P2Prec: a Recommendation Service for P2P Content Sharing Systems

Fady Draidi, Esther Pacitti, Patrick Valduriez
INRIA and LIRMM, Montpellier, France
Fady.Draidi@lirmm.fr, Esther.Pacitti@lirmm.fr, Patrick.Valduriez@inria.fr
Bettina Kemme
McGill University, Montreal, Canada
kemme@cs.mcgill.ca

Résumé. Dans cet article, nous proposons P2Prec, un service de recommandation pour systèmes de gestion de contenus P2P, qui exploite les données sociales des utilisateurs. L'idée est de recommander à l'utilisateur des documents de qualité en utilisant les notes données par les amis (ou amis d'amis) experts du sujet. Pour gérer les données sociales, nous nous appuyons sur des descriptions FOAF. P2Prec a une architecture P2P hybride, indépendante du système de gestion de contenus P2P, qui exploite l'efficacité d'une DHT pour indexer les fichiers FOAF et la robustesse du gossip pour disséminer les sujets d'expertise entre amis. Dans notre évaluation expérimentale, avec les données de CiteSeer, nous montrons que P2Prec permet d'obtenir un rappel maximum, avec de très bonnes performances. De plus, il améliore le rappel et la précision des réponses par un facteur 2 par rapport aux solutions centralisées.

Abstract. In this paper, we propose P2Prec, a recommendation service for P2P content sharing systems that exploits users' social data. The key idea is to recommend to a user high quality documents in a specific topic using ratings of friends (or friends of friends) who are expert in that topic. To manage users' social data, we rely on Friend-Of-A-Friend (FOAF) descriptions. P2Prec has a hybrid P2P architecture to work on top of any P2P content sharing system. It combines efficient DHT indexing to manage the users' FOAF files with gossip robustness to disseminate the topics of expertise between friends. In our experimental evaluation, using the CiteSeer dataset, we show that P2Prec has the ability to get the maximum recall with very good performance. Furthermore, it increases recall and precision by a factor of 2 compared with centralized solutions.

Keywords: P2P content sharing, recommendation service, social network, DHT, gossiping.

1 Introduction

The popularity of P2P content sharing systems such as BitTorrent [7] and eMule [12] has translated into large amounts of documents being spread over high numbers of peers (and users). Although more information is available to more users, users tend to get overwhelmed with high numbers of documents returned as results of their queries. And it is hard for them to distinguish which are the most valuable and relevant documents. In addition, popular P2P content sharing systems such as eMule only provide a very simple keyword search capability, trying to find the documents whose name or description match the keywords provided by the user.

A solution to this problem is to group the peers with similar contents into the same clusters on top of an unstructured overlay, typically using information retrieval (IR) techniques, e.g. [4]. Then, a user query is flooded within the peers in the user's cluster. But flooding consumes much network bandwidth. This solution makes sense only if we can generally assume that a query initiated at a user peer relates to the content stored at that peer. Furthermore, this solution does not help the user distinguishing which of the returned documents have high quality and high value. This is because the users themselves, i.e. their interest in specific topics or their rankings of documents they have read, is simply ignored. In other words, what is missing in P2P content sharing systems is a recommendation service (RS) which can recommend high quality and valuable documents using user information.

In our daily life, we make a choice between alternatives based on opinions and advices that we have received from other resources such as people we know (friends, family members, etc.), general surveys, travel guides, published reviews, experts and so on. In order to enable people share their opinions, advices and benefit from each other's experience without human intervention, recommendation services (RSs) have emerged. RSs (also called recommender systems) work by suggesting documents or information items (i.e. products such as movies, documents, Web pages, CDs, books, etc.) of interest to users according to their preferences [37]. Basically, RSs analyze users' historical patterns (ratings, purchasing, preferences, etc.) to find and recommend new products that the user might be interested in and like to explore or purchase. In general, an RS collects a user's historical patterns, then finds other users with similar patterns or products in the historical patterns, and finally uses the data from those similar users or products to suggest products the user might be interested in.

In this paper, we propose *P2Prec*, an RS for P2P content-sharing systems that exploits users' social data. To manage users' social data, we rely on the Friend-Of-A-Friend (FOAF) project [45]. FOAF offers an open, detailed description of profiles of users and the relationships between them using a machine-readable syntax. Whenever a user[1] generates its FOAF file, it can obtain an identity for that file on the Web in the form of a URI. This URI could point to a reference in the user's FOAF file stored in a server that the user trusts. In this sense, FOAF becomes an important tool to provide simple directory services and one can use information from FOAF files to locate people. One can imagine FOAF as a way of describing a distributed directed graph of friendship relations, where each user specifies its interests, topics of expertise and friends in its FOAF file, and then stores it in a server that its trusts.

P2Prec'general goal is to improve the quality and efficiency of query responses in P2P content sharing systems, by exploring the synergy between RSs and the social relations between users. P2Prec is useful to recommend to a user high quality documents related to a specific topic from documents that have been seen or created and rated by friends (or friends of friends) which are expert in that topic. As an example, let us consider a user new to a topic, e.g., vegetarian, who would like to get good and valuable documents related to the topic vegetarian cuisine. Then the user should be able to consult its friends (or friends of friends) which have some expertise or experience in vegetarian cuisine so those users can return back high quality documents related to that topic. Thus, a key capability of P2Prec is to help users in a P2P content sharing system finding high quality documents in one or more topics from friends (of friends) which are experts in those topics. Users' topics of expertise are automatically calculated based on a combination of topic extraction from their documents (the documents they shares) and rating. To extract and classify the hidden topics available in the documents, we use the Latent Dirichlet Allocation (LDA) technique [8]. Without loss of generality, users which do not have expert friends, which we call *isolated-users*[2], still have the ability to use the system to get high quality recommendations.

P2Prec has a hybrid P2P architecture to work on top of any P2P content sharing system. It combines efficient DHT indexing to manage users which are expert in one or more topics along with their FOAF files with gossip robustness to disseminate the topics of expertise between friends (of friends). This hybrid architecture has two layers: expertise and recommendation. The *expertise layer* organizes the users of the P2P content sharing system which are expert in a Distributed Hash Table (DHT) [24] to efficiently find them and access their FOAF files. The recommendation layer is an unstructured overlay that implements a gossip-based protocol to let each user maintain an up-to-date view of the topics of expertise of its friends and a subset of their friends of friends for the purpose of generating recommendations.

---

[1] For simplicity, we use the term *user* for both the person and "software on the behalf of the user" (thus, we use words like "it" instead of "she", "which" instead of "who", etc.).

[2] For simplicity of presentation, we will start by assuming that each user has at least one expert friend. We will introduce the solution for the *isolated-users* in Section 4.3

In this paper, we make three main contributions. First, we propose a hybrid architecture for P2Prec to work on top of any P2P content sharing system, managing users' social data and topics of expertise. Second, we propose an efficient query routing algorithm for the unstructured overlay and introduce a novel method to rank the returned results by combining documents' popularity with their semantics. Finally, we provide an experimental evaluation that demonstrates the efficiency of P2Prec over the CiteSeer dataset [21].

The rest of this paper is organized as follows. Section 2 introduces some background that we use for P2Prec. In Section 3, we give an overview of P2Prec, with the main terms and assumptions used in the paper. Section 4 describes the design of P2Prec. In Section 5, we describe our algorithm for query routing. Section 6 provides an experimental evaluation of P2Prec. In Section 7, we discuss related work. Section 8 concludes.

2 Background

P2Prec uses FOAF files to manage users' social profiles and LDA for automatic topic extraction, and exploits P2P networks. In this section, we briefly introduce these concepts.

2.1 FOAF Files

FOAF [45] provides a simple, machine-readable vocabulary serialized in RDF/XML to describe people, content objects and the connections that bind them all together. A FOAF file is typically created by the individual user and published on a server that the user trusts. Over the last few years, FOAF has become increasingly popular and used in many different projects. A lot of public FOAF data are now available on the Web and can be indexed and managed by many tools, e.g. Google's Social Graph API, Yahoo's SearchMonkey, OpenLink's Virtuoso, etc. Some Web social networks such as LiveJournal[27] have adopted FOAF to automatically manage user profiles. With a FOAF file, a user can start describing herself using the foaf:Person class, listing attributes such as name, address and interests and use foaf:knows to describe its friends, listing attributes such as name and see also. Whenever a user generates its FOAF file, it stores its FOA file in a host server that it trusts and obtains an identity for the file on the Web in the form of a URI from that host server. Overall, the FOAF vocabulary is simple and can be integrated with any other semantic Web vocabularies. Users can easily create their own FOAF file and link to it from their homepage, which has encouraged its adoption and make it suitable to a wide range of uses. Consider the following code example with a simple XML serialization for the FOAF (partial) description of a person.

<foaf:Person><foaf:name>Jean</foaf:name>
<foaf:interest> P2P and Social Networks</interest>
<foaf:knows><rdfs: seeAlso rdf: resource = "http://www.lirmm.fr/Peter.rdf"/>
</foaf:knows> </foaf:Person>

This example shows that user Jean is interested in "P2P and Social Networks" and knows a friend whose FOAF file is at "http://www.lirmm.fr/Peter.rdf".

2.2 LDA Topic Extraction

We need a technique to extract and classify the hidden topics available in the documents that will be used to define the users' topics of expertise. Classifying the hidden topics available in a set of documents is an interesting problem by itself. Several models have been proposed, described and analyzed in the IR literature [11] to tackle this problem. The one we use is Latent Dirichlet Allocation (LDA) [8]. LDA is a topic classifier model that represents each document as a mixture of topics and models each topic as a probability distribution over the set of words in the document. For example, a document talking about *vegetarian cuisine* is likely to be generated from the mixture of words from the topics *food* and *cooking*.

LDA assumes a fixed finite number of topics. Hierarchical Dirichlet Processes (DP) [44] has been proposed as an extension to the standard LDA [8] to adapt the number of topics automatically but is more complex. In P2Prec, we use the standard LDA [8] and thus use a fixed finite number of topics. We choose LDA because it is efficient in clustering high dimensional and sparse data and it has ability to solve synonymy (different words with identical meaning) and polysemy (same word with multiple meanings). But we could easily extend P2Prec to use Hierachical DP.

A document doc is a series of words, $doc=\{word_1,...,word_n\}$, where $word_i$ is the i^{th} word in doc and n is the total number of words in doc. D is a set of m documents denoted by $D=\{doc_1,.,doc_m\}$, where doc_i is the i^{th} document in D. In LDA, a document doc is represented as a vector of d dimensions, called document topic vector, $V_{doc}=[w_{doc}^{t_1}....w_{doc}^{t_d}]$, where d is the number of topics, T is the set of topics in the system and $w_{doc}^{t_i}$ is the weight which topic $t_i \in T$ has received in doc. The set of document topic vectors of all documents in D is denoted by θ which is a matrix of dimensions $m*d$. Each topic $t_i \in T$ is modeled as a probability distribution over the words in the documents with word $word_j$ having a weight $\phi_{t_i}^{word_j}$ in topic t_i. That is, each topic $t_i \in T$ is modeled over all the words in D. Thus, the probability distribution of all topics T over words in D can be represented as a matrix ϕ with dimensions $d*z$, where z is the total number of the unique words in D.

As in any classifier, we need to train LDA over a sample of documents to estimate parameters θ and ϕ, which can then be used to predict the topics in new documents. Several algorithms have been proposed to estimate LDA parameters θ and ϕ, e.g. a variation Expectation-Maximization algorithm [8] and Gibbs sampling [18]. Gibbs sampling is a special case of the Markov-chain Monte Carlo (MCMC) algorithm [15] to a proximate the joint distribution or to compute an expected value. It used to generate a sequence of samples from the joint probability distribution of two or more random variables. The joint probability is the probability of events defined in terms of the random variables. In Gibbs a sample set of documents is sufficient to estimate the parameters θ and ϕ. In P2P systems, we cannot easily collect all the documents in the network because the users may join and leave the network at anytime. Thus, we use Gibbs sampling to estimate θ and ϕ from a sample of documents in the P2P system. After the LDA parameters are extracted, we use them to identify θ and ϕ of the new documents. The task of the parameter estimation is given to a distinguished peer p in the P2P content sharing system. It aggregates a sample of documents from the peers in the system and estimates θ and ϕ, and then provides them to the peers in the network which in turn can use this parameter to compute the θ and ϕ of their documents i.e., the topics of their documents. Peer p runs in the background providing passive support and computes the parameters at regular intervals of time to acquire the changes in the documents corpus in the system.

2.3 P2P Networks

P2P networks can be classified according to their overlay topology between *unstructured* and *structured*. Typically they differ on the constraints imposed on how users are organized and where shared contents are placed [36]. Unstructured networks impose few constraints on users' neighborhood and content placement [36] so that users in the overlay get loosely connected. Unstructured networks typically use flooding [36], gossiping [14] or random walk [36] protocols to disseminate discovery messages or queries. With flooding, a user sends a query to all its neighbors. With random walk, a user sends a query to a randomly-selected subset of its neighbors. Gossip-based protocols have attracted a lot interest for building and managing unstructured networks. With gossip, each user periodically exchanges its state (a user's state might be its shared date or documents, a set of other contacts, etc.) called *view*, with another randomly-selected user. Thus, after a while, as with gossiping in real life, each user will have a partial view of what other users in the system know.

On the other hand, structured networks exploit a distributed data structure such as a tree or hash table for efficient query routing among peers. The most popular form of structured network is DHT, which implements distributed hash table functionality [36]. A DHT assigns each user a unique id (e.g., by hashing the user's IP address). Based on these identifiers, a DHT maps a given key k (an object identifier) onto a user u using a hash function and can lookup u efficiently, usually in $O(\log n)$

routing hops where n is the number of users. DHTs typically provide two basic operations: *put(k, data)* that stores a key k and its associated data and *get(k)* that retrieves the data associated with k in the DHT.

3 Overview of P2Prec

In this section, we give a short overview of P2PRec, with the main terms and assumptions used in the paper.

P2Prec can return to a user in a P2P content sharing system documents that have been reviewed and rated by friends (of friends) who are expert in the documents' topic, thus increasing the quality of query responses as well as the user's confidence in the returned documents. Sinha et al. [43] have shown that users prefer the advices that come from known friends (friends, family members, colleagues) in terms of quality, confidence and usefulness. We model a P2P content sharing system as a graph $G = (D,U,E,T)$, where D is the set of shared documents, U is the set of users in the system, E is the set of edges between the users such that there is an edge $e(u,v)$ if users u and v are friends, and T is the set of users' topics of expertise. Each user $u \in U$ is associated with a set of topics of expertise $T_u \subset T$, so $t \in T_u$ indicates a topic t for which user u is an expert. The cardinality of U is denoted by $|U|$ and the cardinality of T is denoted by $|T|$.

We assume that each user $u \in U$ stores and maintains locally (on its peer) a set $D_u \subset D$ of documents that it has rated, each of its rates over a document $doc \in D_u$ being denoted by $rate^u_{doc}$. The cardinality of D_u is denoted by $|D_u|$. The user's rate over a document $doc \in D_u$ can be either explicit or implicit [38]. The system may ask the user to explicitly give a numeric rate for doc. On the other hand, the user's rate over $doc \in D_u$ might be extracted by monitoring implicitly its behavior over doc, e.g., the time the user spends in reading doc, how many times the user browses doc, etc. The user ratings either explicit or implicit are often represented by discrete values within a certain range, e.g., between 1 and 5.

Each user also maintains a FOAF file which contains a description of its personal data such as its topics of expertise $T_u \subset T$ and links to its friends denoted by $friends(u) = \{f_1, f_2 \ldots\ldots f_n\}$, where n is the number of friends of user u.

P2Prec lets each user extract locally its topics of expertise from the documents it maintains. A user $u \in U$ becomes expert in topics of expertise $T_u \subset T$ if it has rated at least a number x of documents (where x is system defined) in D_u, i.e., $|Du| \geq x$. Recall that u has defined the topics in its documents D_u using LDA. Then, the computation of the topics of expertise T_u is based on the topics and rates associated with D_u. Once the user u has extracted it topics of expertise T_u, it records its T_u in its FOAF file (see more details in Section 4.1).

The idea behind P2Prec is to let each user periodically exchange (gossip) its topics of expertise if it has and the topics of expertise of some of its friends (of friends) with some of its friends. Thus, each user continuously maintains a partial view of the topics of expertise of the users in the system. Consequently, a user u posing or receiving a keyword query q, uses its view to find potential expert users that might have high quality documents related to q. LDA is used to extract the topics from the query q keywords.

In order to let a user u gossip with its friends to construct and maintain their views, user u should know which of its friends are expert, what are their topics of expertise and IP addresses. To accomplish this, P2Prec has an expertise layer which organizes expert users in a DHT. An expert user that enters P2Prec joins this layer and publishes its FOAF file over the DHT using the hash result of the URI of its FOAF file as key as well as its id. Whenever a user uses P2Prec, it can retrieve its friends' FOAF files from the DHT using the hash results of the URIs of its friends' FOAF files as keys. Consequently, it knows which of its friends are embedded in the expertise layer (i.e., expert), what are their topics of expertise and IP addresses. Then, it can select any one of those friends to interact with. Basically, the expertise layer replaces the random peer sampling protocol (RPS) [23] and provides fast lookup of the expert users. In order to generate recommendations, an unstructured recommendation layer on top of the expertise layer is constructed by using gossip

protocols. Once a user has retrieved the FOAF file of its friends, periodically it exchanges with a randomly-selected friend their topics of expertise and the topics of expertise of some of their friends (of friends). Thus, each user u eventually gets links to its friends and some of their friends (of friends) where it uses them to route and serve its queries.

In summary, P2Prec relies on a hybrid architecture with two layers as illustrated in Figure 1. The expertise layer manages expert users along with their FOAF files using a DHT. It is also used for peer sampling by the gossip protocol of the second layer and provides reliable lookup of the expert users. The recommendation layer maintains and manages the users' views and serves the users queries.

In the example of Figure 1, P2Prec covers two topics t_1 (circle shape) and t_2 (rectangle shape) and 8 users, each user being either an expert at least in one topic (circle and rectangle shapes) or non expert in any topic (triangle shape) e.g., u_1 is expert in t_1 while u_7 is not expert in any topic. Expert users u_1 to u_6 are embedded in the expertise layer (DHT). In the recommendation layer each user u_1 to u_8 knows the topics of expertise of its contacts, e.g., u_1 knows that u_4 and u_5 are experts in t_1 while u_6 is expert in t_2.

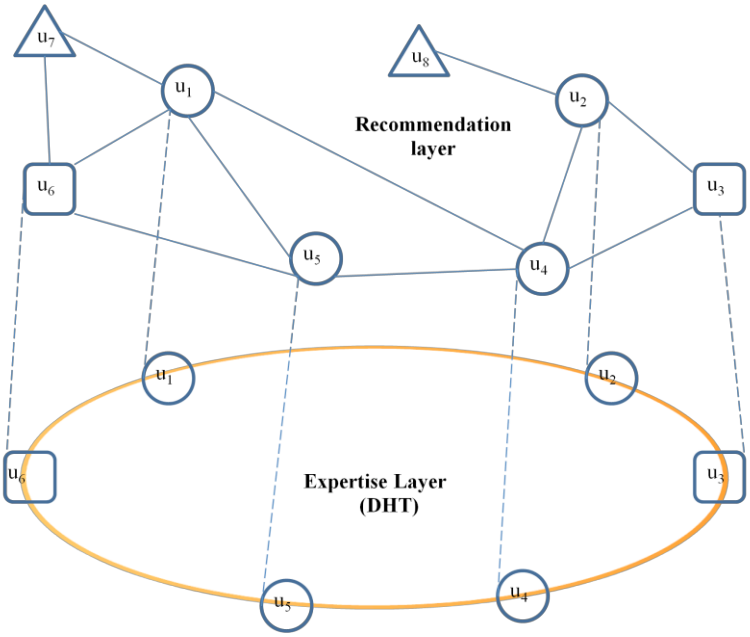


Figure 1. P2Prec architecture with two topics t_1 and t_2

4 P2Prec Design

In this section, we describe the design of the two main layers of P2Prec. Before, we introduce the way we compute users' topics of expertise.

4.1 Extracting Users' Topics of Expertise

Each user u which has $|D_u|$ exceeds x locally computes its topics of expertise $T_u \subset T$, in two steps. First, it computes the document quality for each document doc it has rated and records it locally in a vector denoted by $quality(doc,u)$. This is done by multiplying the document topic vector $V_{doc}=[w_{doc}^{t_1} \ldots w_{doc}^{t_d}]$ that has been extracted using LDA, by the rate $rate_{doc}^{u}$ that has been given by user u over doc. Thus, we have:

$$quality(doc,u) = [w_{doc}^{t_1} * rate_{doc}^{u} \ldots\ldots w_{doc}^{t_d} * rate_{doc}^{u}]$$

Then, user u extracts for each topic $t \in T$ only the documents that have high quality in that topic t. A document doc is considered a high quality document in a topic t if its weight in that topic w_{doc}^{t} multiplied by its rate $rate_{doc}^{u}$ exceeds a threshold value (which is system defined). We can consider that a document is high quality in a topic t if its weight in that topic w_{doc}^{t} exceeds a threshold value.

But we choose to combine it with the user's rate to exploit the user's feedback. In this case, the document quality is computed based on the user opinion and the document semantics. That is, a *doc* with high weight in a topic t will not be considered high quality if it has received a low rate from the user.

In the second step, user u computes its topics of expertise $T_u \subset T$ where u is considered an expert in topic $t \in T_u$ if a percentage of its documents have high quality in that topic t. Finally, u records its topics of expertise $T_u \subset T$ in its FOAF file, which contains *friends(u)* and its topics of expertise $T_u \subset T$. In addition, it records locally the documents belonging to its topics of expertise $T_u \subset T$ in its shared area. Later on, u will use these documents to answer queries. User u has the ability to download and rate the document recommendations it receives, and add or delete documents. Thus, its topics of expertise might be changed. To capture this dynamic behavior, user u computes its topics of expertise T_u at every fixed period of time, or if a number of documents have been added to (or deleted from) its D_u and exceeds a system-defined threshold. Figure 2 shows how a user u can extract its topics of expertise $T_u \subset T$ and then update its FOAF file.

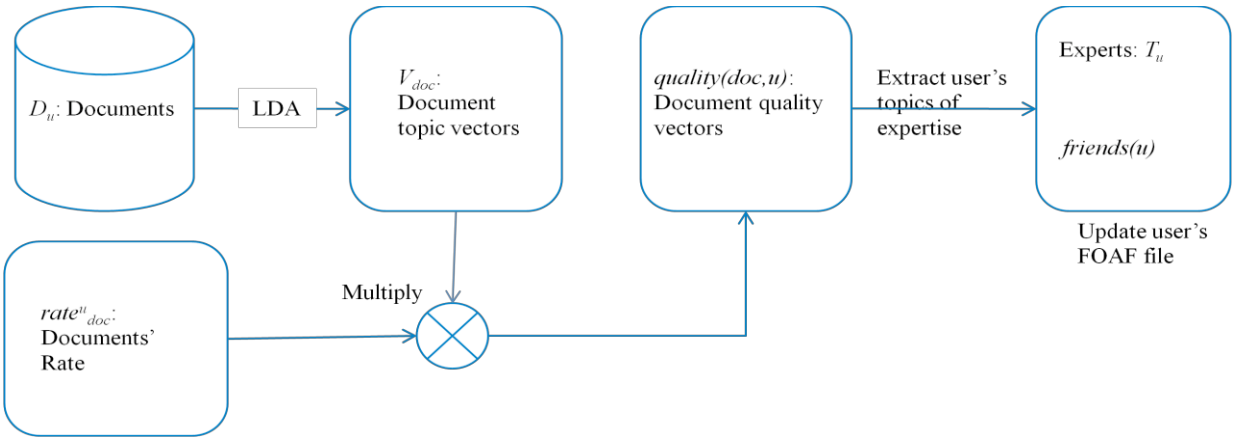


Figure 2. How a user u locally computes its topics of expertise $T_u \subset T$

4.2 Expertise Layer

The *expertise layer* organizes the expert users of the P2P content sharing system in a DHT and is responsible for managing their FOAF files. It is also in charge of letting each user know which of its friends are expert and what are their IP addresses and their topics of expertise. The expertise layer uses a DHT to provide fast lookup of expert user and their FOAF files, based on the URI of their FOAF files.

A user u joins the expertise layer, if it has become expert in a set of topics $T_u \subset T$. Recall that user u is considered expert in T_u if its $|D_u|$ exceeds x and a percentage of its D_u are high quality in T_u. Thus, a user u which has joined this layer is expert in T_u and stores high quality documents related to T_u. The expert user u which joins the expertise layer uses the hash result of the URI of its FOAF file as its id in the DHT as well as the key for its FOAF file. Thus, when it publishes its FOAF file in the DHT, it keeps it on its peer. And whenever a user leaves the expertise layer, its FOAF is removed automatically. This reduces the overhead maintenance of the DHT.

Each P2Prec participant user u either expert or non expert periodically retrieves with a *get* operation the FOAF files of its friends along with their IP addresses. The IP address and the FOAF information, such as topics of expertise, can then be used in the recommendation layer. Let us illustrate with the example of Figure 3, which shows an expertise layer with six users (users' ids are the subscript numbers attached to their names). Let us consider that user u_7 uses P2Prec and assume that u_1 is one of its friends. First, u_7 forwards *get*(1) to the DHT where the value 1 of k is obtained by hashing the URI of u_1's FOAF file. Then, the DHT routes u_7's request through u_5 and u_3 to u_1 which holds k.

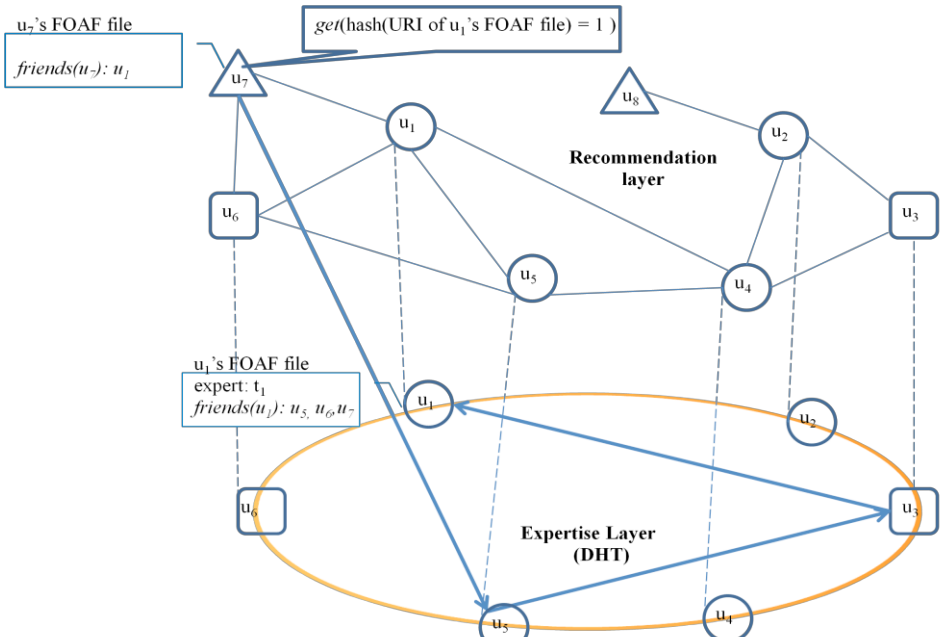


Figure 3. How a user retrieves its friends FOAF file from the expertise layer

4.3 Recommendation Layer

The *recommendation layer* is the unstructured overlay produced from the periodic gossiping between a user u and a user v randomly selected among its expert friends (a friend in the expertise layer i.e., expert at least in one topic). Each user u has a local state called *summary* that is a set of user descriptors, each descriptor referring either to an expert friend or an expert friend of friend. A user descriptor contains the IP address of the user, along with its topics of expertise.

We now describe in more details the construction of the recommendation layer and the gossip algorithm. The recommendation layer is constructed and maintained by filling the summaries at all participant users with descriptors of expert friends (of friends). Each participant user periodically gossips with its expert friends to exchange local summaries. Thus, after each gossip round, a user gets a full view of the topics of expertise of its expert friends and a partial view of the topics of expertise of a subset of their expert friends of friends. Thus, when the user initiates a query, it can select potential expert users from its local summary and route the query to them.

Users use gossip-style communication to construct the recommendation layer and exchange their summaries in an epidemic manner [19]. Users also gossip to discover newly joined expert friends and to detect failed ones. We choose gossip-style communication for the following reasons. First, the continuous exchange of summaries between users enables the building of an unstructured overlay network in a continuous manner that reflects the natural dynamism of P2P networks and helps provide very good connectivity in the presence of failures or peer disconnections [14]. Second, it provides a reliable way to disseminate information in large-scale dynamic networks, so that users discover new expert friends and their expert friends' of friends along with their topic of expertise [25]. Third, a gossip-style communication ensures load balancing during the disseminating of information between users, since all users have the same number of gossip targets and the same exchange period, and thus send exactly the same number of messages [14]. Finally, gossip is scalable, reliable, efficient and easy to deploy [22].

Basically, gossip proceeds as follows: A user u (either expert or non expert) acquires its initial summary when it retrieves its friends' FOAF files from the expertise layer. The initial summary contains at first the descriptors of its expert friends only. Then, periodically (with a gossip period noted T_{gossip}), every user u exchanges its local state information (summary) with one of its expert friends. Whenever a user u initiates an information exchange with an expert friend v obtained randomly from its expert friends, it sends a subset of its local summary to v. In turn, its expert friend

v which receives an exchange message replies with a subset of its local summary to the initiator. Finally, user *u* updates its local summary based on the messages received during the information exchange. The update process merges the received messages with its current summary, then extracts the descriptors of its expert friends and a subset of their expert friends' of friends which have the highest value of topics of expertise.

The gossip behavior of each user *u* is illustrated in Algorithm 1. The active behavior describes how a user *u* initiates a periodic gossip exchange message, while the passive behavior shows how the user *u* reacts to a gossip exchange initiated by some other friend *v*. Non expert users perform the active behavior part only while expert users perform the active and passive behaviors. Non expert users perform the active behavior only because they will not be selected by their friends to gossip with (since they are not expert, they are not be included in their friends' local summaries).

The active behavior is executed every time unit T_{gossip}. A user *u* initiates a communication, selects: (1) an expert friend *v* from its expert friend descriptors using the SelectRandom() method and (2) a subset of expert users noted *viewSubset* from its local summary, a random subset of L_{gossip} summary descriptors using the SelectView() method. Then, user *u* sends to expert friend *v* *GossipMsg*, a message that contains the descriptor of the expert users in *viewSubset*. In turn, user *u* will receive a gossip message *GossipMsg** that contains *viewSubset** of the expert friend *v* *local* summary. A call to the merge() method returns in a buffer the merge of user *u's* local summary and the information in *GossipMsg**. Then, it uses the extract() method to select from the buffer the descriptors of its expert friends and a subset of their expert friends of friends which have the highest topics of expertise values.

In the passive behavior, user *u* waits for a gossip message *GossipMsg* from a friend *v* either expert or non expert. Upon receiving a message, it replies by sending back a gossip message containing descriptors of a subset of its local summary and updates its local summary using the merge() and extract() methods as described previously. Through the exchange messages that take place by the passive and active behavior, each user *u* will have a full view of the state of its expert friends and a partial view of the state of their expert friends' of friends.

Algorithm 1 - Gossip behavior of user *u*

```
// active behavior
Loop
   wait(T_gossip)
   expert friend v ← friends.SelectRandom()
   viewSubset ← summary.SelectView()
   GossipMsg ← <viewSubset>
   send GossipMsg to expert friend v
   receive GossipMsg* from expert friend v
   buffer ← merge(summary, GossipMsg*)
   summary ← buffer.extract()
end loop
// passive behavior
loop
   waitGossipMessage()
   receive GossipMsg from friend v
   viewSubset ← summary.SelectView()
   GossipMsg* ← <viewSubset>
   send GossipMsg* to friend v
   buffer ← merge(summary, GossipMsg)
   summary ← buffer.extract()
end loop
```

In order to let *isolated-users* benefit from the system, we register each expert user which has joined the expertise layer at a bootstrap server. Once an *isolated-user* has joined in the system, it periodically contacts an expert user randomly-selected from the bootstrap server. Then it asks that expert user to send its *viewSubset* of its local summary. Recall that each user periodically retrieves the FOAF files of its friends from the expertise layer, thus whenever the *isolated-user* finds an expert friends, it replaces the entries in its local summary with the entry of its expert friend and starts gossiping with that expert friend.

Figure 4 shows how the users are connected in the recommendation layer. A solid line represents a link between friends, a dashed line represents a link between a user and friend of friend, a dot line represents a link between expert user and *isolated-user* while the arrows mean the directions of connections. For example u_1 has links to its friends u_5 and u_6 and a link to u_4 which is a friend of its friend u_5. While u_3 is an *isolated-user* and has links to expert users u_2 and u_4 but u_2 and u_4 have not links to u_3 i.e., they do not include it in their local summaries.

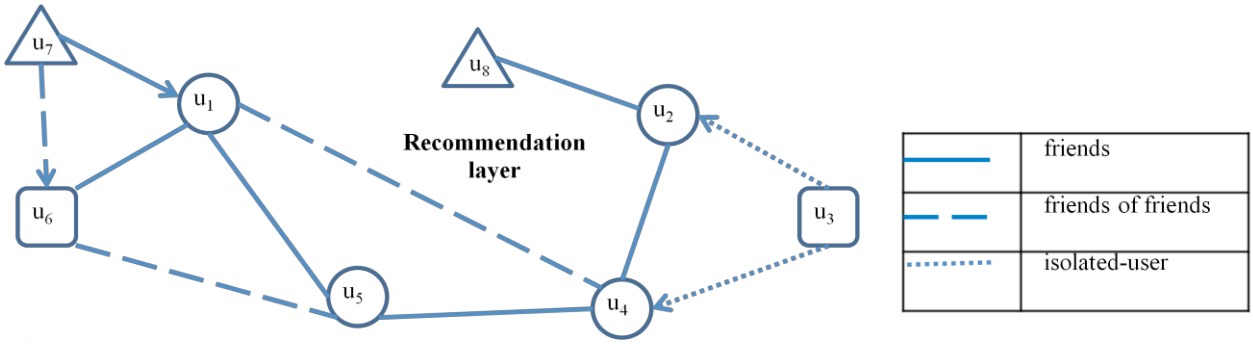


Figure. 4. A snapshot of the recommendation layer

5 Query Routing and Result Ranking

In this section, we describe the query routing algorithm that we use to generate recommendations as well as the ranking model we use to order the returned results

We assume keyword queries of the form $q = \{word_1, word_2, ..., word_l\}$, where l is the number of keywords in the query and $word_i$ is the i^{th} keyword in q. Query q can be of type *push* or *pull*. In the push type, the system automatically extracts the keywords of the query q from the documents that are belonging to the user's topics of expertise, such as the most frequent words in the document. In the pull type, user u issues a query q with keywords. For both types, the system extracts q's topic vector, denoted by $V_q = [w_q^{t_1} ....w_q^{t_d}]$, using LDA as we did for a document. Then query topic(s) $T_q \subset T$ are extracted. The query q is considered belonging to a topic $t \in T_q$ if its weight w_q^t in that topic exceeds a certain threshold (which is system defined). The cardinality of query q's topics is denoted by $|T_q|$.

Based on this assumption, each query q issued by a user u consists of the topics $T_q \subset T$ of its keywords, the keywords of q and a Time to Live (TTL) limit for the query. Whenever a user u issues a query q with topic(s) $T_q \subset T$, it divides q into $|T_q|$ sub queries. Each subquery q' consists of the keywords of q, TTL of q and one topic $t \in T_q$. Then it routes each subquery q' as follows. First, u selects from its local summary the users which are expert in topic $t \in T_q$ and then forwards q' to them after reducing the query TTL by one. Each user v which receives query q' processes it as shown in Algorithm 2. First, v selects from its local summary the users which are expert in topic $t \in T_q$ and forwards the query to them while the query TTL does not reach zero. Second, user v returns to the initiator the documents it has on topic $t \in T_q$ that are most similar to q. The similarity between a document *doc* and a query q, denoted by *sim(doc,q)*, is measured by using the cosine similarity [39] between the document topic vector $V_{doc} = [w_{doc}^{t_1} ....w_{doc}^{t_d}]$ and the query topic vector $V_q = [[w_q^{t_1} ....w_q^{t_d}]$ which is:

$$sim(doc,q) = \frac{\sum_{i=1}^{d} w_q^{t_i} * w_{doc}^{t_i}}{\sqrt[2]{\sum_{i=1}^{d} w_q^{t_i} * w_q^{t_i} * \sum_{i=1}^{d} w_{doc}^{t_i} * w_{doc}^{t_i}}}$$

Finally, user v returns to the initiator the documents whose similarity exceeds a given (system-defined) threshold. We use q instead of q' in the similarity because the keywords of each subquery q' is the keywords of q and thus the topic vector of each subquery q' is the same as the topic vector of q

---

Algorithm 2 – Routing of query q and processing at user u

```
//query routing
receive q from u'
if q.TTL > 0 then
   q.TTL ← q.TTL -1
   for ( ∀ v ∈ summary ) do
      if ( q.t ∈ T_v ) then
         send q' to v
      end if
   end for
end if
//query processing
receive q from u'
for (∀ doc ∈ D_u and doc is high quality in q.t ) do
   sim(doc,q) ← sim(doc,q)
   if sim(doc,q) ≥ threshold then
      return doc to v
   end if
end for
```

In the example of Figure 5, suppose that user u_1 initiates a query q for topic t_1 with TTL=2. User u_1 sends the query to u_4 and u_5 after reducing TTL by 1. When u_5 receives q its TTL is 1, then u_5 returns to u_1 the answer of q (the documents that are high quality in t_1 and are most similar to q) and stops forwarding q even though its TTL is not zero. This is because u_5 does not have a user in its summary that is expert in t_1 and does not receive a copy of q yet. u_4 receives q with TTL equal to 1; it returns the answer of q to u_1 and forwards q to u_2 after reducing TTL by one. u_2 receives q with TTL equal to zero; it returns the answer of q to u_1 and stops forwarding q because its TTL has reached zero.

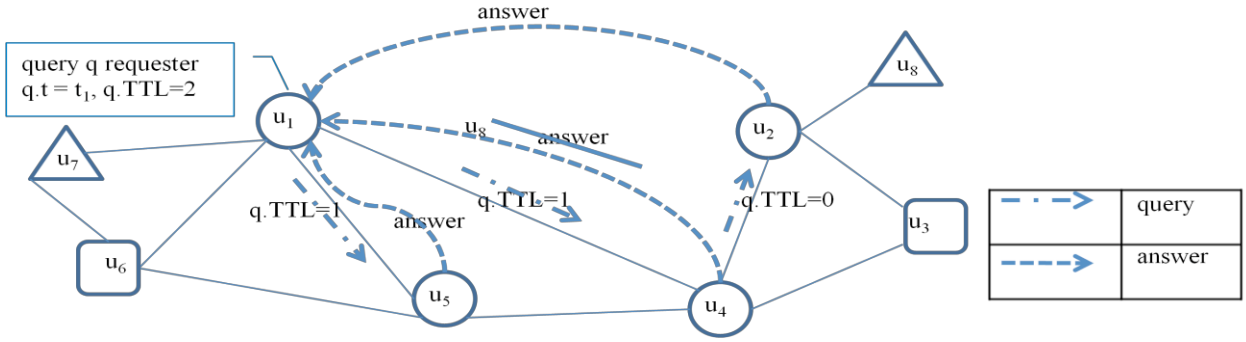


Figure 5. An example of query routing

With such query routing, we avoid sending q to all users' neighbors, thus minimizing the number of messages and network traffic for q. Furthermore, the returned documents are from experts friends (of friends), which increases the confidence of the user in those documents.

When an initiator receives the recommended documents from the responders, it merges all documents that come as a response for each subquery q'. Then it orders them based on their popularity and semantics. That is, the rank of a document *doc*, denoted by *rank(doc)*, consists of its semantic relevance with the query q and its popularity:

$$rank(doc) = a * sim(doc, q) + b * pop(doc)$$

where a and b are scale parameters such that $a + b = 1$ and *pop(doc)* is the popularity of the document *doc* which is equal to the number of responders that have returned document *doc*. The user can specify whether it prefers the highly popular documents or the highly semantically relevant by playing with parameters a and b. Upon receiving recommendation documents, a user u can downloading a copy of a document, gives a rate to that document and includes it in its highly quality documents (i.e., the user u shares it with others) if it becomes belonging to one of its topics of expertise T_u.

6 Experimental Evaluation

In this section, we provide an experimental evaluation of P2Prec to assess the quality of recommendations, search efficiency (cost and speed) and bandwidth consumption. We use the CiteSeer dataset [21] and a combination of implementation and simulation. We first describe the experimentation setup and next the experiments. Finally, we compare P2Prec's performance against that of a centralized version of the CF RSs [9].

6.1 Experimentation Setup

We use the classical metrics *precision* and *recall* that are used in IR and RSs to assess the quality of the returned results [41]. Precision represents the system ability to return documents that are mostly relevant to a query from the dataset. Recall represents the system ability to return all relevant documents to a query from the dataset. Thus, in order to measure recall and precision, the relevant documents set for each query that have been issued in the system should be known in advance. Typically, in order to do that, machine learning cross-validation techniques, e.g., hold-out or k-folding, are applied to the high quality documents of each user and then evaluation metrics run upon. We divide the high quality documents of each user u into two sets, testset, denoted by S_u, and training set. A user u's training set is given to the system during the experiments and the user u uses them to answer other users' queries. On the other hand, u's testset S_u is kept hidden during the experiments and is used to measure the quality of u's queries. Thus, user u uses P2Prec to possibly retrieve the documents that have been in its testset S_u. The set of documents returned by P2Prec for a user u is denoted by P_u. Once a user u has received P_u from P2Prec, it can count the number of common documents in both sets P_u and S_u to compute recall and precision. Thus, recall is defined as the percentage of testset documents $doc \in S_u$ occurring in P_u with respect to the overall number of testset documents $|S_u|$:

$$Recall = 100 . \frac{|P_u \cap S_u|}{|S_u|}$$

Accordingly, precision is defined as the percentage of testset documents $doc \in S_u$ occurring in P_u with respect to the size of P_u:

$$Precision = 100 . \frac{|P_u \cap S_u|}{|P_u|}$$

We use the following metrics to evaluate P2Prec.

- Communication cost: the number of messages in the P2P system for a query;
- Response time: the average time needed to answer a query from all users who receive the query;
- Hit ratio: the percentage of the number of queries that have been successfully submitted.
- Background traffic: the average traffic in bps experienced by a user due to gossip exchanges.

We evaluated P2Prec over the CiteSeer dataset [21] which contains over 400 thousand authors and over 1,2 million documents. We consider each author as a user and the articles he/she authored as the documents. We extracted the social network in CiteSeer data from the authors' cooperation. That is, two authors are considered friends if they co-authored at least one article. The co-authorship graph has been shown to emulate a social network [31]. We use the article's title and abstract only. We keep all the users that have at least 20 documents and have at least one friend. We choose the users with at least 20 documents to possibly extract topics of expertise for each user and apply the hold-out technique over the user document set. As a result 13585 users are left with 112906 links and an average of 8.3 friends per user. And these users have 626244, with an average of 46.1 documents per user.

We extracted the titles and the abstracts of these documents and removed from them all the stop words (e.g., the, and, she, he, …) and punctuations. Then, we fed them to the GibbsLDA++ software [34], a C++ implementation of LDA using Gibbs sampling, to estimate the parameters θ and ϕ. We use $|T|$=100 as the number of topics and we ran GibbsLDA++ 2000 times to estimate the parameters θ and ϕ.

We generate a random rate between 0 and 5 for each document a user has and compute the users' topics of expertise from the documents they have rated. We also keep for each user the documents related to its topics of expertise. In the simulation, we consider that each user is expert in one topic and downloads the documents it receives, thus without changing its expertise topic during the experiments for ease of explanation and simulation.

P2Prec is built on top of a P2P content sharing system which we generated as an underlying network of 13585 nodes which is equal to the number of users left in the dataset. This network is built on top of the internet where nodes are connected with links with variable latencies bounded between 10 and 500 ms [10]. Since the expertise layer is modeled as a DHT, we choose to simulate Chord for its simplicity, using PeerSim [32]. Each experiment is run for 24 hours, which are mapped to simulation time units.

In order to evaluate the quality of recommendations, we let each user issue a query after receiving the results from all the users that have received the previous query or after the query has reached a system-specified timeout, the topic of the query being the same as the topic of expertise of that user and thus the query has one topic. Then we obtain the recommendations for each query and compute recall, precision, communication cost and response time. In order to obtain global metrics, we average the respective metric values for all evaluated queries. Furthermore, we perform five different runs for each experiment, each time with a different random rate and partitioning into training and test to ensure that our results are statistically accurate. The results reported are the averages over these five trials.

In order to evaluate the maximum possible recall under these partitions, we perform an exhaustive search for each user in the dataset. The search is performed by flooding the user's query to all users in the network. Then, we obtain the recommendations and compute the recall metric. The maximum possible recall achieved under this setting is 87.7%. Because there might be documents which were included in the testset that are not hold (e.g., explored and rated) by any other user in the network, these documents will never be obtained during the exhaustive search and thus the recall never reaches 100%.

6.1.1 Experiments

We perform two types of experiments: (1) under static network, i.e., the size of the network is not changed during the run; (2) under churn i.e., the network size is changed during the run due to the joining and leaving of users.

In the first type experiments (static network), we investigate the effect of gossip and TTL on the quality of recommendations over the respective metrics. The experiments are done by varying the gossip parameters: gossip message size L_{gossip}, gossip time T_{gossip}, the number of friends of friends, noted Ff, and the TTL of the query. In each experiment, we vary one of the parameters (L_{gossip}, T_{gossip}, Ff, TTL) and fix the three other parameters. Then, we collect the results for each parameter after 24 simulation hours. We do not show all the results due to space limitations but we explain our observations.

Decreasing T_{gossip} increases the speed of reaching the maximum recall. It takes 1 hour to reach a recall of 87% when T_{gossip} is 1 minute and 15 hours to reach the same recall when we increase T_{gossip} to 1 hour. But decreasing T_{gossip} increases the bandwidth consumed by the users, because gossip exchanges are less spaced and thus more frequent. The bandwidth consumed by a user is multiplied by 60 when decreasing T_{gossip} from 1 hour to 1 minute.

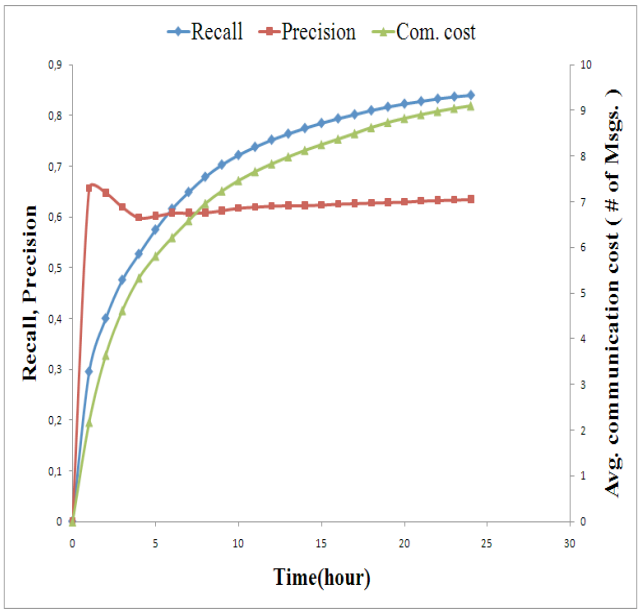
Increasing L_{gossip} increases the bandwidth consumed by a user due to gossiping. When L_{gossip} increases from 5 to 20, the bandwidth of a user is increased by a factor of 4. When L_{gossip} increases, more entries are carried out in the gossip message, thus, increasing message size and bandwidth.

The number of expert friends of friends Ff involved in gossiping does not have any impact over the performance of the system. This is due to the nature of the data set since the documents are shared between the direct friends and thus friends' of friends might not hold those documents. Therefore, visiting them by a query does not affect recall but slightly reduces precision because some irrelevant documents might be returned from these users.

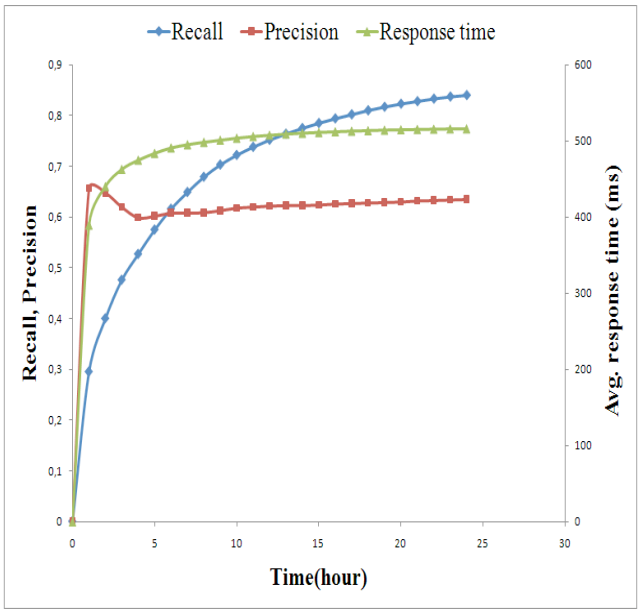
The query TTL variation has significant impact on precision, response time and communication cost but it has no significant impact on the recall. When increasing TTL, more users are visited, thus increasing communication cost and waiting time. The communication cost is multiplied by 9 when TTL increases from 1 to 3. But the increase in TTL decreases precision, because when more users are visited more documents are returned back. Some of these documents might be irrelevant due to the nature of the dataset where usually relevant documents to a query of a user are hold by its direct friends. The precision decreases by a factor 2 when TTL increases from 1 to 3.

The second type of experiments (with churn) starts with a stable expertise layer with 500 users. Then as experiments are run, new users are joining and some of the existing users are leaving. We do the same tests as in the first type experiments. We observe that the results of the variation of gossip parameters and TTL under churn are the same as those under static network.

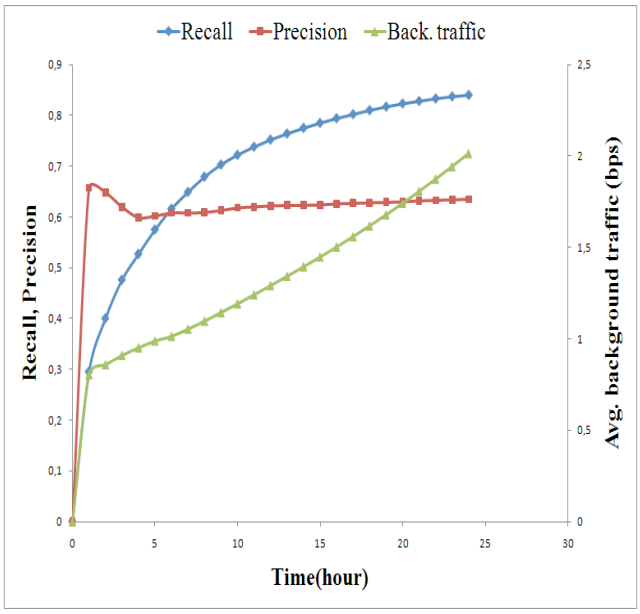
In Figure 6, we show the variation of the respective metrics versus time when T_{gossip} = 30 minutes, L_{gossip} = 10, Ff = 10 and TTL =1. Figure 6(a) shows that the recall almost stabilizes after 17 hours at 85%. At the beginning, the network size is small and many users' expert friends are not alive. Thus many irrelevant documents are returned, which reduces recall. Precision starts at 66% and then decreases until stabilizing at 61% after 11 hours. Initially, a user might have a few numbers of expert users in its summary. As gossip is used, more expert users are added in the user's summary. Thus, more expert users might be visited and thus more irrelevant documents might be returned. The communication cost keeps on increasing as the network size increases, because more contacts become available in the user local summary as the network size increases. Figure 6(b) shows that there is no difference between the response time under churn or under fixed size network. The background traffic keeps on increasing as the network size increases as shown in Figure 6(c). As the network size increases, more expert friends of the user are alive and L_{gossip} reaches its upper bound (10) more rapidly. Figure 6(d) shows the variation of hit ratio. We observe that the hit ratio starts at 0.676 and keeps on growing as the network size grows and becomes stable at 0.98 after 17 hours. Note that, at the beginning, a user peer may not have live contacts in its local summary satisfying its query and thus is not able to forward the query.



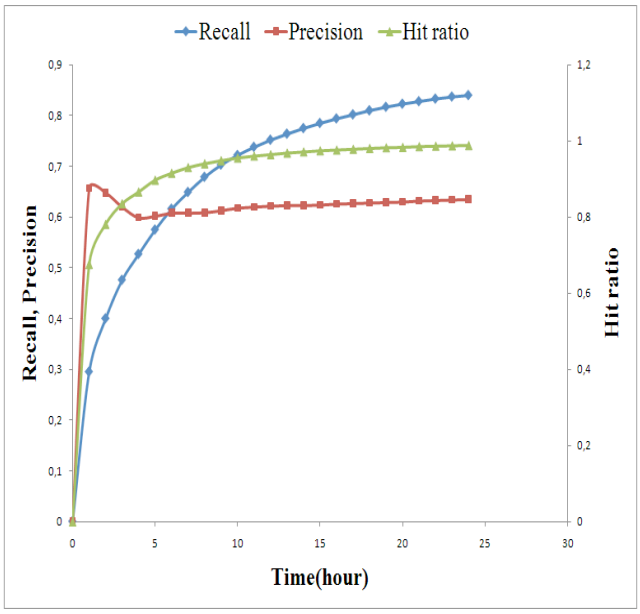
(a) Trade off between recall, precision and cost



(b) Trade off between recall, precision and response time



(c) Trade off between recall, precision and background traffic



(d) Trade off between recall, precision and hit ratio

Figure. 6. Variation of the respective metrics versus time

6.1.2 Social Effect

We compare P2Prec with a centralized CF RS [9], where all users and their rated documents are stored in a central server which also performs all the recommendation processes. In order to recommend a set of documents that a user *u* may like, we compute the similarity between a user *u* and all the users in the system. Then, we select a set of users, noted *neighbors(u)*, which have a similarity with *u* that exceeds a threshold value. We use the cosine similarity to extract a user's *neighbors*, based on the rates that are given by the users over the documents they have seen or created. Thus, the similarity between a user *u* and another user *v*, noted *sim(u,v)*, is computed by making the product between the rates that have been given by *u* and *v* over their documents:

$$sim(u,v) = \frac{\sum_{i=1}^{m} rate^{u}_{doc_i*} rate^{v}_{doc_i}}{\sqrt{\sum_{i=1}^{m} rate^{u}_{doc_i*} rate^{u}_{doc_i} * \sum_{i=1}^{m} rate^{v}_{doc_i*} rate^{v}_{doc_i}}}$$

Once the similarity between *u* and *v* has been computed, *v* is considered a neighbor to *u* if the *sim(u,v)* exceeds the system-defined threshold, which we call *threshold similarity*, i.e., *sim(u,v)* ≥ *threshold similarity*. Once the *neighbors(u)* are extracted, we select for user *u* the documents that have been rated by its *neighbors* but which user *u* did not explore yet (interested readers can refer to [9]). For the purpose of evaluation, we divide each user's documents into training and testset as explained previously. Then, we run the system and generate recommendations for each user from its *neighbors* and then compute the average recall and precision of all users. We observe that the similarity measure is time consuming as it takes about 25 minutes to compute the similarity between the 13585 users with 375858 documents. This time increases exponentially as the numbers of documents and users increase.

The experiments are done by increasing the *threshold similarity* from 0.1 to 0.9 and we compute the recall, precision and hit ratio under each *threshold similarity* value. Figure 7 shows the recall, precision and hit ratio obtained with different *threshold similarity* values. The figure shows that the recall, precision and hit ratio decrease as the *threshold similarity* increases. As the *threshold similarity* decreases, more *neighbors* get selected for a user which increases the hit ratio and thus increases the possibility of finding the documents that the user has in its testset which in turn increases recall and precision. However, P2Prec increases the recall and precision by a factor of 2 even for the lowest *threshold similarity* in comparison with the centralized CF.

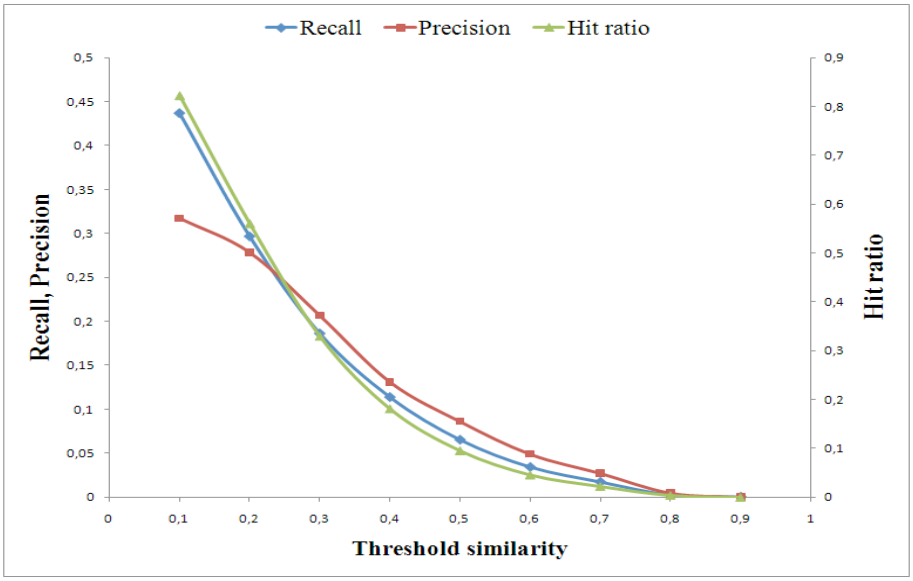


Figure. 7. Variation of the respective metrics with threshold similarity

7 Related Work

P2Prec is a hybrid P2P RS designed to recommend high quality documents to users in a P2P content sharing system by exploiting their social information. In this section, we discuss the work most related to ours which includes centralized RSs, distributed RSs, social P2P networks as well as multiple overlay networks.

Centralized RSs. Since the first work carried out by Goldberg et al [16], RSs have been used in major applications such as e-commerce, e.g. Amazon.com and CDNow [41]. RSs typically work by recommending to users products (items, documents, movies, books, newsletter, CDs, etc.) based on what the user was interested in previous times. There are two kinds of RSs: collaborative filtering (CF) [16] or content-based [28]. CF can recommend products to a user based on the products

previously rated by similar users. It works by measuring the similarity between users based on the ratings that have been given by the users over the products [37]. It typically works with three generic steps: (1) measure the similarity between the potential user (the user asking for recommendations) and all users in the system; (2) select those users who are most similar to the potential user; (3) normalize and compute the weighted sum of the selected users' ratings, then make suggestions based on those ratings. CF has been widely used for building RSs, e.g. in GroupLens [37], Ringo/Firefly [42], Tapestry RS [16] and Recommender [20]. However, CF suffers from data sparsity, i.e. users rate small numbers of products in the system, which leads to several problems. First, a user might have rated little numbers of products and thus the user might not find similar users. Second, a new user which has not rated any products yet will not find similar users to help in finding recommendations [2,48]. In contrast, P2Prec relies on the explicit friendships between users, which dramatically reduces the possibility that a user will not find friends because it does not have rated enough products yet.

Content-based filtering has been introduced to alleviate CF from the data sparsity problem [1,5,9]. Content-based RSs work by suggesting to the user products that are similar to products that the user has seen or rated [9]. In these systems, the similarity measure is computed between the products the user has seen and the nominated products. Products with high similarity are suggested to the user. However, a user is limited to receive products that are only similar to the products it has rated and thus might not explore new interesting topics. In P2Prec, each user maintains a list of friends (of friends) with different topics of expertise so a user has the possibility to search variety of topics even though it is not interest or expert in those topics. All the approaches described above also rely on the client-server model which has scalability problems.

Distributed RSs. Recently, there has been little work on distributed RSs. The first work is Tveit [46], a P2P RS to suggest recommendations for mobile customers. The system is based on a pure P2P topology like Gnutella and queries that include users' rates are simply propagated by flooding. Miller et al [30] explore P2P RS and propose four architectures including random discovery similar to Gnutella, transitive traversal, DHT, and secure blackboard. In these architectures, a user aggregates the rating data from the system to make recommendations. Peng et al. [33] propose to store users' rates over a DHT in order to distribute users' rates. All users which have rated a product with same rate are grouped in one cluster, called *bucket*. These buckets are spread over the DHT. A user aggregates all buckets of its products to make recommendations. However, there are two main drawbacks. First, aggregating users' rates to make recommendations for a user increases the amount of traffic within the network. In contrast, P2Prec lets each user maintain locally a set of users which are expert in a set of topics, so the user uses them to support its query. Thus, a P2Prec user does not need to aggregate users' rates to make recommendations, thus reduces network traffic. Second, these systems distribute users' rates independently of any semantic or social structure of the network, while P2Prec is structured based on users' expertise (documents' semantics) and friendships (users' social).

Social P2P networks. Recent work [3,13,47] exploits the social relations between users to self-organize, manage users' information and enhance search in the network. Bai et al. [3] design a personalize P2P top-k search for collaborative tagging systems. Each user u maintains locally a profile which includes the products that it has tagged. In addition, it maintains a *personal network* of users with similar interest along with their profiles. Two users are considered similar if they share a common number of tagged products. In order to find and construct a user *personal network*, two gossip protocols are used. The first is used as peer sampling to keep the overlay connected. The second is used to gossip users' profiles and measure the similarity between them based on their profiles. Once the user has determined its *personal network*, it stores locally their profiles. When a user issues a query, it uses the profiles of its *personal network* members to process locally its query. In [13], a P2P RS for scientific papers, called Papeer, is proposed. As in [3], each user builds its own *personal network* by using two gossip protocols and keeps locally their profiles. Papeer gives users the ability to tag and comment the papers they have explored. This information might be used to measure the similarity of interest between users, find papers with similar such information or recommend users which might be interested in such information. When a user searches for a paper, it

uses its *personal network* to find that paper. Also the system gives each user the ability to find which users in its *personal network* might be interested in a given paper. Similar to Papeer, Wang et al. [47] propose a P2P RS for television systems on top of Tribler [35]. Each user maintains locally a set of top most similar users (*buddies*) and a set of random peers along with their profiles. Whenever a user selects another user to contact, it first merges the *buddies* with the random peer and ranks them based on the similarity between their profiles with its profile (the similarity between two profiles is measured by counting how many common files they have). Then one user is randomly selected according to a roulette wheel approach. This gives more chance for more similar user to be selected and gives a chance for new user to be explored.

These systems have several problems. First, making users maintain locally the profiles of their *neighbors* (either a user's *personal network* or a user's *buddies*) leads to storage and inconsistency problems. In P2Prec, each user maintains only its documents, thus eliminating this problem. Second, users construct their *neighbors* by gossiping their profiles, which may yield high network bandwidth consumption. In contrast, P2Prec only uses gossip to exchange the topics of expertise between users and the topics of expertise are small (represented by sets of numbers). Finally, these systems are based on the implicit relationships between users and thus might deteriorate users' confidence over the recommendations. P2Prec relies on the explicit friendships, which increases users' confidence over the recommendations.

Multiple overlay networks. In [29], the authors explore the idea of adapting multiple overlay networks cross peers, so the deployment of an overlay can utilize the information in the other to reduce maintenance costs. In particular, they propose a hybrid of DHT and unstructured overlay, using the entries of the users' routing tables in the DHT for peer sampling for the gossip protocols in the unstructured overlay. Their architecture has some similarity to ours as it uses DHT and gossip. But in P2Prec, we use the DHT to provide fast lookup for the users' social data and we use the friendships between users as the peer sampling of the gossip protocols.

8 Conclusion

In this paper, we proposed P2Prec, an RS for P2P content-sharing systems that exploits users' social data. P2Prec is useful to recommend to a user high quality documents related to a specific topic from documents that have been seen or created and rated by friends (or friends of friends) who are expert in that topic. To manage users' social data, we use a FOAF file for each user. Each user in the system is automatically assigned topics of expertise, based on a combination of topic extraction from its documents (the documents she shares) and rating. To extract and classify the hidden topics available in the documents, we use the LDA technique. P2Prec has a hybrid P2P architecture to work on top of any P2P content sharing system. It combines efficient DHT indexing to manage the users' FOAF files with gossip robustness to disseminate the topics of expertise between friends.

In our experimental evaluation, using the Citeseer dataset, we showed that P2Prec has the ability to get the maximum recall with acceptable query processing load and network traffic. From our experimentations, we can make two main observations. First, the usage of users' social information (friendship and topics of expertise) improves the RS in many ways. First, exploiting the friendships between users removes the time needed to compute the similarity between users while the system still has the ability to reach the maximum recall and acceptable precision. In contrast, centralized collaborative filtering consumes a lot of time in measuring the similarity between users to select users' *neighbors* and leads to low recall, low precision as well as low hit ratio. Second, exploiting the friends' topics of expertise reduces the query communication cost and response time because we avoid sending the query to users irrelevant to the query while preserving the ability of the system to get maximum recall and good precision. Third, using gossip style communication to exchange the topics of expertise between friends increases the system ability to get to the maximum recall with acceptable precision and low overhead in terms of bandwidth consumption. Furthermore, it increases the query hit ratio because gossiping brings more live contacts in a user local summary, thus reducing the possibility that the user does not find users satisfying its query.

References

1. Aggarwal, C., Wolf, J., Wu, K., Yu, P., Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 201–212, 1999.
2. Adomavicius, G., Tuzhilin, A., Toward the next generation of recommender systems: a survey of the state-of-theart and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734–749, 2005.
3. Bai, X., Bertier, M., Guerraoui, R., Kermarrec, A. M., Toward personalized peer-to-peer top-k processing. In: *International workshop on Social Network Systems*, 2009.
4. Bawa, M., Manku, G. S., Raghavan, P., SETS: Search Enhanced by Topic Segmentation. *ACM SIGIR Conf.*, 306–313, 2003.
5. Billsus, D. Pazzani, M. J., Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning*. Morgan Kaufmann, 46–54, 1998.
6. Birman, K. P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y., Bimodal Multicast. *ACM Trans. on Computer Systems*, 17(2):41–88, 1999.
7. BitTorrent P2P File Sharing. http://www.bittorrent.com/index.html.
8. Blei, D. M., Ng, A. Y., Jordan, M. I., Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
9. Breese, J.S., Hecherman, D., Kadie, C., Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 43–52, 1998.
10. Brite project. http://www.cs.bu.edu/brite.
11. Callan, J., Distributed Information Retrieval. In W. B. Croft, editor, Advances in Information Retrieval,127-150, Kluwer Academic Publishers, 2000
12. eMule project. http://www.emule project.net.
13. Frey, D., Kermarrec, A.-M., Leroy, V., PAPEER: Bringing Social Networks into Research. http://www.gossple.fr, 2009.
14. Gavidia, D., Voulgaris, S., Steen, M., Cyclon: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and System Management*, 13(2), 2005.
15. Gilks, W. R., Richardson, S., Spiegelhalter, D. J., Markov Chain Monte Carlo in Practice. Chapman & Hall, New York, 1996.
16. Goldberg, D., Nichols, D., Oki, B., Terry, D., Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*. 35(12):61–70, 1992.
17. Good, N., Schafer, B., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., Riedl, J., Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 1999 Conference of the American Association of Artificial Intelligence*,1999.
18. Griffiths, T.L., Steyvers, M., Finding Scientific Topics. *Proceedings of the National Academy of Science,* 101:5228-5235, 2004.
19. Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Demers, A., Greene, D., Terry, D.: Epidemic Algorithms for Replicated Database Maintenance. *ACM Symp. on Principles of Distributed Computing (PODC),* 1-12, 1987.
20. Hill, W., Stead, L., Rosenstein, M. Furnas, G., Recommending and evaluating choices in a virtual community of use. *Conference Proceedings on Human Factors in Computing Systems*, 194-201, 1995.
21. http://copper.ist.psu.edu/oai/oai_citeseer
22. Jelasity, M., Montresor, A., Epidemic-style Proactive Aggregation in Large Overlay Networks. *IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 102-109, 2004.
23. Jelasity. M., Voulgaris, S., Guerraoui, R., Kermarrec, A. M., VanSteen, M., Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.

24. Karger, D., Kaashoek, F., Stoica, I., Morris, R., Balakrishnan, H., Chord: A scalable P2P lookup service for internet applications. *In SIGCOMM*, 2001
25. Kermarrec, A.M., Eugster, P.T., Guerraoui, R., Massoulieacute, L., Epidemic Information Dissemination in Distributed Systems. *IEEE Computer*, 37(5): 60-67, 2004.
26. Konstan, J.A., Riedl, J., Borchers, A. Herlocker, J.L., Recommender Systems: A GroupLens Perspective. In: *Recommender Systems: Papers from the 1998 Workshop*, 60-64, 1998.
27. LiveJournal social network. http://livejournal.com
28. Malone, T., Grant, K., Turbak, F., Brobst, S., Cohen, M., Intelligent Information-sharing Systems. *Communications of the ACM*, 30(2):390–402, 1987.
29. Maniymaran, B., Bertier, M., Kermarrec, A.-K., Build one, get one free: Leveraging the co existence of multiple p2p overlay networks. *In: Proc of the 27th International Conference on Distributed Computing Systems (ICDCS)*, 2007
30. Miller, B.N., Konstan, J.A., Riedl, J.: PocketLens, Toward a Personal Recommender System. *ACM Trans. on Information Systems*, 22(3):437-476, 2004.
31. Newman, M. E. J., The Structure of Scientific Collaboration Networks. *Proceedings of the National Academy of Sciences*. 98:404 – 409, 2001.
32. Peersim p2p simulator. http://www.peersim.sourceforge.net
33. Peng, H., Bo, X., Fan, Y., Ruimin, S., A scalable p2p recommender system based on distributed collaborative filtering. *Expert systems with applications*, 2004.
34. Phan, X.-H., http://gibbslda.sourceforge.net
35. Pouwelse, J.A., Garbacki, P., Wang, J., Bakker, A.,Yang, J., Iosup, a., Epema, D.H.J., Reinders, M., Vansteen, M.R., Sips, H.J., TRIBLER: a social-based peer-to-peer system. *In IPTPS'06, Santa Barbara*, 2006.
36. Qiao, Y., Bustamante, F.E., Structured and unstructured overlays under the microscope: a measurement-based view of two P2P systems that people use. *In Proceedings of the USENIX Annual Technical Conference (ATEC)*, 341–355, 2006.
37. Resnick, P., Iakovou, N., Sushak, M., Bergstrom, P., Riedl, J., GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Computer Supported Cooperative Work Conf.*, 175-186 , 1994.
38. Ruthven, I., Lalmas, M., A Survey on the Use of Relevance Feedback for Information Access Systems. *The Knowledge Engineering Review*, 18, 95-145,2003
39. Salton, G., A Theory of Indexing. *Regional Conf. Series in Appl. Math., Soc. For Indust. And Appl. Math.*, Philadelphia, Pennsylvania, 1975.
40. Sarwar, B., Konstan, J., Borchers, A., Herlocker, J.,Miller, B., Riedl, J., Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. *In Proceedings of the 1998 Conference on Computer Supported Cooperative Work*, 1998.
41. Sarwar, B., Karypis, G., Konstan, J., Riedl, J., Analysis of Recommendation Algorithms for e-commerce. *ACM Conf. on Electronic Commerce*, 158–167, 2000.
42. Shardanand, U., Maes, P., Social Information Filtering: Algorithms for Automating "Word of Mouth". *Conference Proceedings on Human Factors in Computing Systems*, 210-217, 1995.
43. Sinha, R., Swearingen, K., Comparing Recommendation made by Online Systems and Friends. *In the Proc. Of the DELOS-NSF Workshop on Personalization and Recommender Systems in Digital Libraries*, Ireland, 2001
44. Teh, Y., Jordan, M., Beal, M., Blei, D., Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 2006. 101(476),1566-1581,2006
45. The friend of a friend (FOAf) project. http://www.foaf-project.org.
46. Tveit, A., Peer-to-Peer Based Recommendations for Mobile Commerce. *Proceedings of the International Workshop on Mobile Commerce*, 26–29, 2001.
47. Wang, J., Pouwelse, J.A., Fokker, J., Reinders, M.J.T., Personalization of apeer-to-peer television system. *In Proc.of Euro ITV2006*, Athens, 2006.
48. Yu, K., Schwaighofer, A., Tresp, V., Xu, X., Kriegel, H.-P., Probabilistic memory-based collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering*, 16(1), 56–69, 2004.