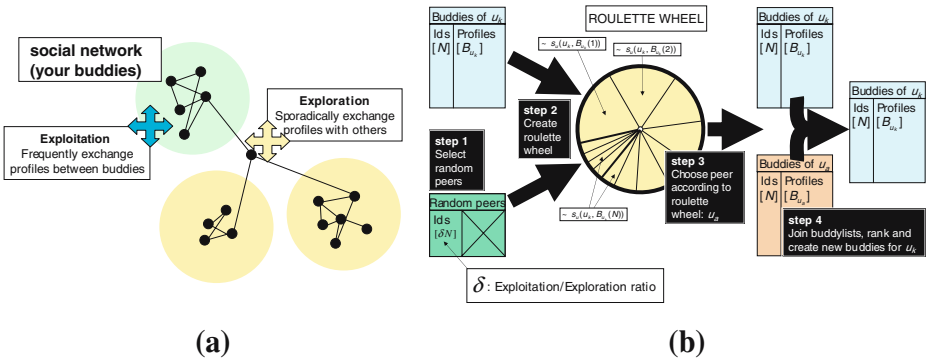

Fig. 3 The illustration of the *BuddyCast* algorithm

buddy cache is then ranked (according to the similarity with the own profile), and the top-N best ranked users are kept.

3.3 Recommendation by relevance models

In the *Tribler* system, after collecting user preferences by using the *BuddyCast* algorithm, we are ready to use the collected user preferences to identify (rank) interesting TV programs for each individual user in order to facilitate taste-based navigation. The following characteristics make our ranking problem more similar to the problem of text retrieval than the existing rating-based collaborative filtering. 1) The implicit interest functions introduced in the previous section generate binary-valued preferences. Usually, one means 'relevance' or 'likeness', and zero indicates 'non-relevance' or 'non-likeness'. Moreover, non-relevance and non-likeness are usually not observed. This is similar to the concept of 'relevance' in text retrieval. 2) The goal for rating-based collaborative filtering is to predict the rating of users, while the goal for the log-based algorithms is to rank the items to the user in order of decreasing relevance. As a result, evaluation is different. In rating-based collaborative filtering, the mean square error (MSE) of the predicted rating is used, while in log-based collaborative filtering, recall and precision are employed.

This paper adopts the probabilistic framework developed for text retrieval [14] and proposes a probabilistic user-item relevance model to measure the relevance between user interests and TV programs, which intends to answer the following basic question:

> *What is the probability that* this *program is relevant to* this *user, given his or her profile*?

To answer this question, we first define the sample space of relevance: Φ_R. It has two values: 'relevant' r and 'non-relevant' $\bar{r}$. Let R be a random variable over the sample space Φ_R. Likewise, let U be a discrete random variable over the sample space of *user id*'s: $\Phi_U = \{u_1, \ldots, u_K\}$ and let I be a random variable over the sample space of *item id*'s: $\Phi_R = \{i_1, \ldots, i_M\}$, where K is the number of users and M the number of items in the collection. In other words, U refers to the user identifiers and I refers to the item identifiers.

We then denote P as a probability function on the joint sample space $\Phi_U \times \Phi_I \times \Phi_R$. In a probability framework, we can answer the above basic question by estimating the probability of relevance $P(R|U, I)$. The relevance rank of items in the collection Φ_I for a given user $U = u_k$ (i.e., retrieval status value (RSV) of a given target item toward a user) can be formulated as the odds of the relevance:

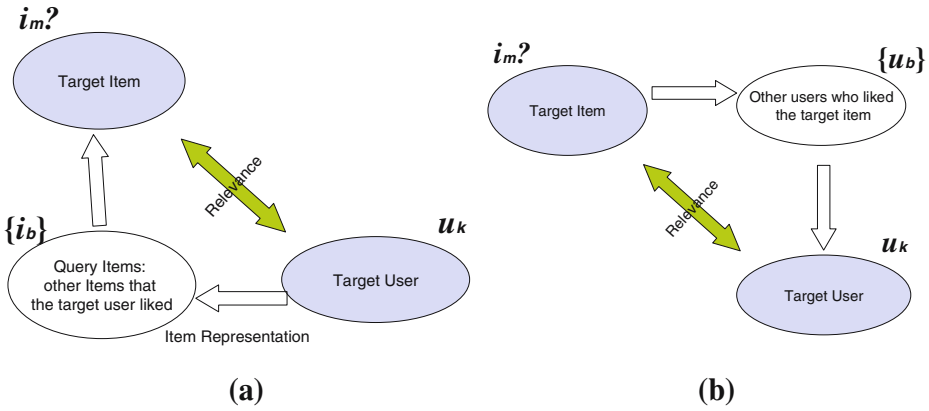

Fig. 4 Two different models in the user-item relevance model

$$\mathrm{RSV}_{u_k}(i_m) = \frac{\log P(r|u_k, i_m)}{\log P(\overline{r}|u_k, i_m)} \tag{3}$$

For simplicity, $R=r$, $R=\overline{r}$, $U=u_k$ and $I=i_m$ are denoted as r, $\overline{r}$, u_k and i_m, respectively.

Hence, the evidence for the relevance of an item towards a user is based on both the positive evidence (indicating the relevance) as well as the negative evidence (indicating the non-relevance). Once we know, for a given user, the *RSV* of each item I in the collection (excluding the items that the user has already expressed interest in), we sort these items in decreasing order. The highest ranked items are recommended to the user.

In order to estimate the conditional probabilities in (3), i.e., the relevance and non-relevance between the user and the item, we need to factorize the equation along the item or the user dimension. We propose to consider both *item-based generation* (i.e., using items as features to represent the user) and *user-based generation* (i.e., treating users as features to represent an item). This is illustrated in Fig. 4.

3.3.1 Item-based generation model

By factorizing $P(g|u_k, i_m)$ with

$$\frac{P(u_k|i_m, g)P(g|i_m)}{P(u_k|i_m)},$$

the following log-odds ratio can be obtained from (3):

$$\begin{aligned} &\mathrm{RSV}_{u_k}(i_m) \\ &= \log \tfrac{P(r|i_m,u_k)}{P(\overline{r}|i_m,u_k)} \\ &= \log \tfrac{P(u_k|i_m,r)}{P(u_k|i_m,\overline{r})} + \log \tfrac{P(i_m|r)P(r)}{P(i_m|\overline{r})P(\overline{r})} \end{aligned} \tag{4}$$

Equation (4) provides a general ranking formula by employing the evidences from both relevance and non-relevance cases. When there is no explicit evidence for non-relevance, following the language modeling approach to information retrieval [14], we now assume that: (1) independence between u_k and i_m in the non-relevance case ($\overline{r}$), i.e., $P(u_k|i_m, \overline{r}) = P(u_k|\overline{r})$; and, (2) equal priors for both u_k and i_m, given that the item is non-

relevant. Then the two terms corresponding to non-relevance can be removed and the *RSV* becomes:

$$\mathrm{RSV}_{u_k}(i_m) = \log P(u_k|i_m, r) + \log P(i_m|r) \tag{5}$$

Note that the two negative terms in (5) can always be added to the model, when the negative evidences are captured.

To estimate the conditional probability $P(u_k|i_m, r)$ in (5), consider the following: Instead of placing users in the sample space of *user id*'s, we can also use the set of items that the user likes (denoted L_{u_k} or $\{i_b\}$) to represent the user (u_k) (see the illustration in Fig. 4a). This step is similar to using a 'bag-of-words' representation of queries or documents in the text retrieval domain. This implies: $P(u_k|i_m, r) = P(L_{u_k}|i_m, r)$. We call these representing items the *query items*. Note that, unlike the target item i_m, the query items do not need to be ranked since the user has already expressed interest in them.

Further, we assume that the items $\{i_b\}$ in the user profile list L_{u_k} (query items) are conditionally independent from each other. Although this naive Bayes assumption does not hold in many real situations, it has been empirically shown to be a competitive approach (e.g., in text classification [8]). Under this assumption, (5) becomes:

$$\begin{aligned} &\mathrm{RSV}_{u_k}(i_m) \\ &= \log P(L_{u_k}|i_m, r) + \log P(i_m|r) \\ &= \left(\sum_{\forall i_b : i_b \in L_{u_k}} \log P(i_b|i_m, r) \right) + \log P(i_m|r) \end{aligned} \tag{6}$$

where the conditional probability $P(i_b|i_m, r)$ corresponds to the relevance of an item i_b, given that another item i_m is relevant. This probability can be estimated by counting the number of user profiles that contain both items i_b and i_m, divided by the total number of user profiles in which i_m exists:

$$P_{ml}(i_b|i_m, r) = \frac{P(i_b, i_m|r)}{P(i_m|r)} = \frac{c(i_b, i_m)}{c(i_m)} \tag{7}$$

Using the frequency count to estimate the probability corresponds to using its maximum likelihood estimator. However, many item-to-item co-occurrence counts will be zero, due to the sparseness of the user-item matrix. Therefore, we apply a smoothing technique to adjust the maximum likelihood estimation.

A linear interpolation smoothing can be defined as a linear interpolation between the maximum likelihood estimation and background model. To use it, we define:

$$P(i_b|i_m, r) = (1 - \lambda_i) P_{ml}(i_b|i_m, r) + \lambda_i P_{ml}(i_b|r)$$

where P_{ml} denotes the maximum likelihood estimation. The item prior probability $P_{ml}(i_b|r)$ is used as background model. Furthermore, the parameter λ_i in [0,1] is a parameter that balances the maximum likelihood estimation and background model (a larger λ_i means more smoothing). Usually, the best value for λ_i is found from a training data by using a cross-validation method.

Linear interpolation smoothing leads to the following RSV:

$$\mathrm{RSV}_{u_k}(i_m) = \left(\sum_{\forall i_b : i_b \in L_{u_k}} \log\left((1-\lambda_i) P_{ml}(i_b|i_m, r) + \lambda_i P_{ml}(i_b|r)\right) \right) + \log P_{ml}(i_m|r) \quad (8)$$

where the maximum likelihood estimations of the item prior probability densities are given as follows:

$$P_{ml}(i_b|r) = \frac{c(i_b, r)}{c(r)}, P_{ml}(i_m|r) = \frac{c(i_m, r)}{c(r)} \quad (9)$$

3.3.2 User-based generation model

Similarly, by factorizing $P(g|u_k, i_m)$ with

$$\frac{P(i_m|u_k, g)P(g|u_k)}{P(i_m|u_k)}$$

the following log-odds ratio can be obtained from (3) :

$$\mathrm{RSV}_{u_k}(i_m) = \log \frac{P(i_m|u_k, r)}{P(i_m|u_k, \bar{r})} + \log \frac{P(u_k|r)P(r)}{P(u_k|\bar{r})P(\bar{r})} \propto \log \frac{P(i_m|u_k, r)}{P(i_m|u_k, \bar{r})} \quad (10)$$

When the non-relevance evidence is absent, and following the language model in information retrieval [14], we now assume equal priors for i_m in the non-relevant case. Then, the non-relevance term can be removed and the RSV becomes:

$$\mathrm{RSV}_{u_k}(i_m) = \log P(i_m|u_k, r) \quad (11)$$

Instead of using the item list to represent the user, we use each user's judgment as a feature to represent an item (see the illustration in Fig. 4b). For this, we introduce a list L_{i_m} for each item i_m, where $m = \{1,...,M\}$. This list enumerates the users who have expressed interest in the item i_m. $L_{i_m}(u_k) = 1$ (or $u_k \in L_{i_m}$) denotes that user u_k is in the list, while $L_{i_m}(u_k) = 0$ (or $u_k \notin L_{i_m}$) otherwise. The number of users in the list corresponds to $|L_{i_m}|$.

Replacing i_m with L_{i_m}, after we assume each user's judgment to a particular item is independent, we have:

$$\mathrm{RSV}_{u_k}(i_m) = \log P(L_{i_m}|u_k, r) = \sum_{\forall u_b : u_b \in L_{i_m}} \log P(u_b|u_k, r) \quad (12)$$

Similar to the item-based generation model, when we use linear interpolation smoothing to estimate $P(u_b|u_k, r)$, we obtain the final ranking formula:

$$\mathrm{RSV}_{u_k}(i_m) = \log P(L_{i_m}|u_k, r) = \sum_{\forall u_b : u_b \in L_{i_m}} \log\left((1-\lambda_u) P_{ml}(u_b|u_k, r) + \lambda_u P_{ml}(u_b|r)\right) \quad (13)$$

where $\lambda_u \in [0, 1]$ is the smoothing parameter.

3.4 Statistical ranking mechanisms

Our models provide a very intuitive understanding of the statistical ranking mechanisms that play a role in log-based collaborative filtering. More formally, from (8) and (13), we

can obtain the following ranking functions for the user-based generation and item-based generation models, respectively (see [22] for the detailed information):

Item-based Generation Model:

$$\mathrm{Rank}_{u_k}(i_m) = \left(\sum_{\forall i_b : i_b \in L_{u_k} \cap c(i_b, i_m) > 0} \log\left(1 + \frac{(1-\lambda_i) P_{ml}(i_b|i_m, r)}{\lambda_i P_{ml}(i_b|r)}\right)\right) + \log P(i_m|r) \tag{14}$$

User-based Generation Model:

$$\mathrm{Rank}_{u_k}(i_m) = \left(\sum_{\forall u_b : u_b \in L_{i_m} \cap c(u_b, u_k) > 0} \log\left(1 + \frac{(1-\lambda_u) P_{ml}(u_b|u_k, r)}{\lambda_u P_{ml}(u_b|r)}\right)\right) + |L_{i_m}| \log \lambda_u \tag{15}$$

From the item-based generation model (14), we can see that: (1) The relevance rank of a target item i_m is the sum of its popularity (prior probability $P(i_m|r)$) and its co-occurrence (first term in (14)) with the items i_b in the profile list of the target users. The co-occurrence is higher if more users express interest in target item (i_m) as well as item i_b. However, the co-occurrence should be suppressed more when the popularity of the item in the profile of the target user ($P(i_b|r)$) is higher. (2) When λ_i approaches 0, smoothing from the background model is minimal. It emphasizes the co-occurrence count, and the model reduces to the traditional item-based approach [5]. When the λ_i approaches 1, the model is more smooth, emphasizing the background model. When the parameter equals 1, the ranking becomes equivalent to *coordination level matching*, which is simply counting the number of times for which $c(i_b, i_m) > 0$.

From the user-based generation model (15), we can see that the relevance rank is calculated based on the opinions of other similar users. For a target user and target program, the rank of their relevance is basically the sum of the target user's co-occurrence with other similar users who have liked the target program. The co-occurrence is higher if there are more programs the two users agree upon (express interest in the same program). However, the co-occurrence should be suppressed more when the similar user has liked more programs, since he or she is less discriminative.

3.5 Personalized user interfaces

When designing a user interface for a distributed system like *Tribler*, it is important to reach and maintain a critical mass since the users are the decisive factors of the system's success [9]. Therefore, several usability aspects have to be dealt with: (1) the wealth and complexity of content, (2) the lack of trust among users, (3) no guarantee of system or content integrity, and (4) the need for voluntary cooperation among users. Here we only addresses and illustrates the first two aspects.

A user is unable to deal with an unlimited number of programs to choose from. Our distributed recommender system helps to filter according to the implicitly learned interests. Subsequently it becomes important to communicate the results in a way that makes sense to

a user and allows for exploration and exploitation of the available content in spite of the lack of trust amongst users.

In Fig. 5 we illustrate our thoughts on a user interface for a decentralized recommender system, as applied in *Tribler*. Figure 5a is *Tribler*'s opening screen. In Fig. 5b we show a user's social network in which relations are expressed in social distances: friend, friend-of-a-friend, or taste buddy (which is obtained by running our *BuddyCast* algorithm). With this the exploitation of the community is stimulated because users can look into each other's hard disks directly, thus rewarding the risks users take when allowing taste buddies to communicate with them. Figure 5c shows the personalized tag-based navigation, which is a popular way of displaying filtered results or recommendations, as in http://www.Flickr.com or http://www.CiteULike.org. The font size of each tag reflects its relevance towards user. The relevance rank of each tag can be calculated by summing up all the relevance ranks from its attached programs. This feature incorporates a reflection on the origins, and trustworthiness of the recommended content. We believe this will reduce the uncertainty about the quality and integrity of the programs and lack of trust among users. Moreover it stimulates users to explore new content in a natural way. Figure 5d demonstrates content descriptions of recommended programs. As with the tag-based navigations, it incorporates a reflection on the origins, quality, and integrity. Furthermore, it provides more background information on items, like in http://www.IMDb.com.

4 Experiments and results

We have conducted a set of experiments with the *Tribler* system on a real data set containing the TV zapping behavior of users to address the following questions:

1. What zapping behaviors do we observe and what can be learned from these behaviors to implicitly derive the interest of users in TV programs?
2. How sensitive is the recommendation of TV programs as a function of the user interest threshold T and what is the optimal value taking into account the efficiency of exchanging interest between users?
3. How efficient is our proposed *BuddyCast* algorithm compared to the newscast algorithm when we want to exchange user interest profiles?

4.1 Data set

We used a data set that contained the TV zapping behavior of 6,000 Dutch users over 19 channels from the SKO foundation.[1] The remote controls actions were recorded from January 1 to January 31, 2003. Some basic characteristics of this data set are shown in Fig. 6. We employed the EPG data set obtained from http://Omroep.nl (an online TV program guide) to find TV program IDs.[2] This resulted in 8,578 unique programs and 27,179 broadcasting slots over the 19 Channels in that period (this includes reruns and episodes of the samne TV program). Figure 9 shows statistics about the number of times

[1] http://www.kijkonderzoek.nl

[2] http://omroep.nl

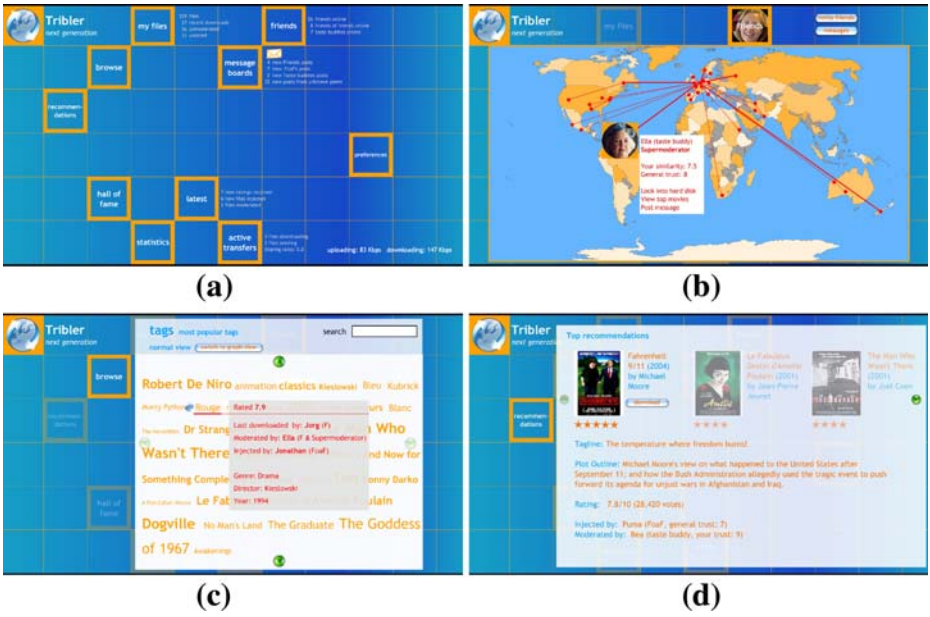

Fig. 5 User interface of Tribler

TV programs are broadcast. For instance, news is broadcast several times a day. Series have different episodes and are broadcast for example weekly.

Another dataset we used to evaluate our recommendation method is called Audioscrobbler dataset. The data set is collected from the music play-lists of the users in the Audioscrobbler community[3] by using a plug-in in the users' media players (for instance, Winamp, iTunes, XMMS etc). Plug-ins send the title (song name and artist name) of every song users play to the Audioscrobbler server, which updates the user's musical profile with the new song. That is, when a user plays a song in a certain time, this transaction is recorded as a form of {userID, itemID, t} tuple in the database.

4.2 Observations of the data set

This SKO TV data set can be used to analyze the zapping behavior of users for particular TV programs. In Fig. 7 this is shown for a more popular movie, "Live and Let Die" (1973), and a less popular movie, "Someone she knows" (1994).

For example, when we look at the beginning of the two programs, it clearly shows the difference of the user attention for the less popular film, i.e., the number of watching users drops significantly for the first 5 min or so. Probably, these users first zapped into the channel to check out the movie and realized that it was not interesting movie for them and zapped away. Contrarily, the number of watching users steadily increasing in the first minutes for the more popular.

Another interesting observation in both figures is that during the whole broadcasting time, there were some intervals of about 5–10 min, in which the number of watching users

[3] https://last.fm

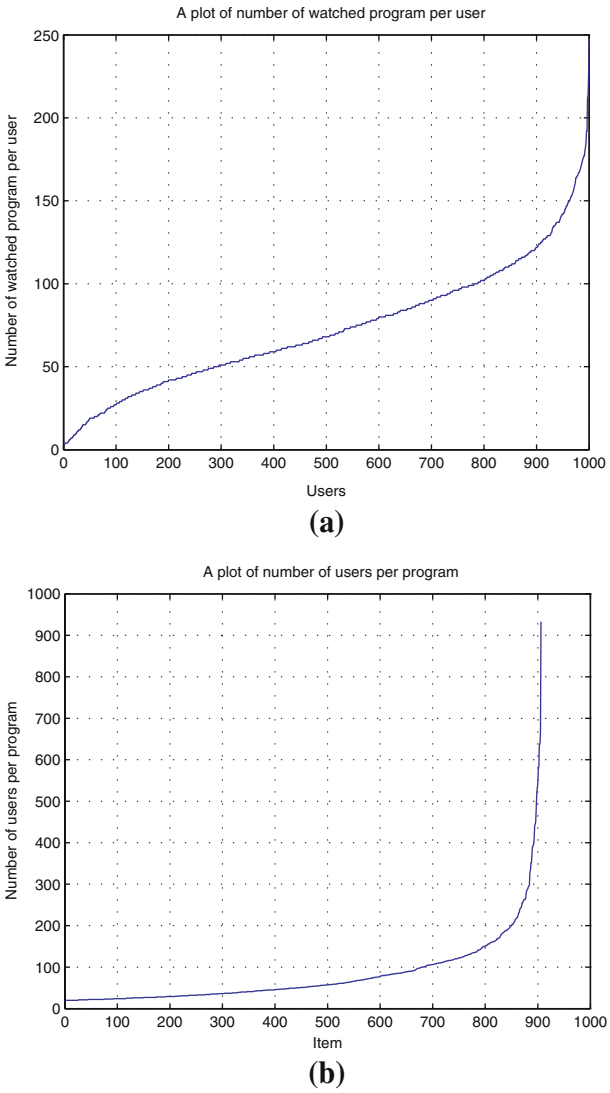

Fig. 6 SKO data set of user actions on remote controls

dropped. This is because some users left the channel when commercials began and zapped back again when they had supposedly ended.

Figure 8 shows the number of users with respect to their percentages of watching times (*WatchLenght*(*k,m*)/*OnAirlength*(*m*)) for programs with different number of times that they are broadcast (on-air times of 1, 5 and 9).

This shows clearly two peaks: the larger peak on the left indicates a large number of users who only watched small parts of a program. The second smaller peak on the right indicates that a large number of users watched the whole programs once regardless of the number of times that the program was broadcast. That is, the right peak happens in 20% of the programs that are broadcast five times (one fifth), and in 11% of the programs that are broadcast nine times (one ninth), etc. There is a third peak which happens in 22% in the

programs which are broadcast nine times. This indicates that there are still a few users who watched the entire program twice, for example to follow a series.

These observations motivated us to normalize the percentage of watching time by the number of broadcastings of a program as explained in (2), in order to arrive at the measure of interest within a TV program. This normalized percentage is shown in Fig. 10. Now all the second peaks are located at the 100% position.

4.3 Learning the user interest threshold

The threshold level, T, above which the normalized percentage of watching time is considered to express interest in a TV program (3) is determined by evaluating the performance of the recommendation for different setting of this threshold.

The recommendation performance is measured by using *precision* and *recall* of a set of test users. Precision measures the proportion of recommended programs that the user truly

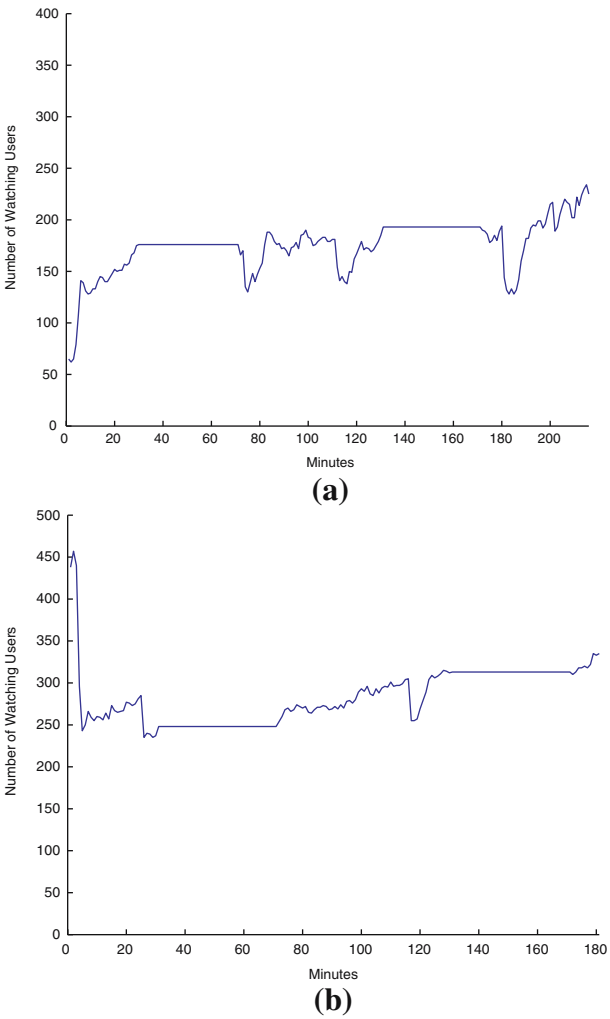

Fig. 7 Program attention

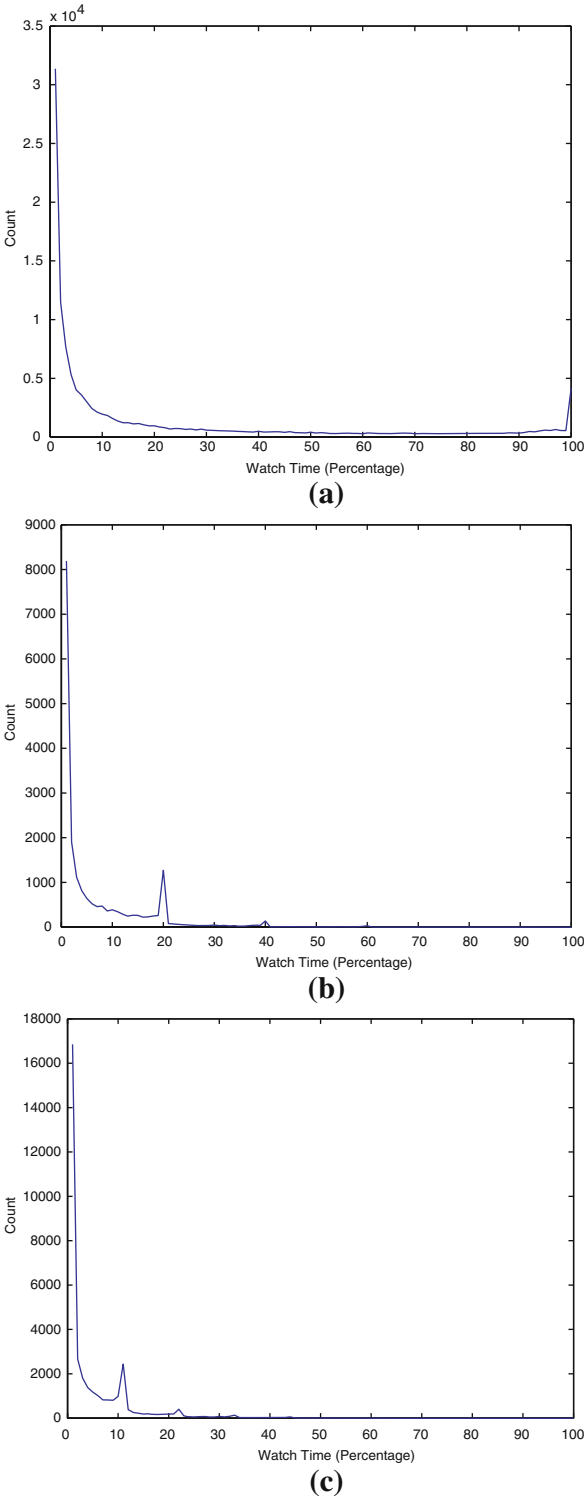

Fig. 8 Percentage of watching time for programs with different on-air times

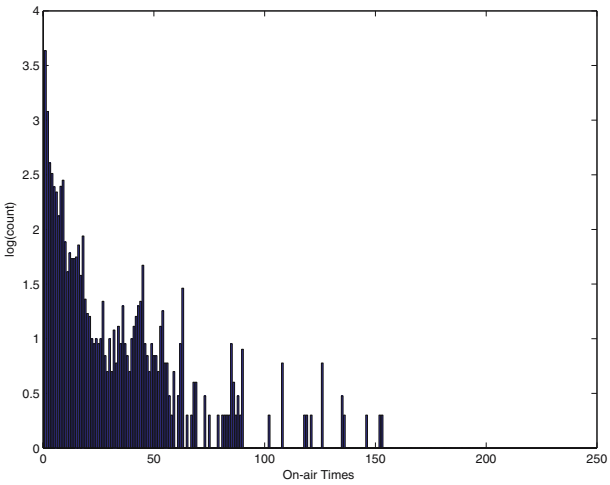

Fig. 9 Program on-air times during Jan. 1 to Jan 30, 2003

likes. Recall measures the proportion of the programs that a user truly likes that are recommended. In case of making recommendations, precision seems more important than recall. However, to analyze the behavior of our method, we report both metrics on our experimental results.

Since we lack information on what the users liked, we considered programs that a user watched more than once ($x_{k,m}>1$) to be programs that the user likes and all other programs as shows that the user does not like. Note that, in this way , only a fraction of the programs that the user *truly* liked are captured. Therefore, the measured precision *underestimates* the true precision [12].

For cross-validation, we randomly divided this data set into a training set (80% of the users) and a test set (20% of the users). The training set was used to estimate the model. The test set was used for evaluating the accuracy of the recommendations on the new users,

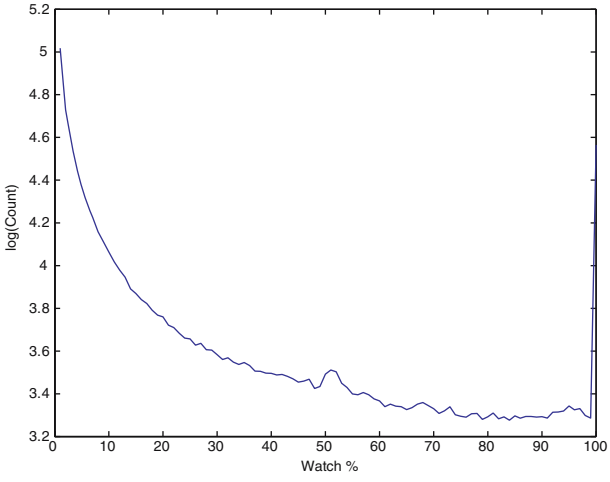

Fig. 10 Normalized percentage of watching time

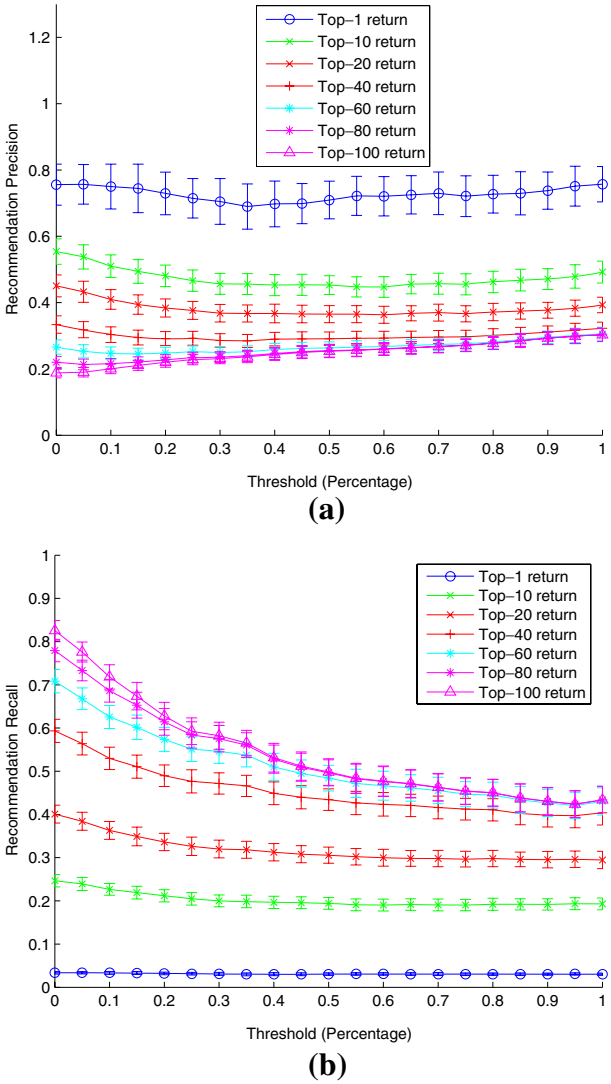

Fig. 11 Recommendation performance v.s. threshold T

whose user profiles are not in the training set. Results are obtains by averaging five different runs of such a random division.

We plotted the performance of recommendations (both precision and recall) against the threshold on the percentage of watching time in Fig. 11. We also varied the number of programs returned by the recommender (top-1, 10, 20, 40, 80 or 100 recommended TV programs). Figure 11a shows that in general, the threshold does not affect the precision too much. For the large number of programs recommended, the precision becomes slightly better when there is a larger threshold. For larger number of recommended programs, the recall, however, drops for larger threshold values (shown in Fig. 11b). Since the threshold does not affect the precision too much, a higher threshold is chosen in order to reduce the

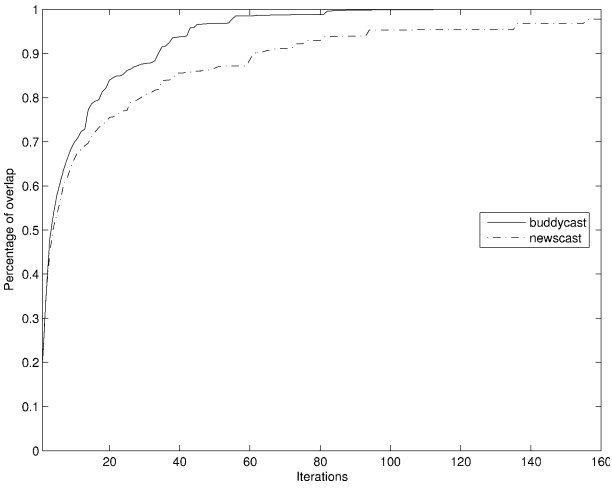

Fig. 12 Convergence of our *BuddyCast* algorithm

length of the user interest profiles to be exchanged within the network. For that reason we have chosen a threshold value of 0.8.

4.4 Convergence behavior of BuddyCast

We have emulated our *BuddyCast* algorithm using a cluster of PCs (the DAS-2[4] system). The simulated network consisted of 480 users distributed uniformly over 32 nodes. We used the user profiles of 480 users. Each user maintained a list of ten taste buddies (N=10) and the 10 last visited users (K=10). The system was initialized by giving each user a random other user. The exploration-to-exploitation δ was set to 1.

Figure 12 compares the convergence of *BuddyCast* to that of newscast (randomly select connecting users, i.e., $\delta \rightarrow \infty$). After each update we compared the list of top-N taste buddies with a pre-compiled list of top-N taste buddies generated using all data (centralized approach). In Fig. 12, the percentage of overlap is shown as a function of time (represented by the number of updates). The figure shows that the convergence of BuddyCast is much faster than that of the Newscast approach.

4.5 Recommendation performance

We first studied the behavior of the linear interpolation smoothing for recommendation. For this, we plotted the average precision and recall rate for the different values of the smoothing parameter λ_i in the Audioscrobbler data set. This is shown in Fig. 13.

Figure 13a and b show that both precision and recall drop when λ_i reaches its extreme values zero and one. The precision is sensitive to λ_i, especially the early precision (when only a small number of items are recommended). Recall is less sensitive to the actual value of this parameter, having its optimum at a wide range of values. Effectiveness tends to be higher on both metrics when λ_i is large; when λ_i is approximately 0.9, the precision seems

[4] http://www.cs.vu.nl/das2

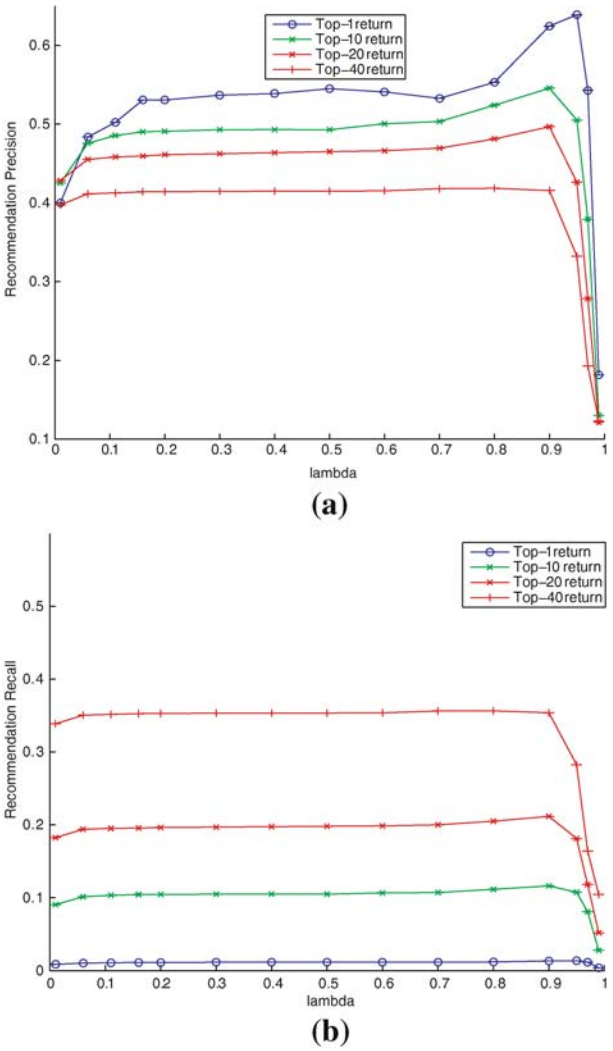

Fig. 13 Recommendation performance of the linear interpolation smoothing

optimal. An optimal range of λ_i near one can be explained by the sparsity of user profiles, causing the prior probability $P_{ml}(i_b|r)$ to be much smaller than the conditional probability $P_{ml}(i_b|i_m, r)$. The background model is therefore only emphasized for values of λ_i closer to one. In combination with the experimental results that we obtained, this suggests that smoothing the co-occurrence probabilities with the background model (prior probability $P_{ml}(i_b|r)$) improves recommendation performance.

Next, we compared our relevance model to other log-based collaborative filtering approaches. Our goal here is to see, using our user-item relevance model, whether the smoothing and inverse item frequency should improve recommendation performance with respect to the other methods. For this, we focused on the item-based generation (denoted as UIR-Item). We set λ_i to the optimal value 0.9. We compared our results to those obtained

Table 1 Comparison of recommendation performance

	Top-1 item	Top-10 item	Top-20 item	Top-40 item
(a) Precision				
UIR-item	0.62	0.52	0.44	0.35
Item-t FIDF	0.55	0.47	0.40	0.31
Item-CosSim	0.56	0.46	0.38	0.31
Item-CorSim	0.50	0.38	0.33	0.27
User-CosSim	0.55	0.42	0.34	0.27
(b) Recall				
UIR-item	0.02	0.15	0.25	0.40
Item-TFIDF	0.02	0.15	0.26	0.41
Item-CosSim	0.02	0.13	0.22	0.35
Item-CorSim	0.01	0.11	0.19	0.31
User-CosSim	0.02	0.15	0.25	0.39

with the *top-N-suggest* recommendation engine, a well-known log-based collaborative filtering implementation[5] [6]. This engine implements a variety of log-based recommendation algorithms. We compared our own results to both the item-based $TF \times IDF$-like version (denoted as ITEM-TFIDF) as well the user-based cosine similarity method (denoted as User-CosSim), setting the parameters to the optimal ones according to the user manual. Additionally, for item-based approaches, we also used other similarity measures: the commonly used cosine similarity (denoted as Item-CosSim) and Pearson correlation (denoted as Item-CorSim). Results are shown in Table 1.

For the precision, our user-item relevance model with the item-based generation (UIR-Item) outperforms other log-based collaborative filtering approaches for all four different number of returned items. Overall, $TF \times IDF$-like ranking ranks second. The obtained experimental results demonstrate that smoothing contributes to a better recommendation precision in the two ways also found by [26]. On the one hand, smoothing compensates for missing data in the user-item matrix, and on the other hand, it plays the role of inverse item frequency to emphasize the weight of the items with the best discriminative power. With respect to recall, all four algorithms perform almost identically. This is consistent to our first experiment that recommendation precision is sensitive to the smoothing parameters while the recommendation recall is not.

5 Conclusions

This paper discussed personalization in a personalized peer-to-peer television system called *Tribler*, i.e., (1) the exchange of user interest profiles between users by automatically creating social groups based on the interest of users, (2) learning these user interest profiles from zapping behavior, (3) the relevance model to predict user interest, and (4) a personalized user interface to browse the available content making use of recommendation technology. Experiments on two real data sets show that personalization can increase the effectiveness to exchange content and enables to explore the wealth of available TV programs in a peer-to-peer environment.

[5] http://www-users.cs.umn.edu/~karypis/suggest/

References

1. Ali K, van Stam W (2004) TiVo: making show recommendations using a distributed collaborative filtering architecture. International ACM SIGKDD conference on knowledge discovery and data mining
2. Ardissono L, Kobsa A, Maybury M (Eds) (2004) Personalized digital television. Targeting programs to individual users. Kluwer
3. Breese JS, Heckerman D, Kadie C (1998) Empirical analysis of predictive algorithms for collaborative filtering. Conference on uncertainty in artificial intelligence
4. Canny J. (1999) Collaborative filtering with privacy via factor analysis. Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval
5. Claypool M, Waseda M, Le P, Brow DC (2001) Implicit interest indicators. International conference on intelligent user interfaces
6. Deshpande M, Karypis G (2004) Item-based top-N recommendation algorithms. ACM Trans Inf Sys
7. Eugster PT, Guerraoui R, Kermarrec AM, Massoulie L (2004) From epidemics to distributed computing. IEEE Comput 21(4):341–374
8. Eyheramendy S, Lewis D, Madigan D (2003) On the naive bayes model for text categorization. In Proceedings of artificial intelligence and statistics
9. Fokker JE, De Ridder H (2005) Technical report on the human side of cooperating in decentralized networks. Internal report I-Share Deliverable 1.2. Delft University of Technology. http://www.cs.vu.nl/ishare/public/I-Share-D1.2.pdf
10. Hofmann T (2004) Latent semantic models for collaborative filtering. ACM Trans Inf Sys
11. Herlocker JL, Konstan JA, Borchers A, Riedl J (1999) An algorithmic framework for performing collaborative filtering. International ACM SIGIR conference on research development on information retrieval
12. Hull D (1993) Using statistical testing in the evalution of retrieval experiments. International ACM SIGIR conference on research development on information retrieval
13. Jelasity M, van Steen M (2002) Large-scale newscast computing on the internet. Internal report IR-503, Vrije Universiteit, Department of Computer Science
14. Lafferty J, Zhai C (2003) Probabilistic relevance models based on document and query generation. In: Croft WB, Lafferty J (eds) Language Modeling and information retrieval. Kluwer
15. Linden G, Smith B, York J (2003) Amazon. com recommendations: item-to-item collaborative filtering. IEEE Internet Computing
16. Marlin B (2004) Collaborative filtering: a machine learning perspective. Master's thesis, Department of Computer Science, University of Toronto
17. Miller BM, Konstan JA, Riedl J (2004) PocketLens: Toward a personal recommender system. ACM Trans Inf Sys
18. Nichols D (1998) Implicit rating and filtering. In Proceedings of 5th DELOS workshop on filtering and collaborative filtering, pp 31–36, ERCIM
19. Pennock D, Horvitz E, Lawrence S, Lee Giles C (2000) Collaborative filtering by personality diagnosis: a hybrid memory and model-based approach. UAI 2000: 437-480
20. Pouwelse JA, Garbacki P, Wang J, Bakker A, Yang J, Iosup A, Epema DHJ, Reinders MJT, van Steen M, Sips HJ (2006) Tribler: A social-based peer-to-peer system. International workshop on peer-to-peer systems (IPTPS'06)
21. Sarwar B, Karypis G, Konstan J, Riedl J (2001) Item-based collaborative filtering recommendation algorithms. International World Wide Web Conference
22 Wang J, de Vries AP, Reinders MJT (2006a) A user-item relevance model for log-based collaborative filtering. European conference on information retrieval
23. Wang J, de Vries AP, Reinders MJT (2006b) Unifying user-based and item-based collaborative filtering by similarity fusion. International ACM SIGIR conference on research development on information retrieval
24. Wang J, Pouwelse J, Lagendijk R, Reinders MJT (2006c) Distributed collaborative filtering for peer-to-peer file sharing systems. ACM Symposium on Applied Computing
25. Xue G, Lin C, Yang Q, Xi W, Zeng H, Yu Y, Chen Z (2005) Scalable collaborative filtering using cluster-based smoothing. International ACM SIGIR conference on research development on information retrieval
26. Zhai C, Lafferty J (2001) A study of smoothing methods for language models appzlied to Ad Hoc information retrieval. International ACM SIGIR conference on research development on information retrieval

Jun Wang received the B.E. degree in electrical engineering from the Southeast University, China, and the MSc degree in computer science from the National University of Singapore. He is currently pursuing the PhD degree with Faculty of Electrical Engineering, Mathematics and Computer Science (EWI), Delft University of Technology, The Netherlands. He has published over 20 research papers including IEEE Trans. on Multimedia, ACM Multimedia System Journal, ACM SIGMM and ACM SIGIR. He received the best Doctoral Consortium award in ACM SIGIR 2006. His research interests include: collaborative filtering (recommender systems), information retrieval, pattern recognition and multimedia information systems (content analysis, retrieval, and personalization).

Johan A. Pouwelse is an assistant professor at Delft University of Technology. He is the technical coordinator of a team of 16 people who are working on P2P content sharing. This team has developed the Tribler P2P system. Some time ago he conducted one of the largest easurements of the Bittorrent P2P network. This detailed measurement study ran over a period of two years and discovered many unique properties of this P2P market leader. He delivered a statement for the Federal Trade Commission in Washington, was a visiting scientist at Massachusetts Institute of Technology (MIT), and recently spent a few months at Harvard Business School to study the economic impact of movie downloads on Hollywood.

Jenneke Fokker is a PhD student at Delft University of Technology within the department of Human Information Communication Design. She holds a Master's Degree in Industrial Design Engineering, for which she worked with the design agency Bureau Mijksenaar that specializes in consultancy and the creation of visually oriented information systems. In September 2004 she started on a PhD project called Inducing Human Cooperation in Peer-to-Peer Systems. Believing that users are the decisive factor for the success of any Peer-to-Peer system, she investigates what social psychological phenomena could stimulate users to engage voluntarily and massively in specific forms of cooperative behavior.

Arjen P. de Vries received his PhD in Computer Science from the University of Twente in 1999, on the integration of content management in database systems. He is especially interested in the design of database systems that support search in multimedia digital libraries. He has worked on a variety of research topics, including (multimedia) information retrieval, database architecture, query processing, retrieval system evaluation, and ambient intelligence. Arjen works as a postdoctoral researcher at the CWI, the National Research Institute for Mathematics and Computer Science in the Netherlands. He is also associate professor in the area of multimedia data management at the Technical University of Delft.

Marcel J.T. Reinders received his MSc degree in Applied Physics and a PhD degree in Electrical Engineering from Delft University of Technology, The Netherlands, in 1990 and 1995, respectively. Recently he became a Professor in Bioinformatics within the Mediamatics Department of the Faculty of Electrical Engineering, Mathematics and Computer Science at the Delft University of Technology. The background of Prof. Reinders is within pattern recognition. Besides studying fundamental issues, he applies pattern recognition techniques to the areas of bioinformatics, computer vision and context-aware recommender systems. His special interest goes towards understanding complex systems (such as biological systems) that are severely under-sampled. Current fields of interest: Bioin-formatics, pattern recognition, computer vision, data mining, recommender systems and man-machine interfacing.