

# Optimizing Peer-to-Peer Keyword Search using Collaborative Filtering

Niels Zeilemaker



Delft University of Technology



# Optimizing Peer-to-Peer Keyword Search using Collaborative Filtering

Research Assignment Computer Science

Parallel and Distributed Systems group  
Faculty of Electrical Engineering, Mathematics, and Computer Science  
Delft University of Technology

Niels Zeilemaker  
1315250  
niels@zeilemaker.nl

16th April 2009



# Preface

In preparation of my Master of Science project in the Parallel and Distributed Systems Group of the Delft University of Technology a literature study was required. This report is the result and lists the current active state of development in the field.

I would like to thank dr. ir. J.A. Pouwelse and ir. M. Clements for their advice regarding both content and layout of this document.

Niels Zeilemaker

Delft, The Netherlands  
16th April 2009



# Contents

<b>Preface</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Problem Description</b>	<b>5</b>
2.1 Existing Solutions . . . . .	5
2.2 Research Question . . . . .	8
<b>3 Similarity</b>	<b>11</b>
3.1 Memory-based . . . . .	12
3.1.1 Pearson’s correlation . . . . .	12
3.1.2 Cosine similarity . . . . .	13
3.1.3 Differences . . . . .	14
3.1.4 Improving . . . . .	14
3.1.5 Item-Item . . . . .	16
3.2 Model-based . . . . .	17
3.3 Hybrid/Combined . . . . .	19
3.4 Log-based . . . . .	20
<b>4 Spam and Duplicates</b>	<b>23</b>
<b>5 Testing Methodologies</b>	<b>29</b>
5.1 Datasets . . . . .	29
5.2 Evaluation . . . . .	30
<b>6 Conclusions and Future Work</b>	<b>33</b>



# Chapter 1

## Introduction

Peer-to-Peer (P2P) technology was introduced in 1979, when Usenet systems started to appear. Usenet is a system in which servers exchange information with each other in a non-centralized way. This approach deviates from the more traditional client-server type network, in which a central component has more responsibilities such as routing, processing etc. A P2P network only has peers (computers/devices connected to the network), which all basically have the same role. The peers together strive to complete a task, which results in a network without a centralized component. P2P networks are most known in their data exchanging variant. But P2P technology has surfaced in other fields as well, some examples are searching for extraterrestrial intelligence (SETI) or making calls possible between computers (Skype).

Because a P2P network does not have a centralized component some tasks are harder to achieve. A common problem is the bootstrapping problem, which occurs when you try to join a P2P network. Because of the dynamic nature of a P2P network and the lack of centralized components, you cannot know which peers are connected to the P2P network. And thus which peer to contact in order to join the network. This problem is usually solved by including a list of known peers into the installer, or using several peers which are always online and allow new peers to connect to the network.

But the lack of central components makes a P2P network also very scalable. When more and more clients connect to a traditional client-server network, the capacity of the server must increase accordingly, otherwise the performance of the network will suffer. In a P2P network this problem does not occur because all peers together are responsible for this task. More peers joining the network will mean more peers responsible for dividing the load and thus the network will scale relatively easy. Another benefit of not having a central component is the elimination of a single point of failure. When the server in a client-server network goes down, the whole network goes down. In a P2P network all peers are equal, thus the loss of one peer does not influence the performance of the network much.

These benefits have caused P2P traffic to dominate the overall Internet traffic.

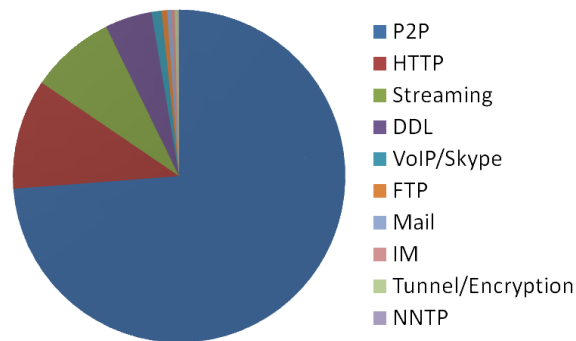


Figure 1.1: Traffic Distribution in Germany 2007, as described by Ipoque[23]

According to a survey conducted by Ipoque in 2007 [23] 69% of all traffic in Germany is P2P traffic (Shown in Figure 1.1). Comparing this number to only 10% for http traffic shows us the acceptance of P2P technology. More recent Ipoque released the results of a survey conducted in 2008<sup>1</sup>. This showed that overall P2P traffic has decreased to 53% and http traffic increased to 26%, caused mainly by the increase of popularity of sites like YouTube and Rapidshare. But still 53% of all traffic is a very large percentage indeed.

## Overlays

P2P networks are constructed as an additional layer on top of a preexisting network, this additional layer is called an overlay. P2P overlays can be organized in different ways, structured and unstructured.

In a structured overlay all peers connect to the network in an organized way. An example of a structured overlay is a dynamic hash table (DHT), this is a ring-based overlay in which all peers have a unique id which is generated by a hash function (the input is usually the ip-address). The id places a peer on the ring. Messages are routed using a routing table which contains id-ip mappings and then forwarded to the nearest peer until the destination is reached. A structured overlay could also be based on a tree. These overlays are particularly useful for streaming, because information can be poured in at the root and easily distributed along its children. Structured overlays have one big problem and that is dealing with churn. Churn is a problem caused by the constant leaving and joining of new peers and the organizational complexity of dealing with this. A P2P network under churn usually has a lot of overhead as a result. In structured overlays this overhead becomes unacceptably big when a network scales (increases in size).

In an unstructured overlay peers are connected in a random way. They maintain

<sup>1</sup>[http://www.ipoque.com/resources/internet-studies/internet-study-2008\\_2009](http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009)

a list of neighbors which is used to route messages. Sending a message to a specific peer has to be done using flooding (sending a message to all neighbors which is then again sent to all neighbors etc.) because there is no correlation between a peer and its location, this is a downside of a unstructured overlay. But because of the unstructured nature of the overlay dealing churn results less overhead.

This literature study will focus on file sharing P2P networks. A lot of controversy surrounds these networks because they allow users to exchange files. Often these files are backups of copyrighted material and are illegal to distribute.

One of the first P2P networks which allowed users to exchange files was Napster. Napster was released in 1999 and allowed users to search and download files. When a user wanted to search for a file in the Napster network, a call was made to a central server. This central component causes Napster to not be a true P2P network. A true P2P network being a network in which all peers are equal (the central server was not equal to all other peers). Eventually a lawsuit against Napster resulted in a \$26 million settlement and Napster was quickly shutdown.

The central component inside the Napster network, the central search server, allowed for an easy shutdown. But having a central server for searches is very efficient. Because Napster had an unstructured overlay, searching in the network would otherwise be done using flooding. Future networks needed to find ways to allow users to search efficiently, but without this central component. The question is then how to improve search efficiently without these centralized components. This is discussed in Chapter 2.



## Chapter 2

# Problem Description

A necessity for file sharing is locating a file, because only after locating a file, a user can download it. Different strategies have been developed for doing so, which all have their upsides and downsides. Chapter 1 introduced the problem of searching in a P2P network with a central component. This chapter lists the current state of development in P2P file searching.

### 2.1 Existing Solutions

How do these strategies work and what kind of optimizations do they employ in order to improve the search efficiency? In order to evaluate the performance of the existing solutions the benchmarks as introduced by Lin et al.[13] are used:

- Query Efficiency ( $QE$ ), which is the  $QueryHits/QueryMsg/NetworkSize$  and indicates the number of messages needed in order to get results. Higher is better.
- Search Responsiveness ( $SR$ ), which is the  $SuccessRate/HopsNumber$  and indicates the speed of a result. Lower is better.
- Search Efficiency ( $SE$ ), which is the  $QE \cdot SR$  and indicates the overall performance of a search technique. Higher is better.

We start by describing Gnutella.

#### Gnutella

Gnutella <sup>1</sup> was released in early 2000 and in late 2007 approximately 1.8 million peers were connected to it. In contrast to Napster, Gnutella did not rely on a central component to perform searches. Instead a message was sent to all neighbors (all other peers a peer is directly connected to) containing the search-term.

---

<sup>1</sup><http://en.wikipedia.org/wiki/Gnutella>

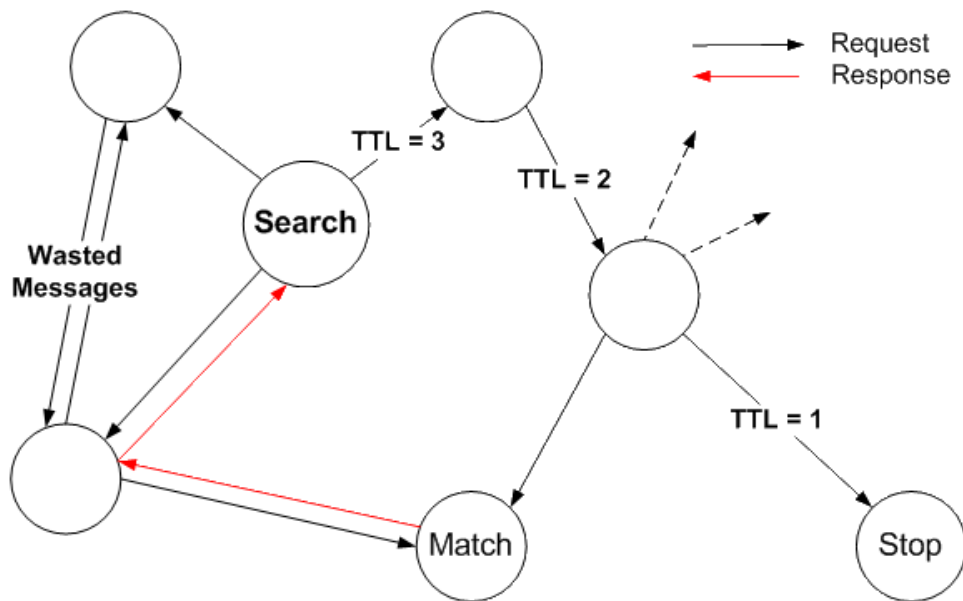


Figure 2.1: Gnutella search, including TTL, a Loop and a Query Response

These messages are called Query messages. They contain an id, the sender, a Time-To-Live (TTL, number of resends allowed) and the specified search string. Query messages are broadcasted to all neighbors (Figure 2.1) until the TTL equals 0. In the first versions of the Gnutella network the TTL was 7, but eventually this number was reduced to 4. Upon receiving a Query message, the search string is matched against a set of local filenames. When a file was found a Query Response is back-propagated, as described by Ripeanu[18].

The search algorithm is a typical flooding variant. It sends messages to all neighbors and thus floods the network with Query messages. This results in a low  $QE$ , because when using a TTL of 7 almost all peers will receive the Query message (95% according to Ripeanu[18]). Additionally the problem is even worse because some messages are sent multiple times to the same peers, a peer notices this but only after already receiving the message. This is illustrated in Figure 2.1, where the two peers on the left send each other messages that are not needed. Because of the random structure of an unstructured overlay this will happen. Flooding will result in a high  $SR$ , because the number of hops between the current peer and the peer with the file is at most the TTL. Combining these results results in a  $SE$  which ends up to be low, which is supported by Ripeanu[18] which reports that 92% of all messages in the network are Query messages.

As a result Risson et al.[19] measured that in 2000, when the Gnutella network had only 50000 connected peers, Query messages alone consumed 1.7% of the total US Internet backbone traffic. Another problem that occurs with flooding is when this many Query messages occur in a network it will become unscalable. Less capable peers will saturate their Internet connection with only Query messages,

which in turn leads to a degradation of the search performance (because not all messages are forwarded by the saturated peers, resulting in parts of the network which are not reached).

## **Kazaa**

The Kazaa<sup>2</sup> program uses the FastTrack protocol. Introduced in 2001 it added the use of superpeers which help to improve *SE*. These more powerful peers, which are chosen by the program itself based on number of available resources a peer has, are used to index the network. These indexes are used to produce search results. When a normal peer searches the network, it sends a message to the superpeers it is connected to. Then this message is flooded to all other superpeers (of which there are less) and results are returned. The use of superpeers is a quick fix, because eventually when the network grows more and more the original problem of saturated peers will occur again. Only this time at the less capable superpeers, this is suggested by Sarshar[20]. Kazaa is also famous for its settlement of \$100 million with 4 big record companies. In December of 2003 the FastTrack network peaked at 4.8 million concurrent users.

## **eMule**

In 2002 eMule<sup>3</sup> was released. It connected to two different networks, the eDonkey and Kad network. For searching in the eDonkey network it relied on central components, which were used to retrieve ip addresses of peers containing a requested file. These servers where not only hosted by eDonkey, as in the case of Napster, but ordinary users could also host a server.

The other network the eMule client connected to was the Kad network. This network is based on a DHT, which in contrast to Gnutella and Kazaa is an structured overlay. A DHT is based on a hash-function which translates everything to a peer-id. Then a keyword search is translated into a lookup for the peer with the id corresponding to the first keyword (the keyword is translated using the hash-function, which results in the id). Then after finding the responsible peer the complete keyword list is send to it. This peer then looks up all the results for the search and returns the peers which should have files matching the keywords, described by Stutzbach et al.[26].

This works because when a peer publishes a file (or shares it) a publish message is created. This message is send to all peers to which a keyword translates to its id using the hash-function (and a couple of neighbors). After 24 hours this process is repeated, as described by Steiner et al.[25].

This scheme allows for easy searches, but publishing is more complex. According to Steiner et al.[25] 90% of all messages in the Kad network are publish messages. This equally bad as the 92% of the Gnutella network and thus only

---

<sup>2</sup><http://en.wikipedia.org/wiki/Kazaa>

<sup>3</sup>[http://en.wikipedia.org/wiki/EDonkey\\_network](http://en.wikipedia.org/wiki/EDonkey_network)

changes the problem. Additionally a DHT has more problems regarding churn etc. making us prefer a solution using an unstructured overlay.

Another feature of the eMule clients was its use of credits. These credits were exchanged between clients after a peer uploaded to it. A peer which had credits on a corresponding peer gained priority over peers which did not and would receive an upload slot quicker. Basically peers were rewarded for uploading. In mid 2005 2-3 million users were connected to the eDonkey network.

## BitTorrent

BitTorrent<sup>4</sup>, the current leader in P2P networks according to Ipoque[23], started operating in 2001 and uses websites to perform searches. These websites provide a way to download a .torrent file. This file contains a list of trackers. Trackers are servers, which ordinary users can setup, and maintain a list of peers which are currently downloading/seeding the file (these peers together make up a swarm). A peer is called a seeder when he has completed the file, but is still uploading. Inside a swarm peers are exchanging pieces of a file, using the tit-for-tat mechanism. This mechanism prevents peers from free-riding (downloading without uploading), its goal is similar to eMule's credits but it is more advanced. BitTorrent thus uses a combination of two centralized components, but with no correlation to the original authors.

Since version 4.2.0 of the BitTorrent protocol some steps are made to move to a trackerless BitTorrent and thus remove one of the two centralized components. But a solution for searching has not been found yet. As with Napster lawsuits against trackers have led to the closure of some of them.

## 2.2 Research Question

Given the current state as described above, a different approach is needed to find an efficient method to locate files in a P2P network without the use of a centralized component. The overhead of not having a centralized component, results in wasting 90% of all messages on search. This creates an unscalable network, which is undesired.

Voulgaris et al.[29] suggest creating an additional semantic overlay, so using two overlays instead of one. Using this additional overlay, which consists of 10 neighbors which are similar, results in a much improved hit-rate and thus a higher  $QE$  (a hit-rate of 36% is achieved). This overlay is then used only for searching and has a very small overhead, also it separates data from meta-data. This is preferred because both have different requirements, the data overlay focusses on speed whereas the meta-data overlay focusses on similarity. Voulgaris et al.[29] also discuss the bandwidth requirements needed for such an additional overlay. These range

---

<sup>4</sup>[http://en.wikipedia.org/wiki/BitTorrent\\_\(protocol\)](http://en.wikipedia.org/wiki/BitTorrent_(protocol))

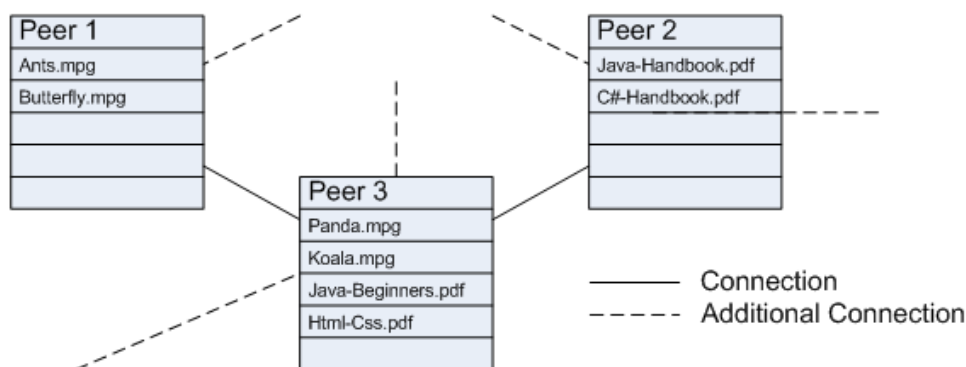


Figure 2.2: An Example of a Semantic Overlay, as Constructed by Tribler

from 213 to 640 bytes per second, which is as suggested by Voulgaris small if not negligible compared to the bandwidth consumed by downloading files.

## Tribler

Tribler[1] is a P2P client which incorporates such an additional overlay. It is a standard BitTorrent client, but uses advanced techniques from the information retrieval domain to find peers with similar interests and construct a semantic overlay. These peers then exchange information regarding their download history using .torrent files (these files will be called items for consistency). Tribler then uses a similarity function which matches the most similar peers and uses them to create a neighborhood. An example of such an overlay is shown in Figure 2.2, where peer 1 and 3 and peer 2 and 3 are connected. But no connection exists between peer 1 and 2, because of not being similar enough (peer 1 is interested in animals, peer 2 in programming).

Because after matching our neighborhood then only consists of peers which have similar interest, search is then much more efficient. Each Tribler peer stores 5000 items locally in a database. It then connects to 10 neighbors which have similar interest. Combining the local database and those of the 10 connected neighbors, peers can search an effective 55000 items as stated in the TriblerSpec[4]. This eliminates the need for a flooding variant keyword search.

When searching for an item, first the local database is searched. Then a QUERY message is send to all 10 neighbors. This message contains an id and a query. The neighbors respond with a QUERY-REPLY message, which contains a list of answers. This list does not contain items yet, but only meta information. Upon receiving a QUERY-REPLY message, the list is added to the search results. First only local results of a query are shown, but after receiving a QUERY-REPLY message this list is updated. If a user clicks on a remote result, the corresponding item retrieved which contains all information needed for downloading.

This schema should in theory provide an efficient way to do keyword search,

because only 10 messages are send. But the quality of the results depend on the similarity function, because when peers are better matched the overlay is better, the hitrate increases and search will improve. The current similarity function is not performing that well (both in terms of matching users and computation performance), which results in the following research question:

“Find a similarity function from which an efficient P2P topology for keyword search emerges.”

To evaluate the different options available Chapter 3 will describe the different types of similarity functions. Then in Chapter 4 the additional requirements introduced by the P2P environment will be treated. Chapter 5 looks at the different datasets available for testing similarity functions and finally Chapter 6 will conclude with future research topics.

## Chapter 3

# Similarity

Using a similarity function allows us to optimize the semantic overlay. But similarity functions are also used in other fields, primarily e-commerce. The explosive growth of the Internet has led to the emergence of e-commerce. This new form of commerce allows the customization of a store to the users preference. The CEO of Amazon.com has said:

“If I have 3 million customers on the Web, I should have 3 million stores on the Web.”[22].

A way to achieve this is to create a personal store tailer made especially for each user. Using the history of a user, and comparing it to the history of other users recommendation can be made. Because if very similar users bought products this user has not, then these products should be very interesting to him.

An example: Mark and Jim have similar taste in books, every time a new Harry Potter book is released they both buy it. But Jim was on a vacation when the new Harry Potter book was released and completely missed the release of it. After his vacation, Jim logs in into his favorite online bookstore and the new Harry Potter is then recommended. This recommendation is made using both the history of Mark and Jim, because Jim has not bought it yet and Mark did. Jim sees the new Harry Potter book and immediately buys it, resulting in an increase in revenue for the bookstore.

### Customers Who Bought This Item Also Bought <sup>1</sup>

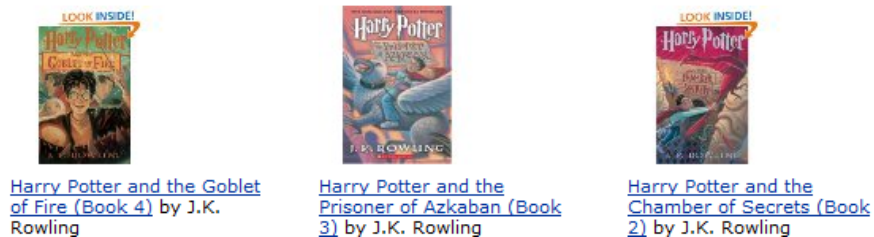


Figure 3.1: Recommendations made by Amazon.com

Using recommendations information can be filtered specifically for a user. If no recommendation was made to Jim, he should have searched for Harry Potter books in the store. Which he obviously would not do on a daily basis. By creating this tailer made store for Jim and other users more books will be sold, the primary reason for the interest in similarity function by e-commerce websites.

Several different similarity functions are being used which can be separated into two main groups. Memory-based similarity functions which use information that is available on users to compute similarities. And model-based similarity functions which generate a model based on the information and then use this model to find similar users. In the next sections the most common similarity functions will be treated.

### 3.1 Memory-based

	item 1	item 2	item 3	item 4
user 1	1	3		5
user 2	1		5	4.5
user 3	4	1	1	
user 4	4	1		2

Table 3.1: Example of User-Item matrix with ratings.

Memory-based similarity functions predict ratings using a database containing all other ratings. This database consists of items and users. Each user has its own row and the items are assigned to a column. Usually ratings range from 0 to 5 with 0.5 intervals. An example is given in Table 3.1. Memory-based similarity functions can be used both on-line and off-line. When used on-line the similarity is calculated between users when necessary, in off-line mode a database is created containing all similarities between users. An additional formula that is used in the following similarity functions is:

$$\bar{v}_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j}$$

where  $I_i$  is the set of items which user  $i$  has rated and  $v_{i,j}$  the vote of this user for this item combining this results in  $\bar{v}_i$  representing the average of this user's ratings. All following similarity functions return a weight (a number between -1 and 1) representing the similarity between 2 users.

#### 3.1.1 Pearson's correlation

One of the first functions used for computing similarity is the Pearson's correlation coefficient, as described to Breese et al.[6]. This function is widely used in science to compute the linear dependence of two variables. It was used in a GroupLens

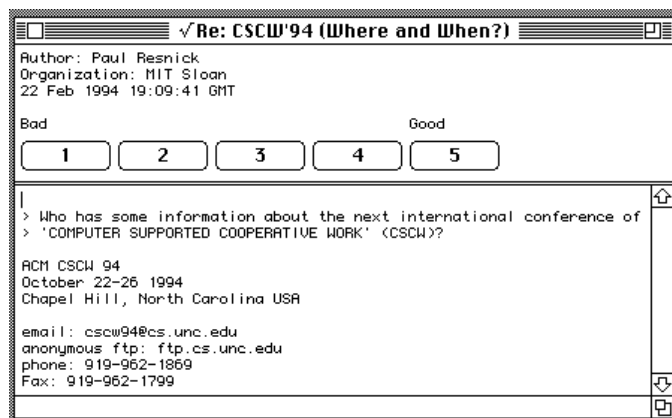


Figure 3.2: A modified newsreader as created by the GroupLens project

project to filter net news. GroupLens modified a newsreader allowing users to rate this news item (Figure 3.2). These ratings were then entered into the Pearson correlation function. Using this function the similarity between user  $a$  and  $i$  can be calculated as:

$$w(a, i) = \frac{\sum_j (v_{a,j} - \bar{v}_a) (v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}} \quad (3.1)$$

where the summations over  $j$  are over the items which both users  $a$  and  $i$  have recorded votes on. Pearson's correlation uses the averages of a user to correct the ratings. This leads to improved similarity because the difference between the rating average of users and this rating is more informative than just using the current rating. As users tend to have a different spread and offset in their ratings.

### 3.1.2 Cosine similarity

Another similarity function is the cosine or vector similarity, also evaluated by Breese et al.[6]. This function is often used in information retrieval to compare two documents by treating each document as a vector of word frequencies. This is being done by i.e. search engines to compare websites. As a similarity function the ratings of the users are used instead of the word frequencies. Resulting in the following formula:

$$w(a, i) = \sum_j \frac{v_{a,j}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \in I_i} v_{i,k}^2}} \quad (3.2)$$

The denominator in the formula is used as a normalizer, such that users with more ratings are not more similar as standard. This improves the similarity function because when a user has rated more items this user is more likely to have a high similarity with more users compared to a user which has only rated 2 items. This is

because there is a larger overlap between users, but the users which have only rated a couple of items could be more predictive. The differences in rating behavior are not used in this algorithm.

### 3.1.3 Differences

Breese et al.[6] has shown by empirical analysis that Pearson's correlation works better for predicting ratings. Other studies have shown that not much difference between them exists.

To predict a rating for a user, the weights between this users and all other users need to be calculated. This can be done by using for example the Pearson's correlation. An example of such a weight calculation for user 1 and 2 is:

$$\frac{(1 - 3)(1 - 3.5) + (5 - 3)(4.5 - 3.5)}{\sqrt{(1 - 3)^2 + (5 - 3)^2 (1 - 3.5)^2 + (4.5 - 3.5)^2}} = 0.98$$

A weight of 0.98 is very high, thus these two users are very similar (which can also be seen in Table 3.1). After computing all the weight a prediction can then be made. To do this we need an additional formula which can convert the weights of multiple users in to a predicted rating. Continuing the example by using such a function as described by Breese et al.[6]:

$$p_{a,j} = \bar{v}_a + k \sum_{i=1}^n w(a,i) (v_{i,j} - \bar{v}_i) \quad (3.3)$$

in which  $k$  is a scaling factor. Using this formula it is possible to predict a rating for item 3 for user 1. The following example is a simplification because only the weight of user 2 is used in the calculation.

$$3 + 1 \cdot (0.98 \cdot (5 - 3.5)) = 4.47$$

Resulting in a prediction of 4.47 for item 3 for user 1. This is as said a simplification but it can be seen that because of the differences in ratingbehaviour between user 1 and 2 (a difference in mean), the rating of 5 is reduced to 4.47.

### 3.1.4 Improving

Both Pearson's correlation and Cosine similarity functions can be improved. These improvements help the similarity functions under special circumstances, i.e. when not much rating information is known on this user (sparseness).

Default voting as suggested by Breese et al.[6] is an example of such an improvement. In Pearson's correlation only the items on which both users have voted on ( $I_a \cap I_j$ ) are used in the calculations. This can lead to a very small input set on which similarity has to be calculated. The term associated for this problem is sparseness (or how to deal with an almost empty matrix). A way to cope with sparseness is using the union of rated items ( $I_a \cup I_j$ ) instead and use a default value to

fill in the gaps. Then we can calculate the similarity over a larger input set. The default value should reflect a neutral or somewhat negative rating. An example of such a rating is the mean of a user's ratings or the middle of the scale. It is empirically shown by Adomavicius et al.[3] that default voting improves the similarity function.

Another possible improvement is incorporating inverse document frequently (IDF) method in the similarity function. IDF is used in information retrieval to distinguish the common words from the important ones by looking at their frequency of use. If a word is used in all documents then it usually has nothing to do with the topic of this document. But if this document contains a word that is not used in any other document then this word is much more describing of this document. Applying this technique to the cosine similarity function removes/reduces the importance popular items which have been ranked by most people. According to Breese et al.[6] this improves the vector similarity function with 2.4%. An example of this technique is given by Herlocker et al.[9] where it is stated that liking the movie "Titanic" says less about a users preference than liking the movie "Sleepless in Seattle". In the dataset they used (MovieLens) almost all users liked "Titanic", therefore it is not considered to be an indicator of a users taste. But ratings of the movie "Sleepless in Seattle" can be used to differentiate users between those who like action movies and those who like romantic ones.

The Ringo music recommender implemented another claimed improvement, the constrained Pearson's correlation coefficient as described by Herlocker et al.[9]. This modification of the Pearson's correlation function replaced the average users vote with the number 4, i.e. default voting. This was the middle of the scale used in the Ringo music recommender.

The Ringo music recommender also used a fixed threshold to limit the number of users which influenced the predicted rating. This improves the similarity function because users with a very low similarity will not contribute anything to the rating and should thus be ignored. The Bellcore video recommender also described by Herlocker et al.[9] expanded on this by selecting only a random number of neighbors to calculate similarity for and then only using the best matching neighbors, this helps reduce the computational complexity of a similarity function. A common name for these improvements are neighborhood selection mechanisms.

Herlocker et al.[9] also look at the neighborhood selection of the similarity functions and observed that users with a small common set ( $I_a \cap I_j$ ) had a significant impact on the predicted rating. To solve this Herlocker et al. introduced significance weighting as an additional parameter which when users had a common set of less than 50 items reduced the significance of the similarity value by  $\frac{|I_a \cap I_j|}{50}$ . This resulted in a significant improvement. The idea behind this improvement is simple when two users only have one item in common, but rate that item the same should those two users have a big impact on the predicted ratings of each other? This improvement works better for the Pearson's correlation function because this function does not take into account the common set, as opposed to the Cosine function.

Rating normalization is another improvement, but an improvement of the prediction formula. The formula described in Formula 3.3 also takes into account the differences in rating behaviour i.e. some users rate items a 5 if they are good and a 1 if they are bad, while others can rate 3 and 1 instead. But using z-scores, improves this formula even more. When using z-scores, as described by Herlocker et al.[9], the prediction formula then changes to:

$$p_{a,j} = \bar{v}_a + \sigma_a \frac{\sum_{i=1}^n w(a,i) \left( \frac{v_{i,j} - \bar{v}_i}{\sigma_a} \right)}{\sum_{i=1}^n w(a,i)} \quad (3.4)$$

This is an improvement at relative cheap computational costs and it is empirically shown that Z-scores are a better way to incorporate the difference in rating behaviour.

### 3.1.5 Item-Item

Instead of calculating similarity between users, similarity between item could also be calculated. This is a common practice in e-commerce because there are usually more users than items and thus item-item similarity reduces the computational complexity. Item-item similarity is usually computed off-line and stored in a table, but after this off-line computation looking up/predicting a new rating is fast. The matrix can be constructed using the ratings of all users and contains all item-item similarities as described by Sarwar et al.[21]. All similarity functions as described above can be used, the only change is using the items as input instead of users. Karypis[11] states that item-based similarity can be as much as 28 times faster, this includes the off-line computation. But this number is very much dependent on the dataset that is used. Furthermore Karypis compares user-based similarity functions with these item-based functions and showed that they did not perform worse.

The computational results for the given datasets are shown in Table 3.2. RTime is the time needed to compute all recommendations, RRate is the number of recommendations/second and MTime the time needed to compute the model. As a side note, no information about the datasets that were used was given and an unoptimized version of a memory based cosine function 3.1.2 was compared to an optimized version of a Pearson's correlation function.

But nonetheless item-based similarity is used in a wide variety of e-commerce websites. Amazon.com being one of the well known examples. Linden et al.[14] describes the algorithm that is used by Amazon.com, it is a variant of the cosine similarity function. It is further optimized by first creating a table which contains all co-purchased items for each item. This creates a dense table upon which the cosine similarity function is run. This step is executed off-line, on-line only a table lookup is needed to recommend items to a user. Because this computation is only dependent on the number of items which a customer rated and independent of the total number of items/customers it can be done in real time. Multiplying the ratings of already rated items by their item similarities with the selected item.

Name	User-based		Item-based		
	RTime	RRate	MTime	RTime	RRate
ecommerce	4.05	1646	0.92	0.33	20203
catalog	27.20	1848	4.11	2.20	22817
ccard	50.04	851	7.85	2.43	17542
skills	6.50	672	1.30	0.23	19017
movielens	3.38	278	1.54	0.20	4715

Table 3.2: Computational complexity of User-based compared to Item-based similarity functions.

The PocketLens[16] paper describes an item-item similarity function which is especially created for mobile use. It is based on the cosine similarity function but modified to allow incremental computation of the item similarities. The item-item matrix is created specifically for a user thus only needs rows for the items which a user rated and columns for the items which it did not. This keeps the model size relatively small. In each cell of the matrix 4 different values are stored allowing for incremental updating. Then when a user receives a new set of ratings  $C$  from a neighbor all cells  $M(O_i, N_j)$  need to be updated of the items in set  $C$  which have been rated ( $O_i \in (O \cap C)$ ) combined with the item which have not ( $O_i \in (O - C)$ ). The values are then calculated as:

$$PartialDot(O_i, N_j) = PartialDot(O_i, N_j) + u_k w_k \quad (3.5)$$

$$PtLenU(O_i, N_j) = PtLenU(O_i, N_j) + u_k u_k \quad (3.6)$$

$$PtLenW(O_i, N_j) = PtLenW(O_i, N_j) + w_k w_k \quad (3.7)$$

$$Cooccur(O_i, N_j) = Cooccur(O_i, N_j) + 1 \quad (3.8)$$

And when similarity between two items needs to be calculated:

$$sim(O_i, N_j) = k \frac{PartialDot(O_i, N_j)}{\sqrt{PtLenU(O_i, N_j)} \sqrt{PtLenW(O_i, N_j)}} \quad (3.9)$$

where  $k$  is the  $\frac{1}{50}$  improvement as suggested in 3.1.4. The time needed for computation is estimated for a pda at 0.263 seconds, when using a neighborhood of 500 items.

## 3.2 Model-based

The opposite of memory-based similarity functions are model-based similarity functions. Here first a model is created from the available data, which is then used to predict a rating of an item for this user. Model-based similarity functions are by definition always off-line created. The general form of a model-based similarity function looks like:

$$p_{a,j} = \sum_{i=0}^m Pr(v_{a,j} = i | v_{a,k}, k \in I_a) i$$

where  $Pr$  is the probability that this user will rate this item a  $i$ . Because after a model is created calculating the predicted ratings are usually some simple table lookups the actual prediction is much faster compared to memory-based similarity functions. The downside of this method is that it needs  $n^2$  space to store for  $n$  items, as described by Sarwar et al.[21]. Optimizations can be made by only storing the  $k$  most similar items, reducing the size of the table to  $kn$ .

## Clustering

An approach to model-based similarity functions are clustering algorithms. The Eigentaste algorithm as proposed by Goldberg et al.[7] is an example of such a function. It uses the Principal Component Analysis (PCA) to divide all users into clusters. PCA can reduce the dimensionality of data, for example PCA can plot all users into a two dimensional graph. Using this graph the data is then divided into clusters, with the density of those clusters increasing towards the origin. Figure 3.3 is an example of this. It is common to use two dimensions for PCA clustering because humans can visualize this better, but the Eigentaste algorithm can work with any number of dimensions.

Eigentaste is implemented in a joke recommending system called Jester. Jester first asks ratings for a set of predefined jokes. Then these ratings are used to determine in which cluster a user resides. Using the ratings the best matching cluster can be selected, by comparing the ratings of a user to the mean-rating of each cluster. After a user is assigned to a cluster, the means are again used but now to display the highest rated/not yet seen jokes. Ratings of new jokes are used to influence the mean of the jokes in this cluster, but this mean is not recalculated on line. Because recommendation now only involves looking up the best not rated joke the complexity is reduced to  $O(k)$  where  $k$  is the number of jokes up for recommendation.

But we can improve on Eigentaste, because one of its limiting factors is the assignment of a user to one cluster. By using multiple clusters, as suggested by Ungar et al.[28], a better recommendation can be made. An example of such an algorithm is the K-Mean clustering, which is a simple example of a multiple clustering algorithm. K-Mean clustering compares the ratings of a user to the mean of a cluster to determine the degree of membership, instead of selecting the best matching cluster. This results in multiple memberships, but to a certain extent. Then when a recommendation has to be made all the  $means \cdot degreeofmembership$  are calculated for all non-rated items to predict the best item. For example if a user belongs to cluster  $A$  with a degree of membership of 0.5, then the means of cluster  $A$ 's non-rated jokes are multiplied by 0.5, the same goes for cluster  $B$ ,  $C$  etc. Eventually all clusters are processed and the best joke is presented to the user. This improves the recommendation.

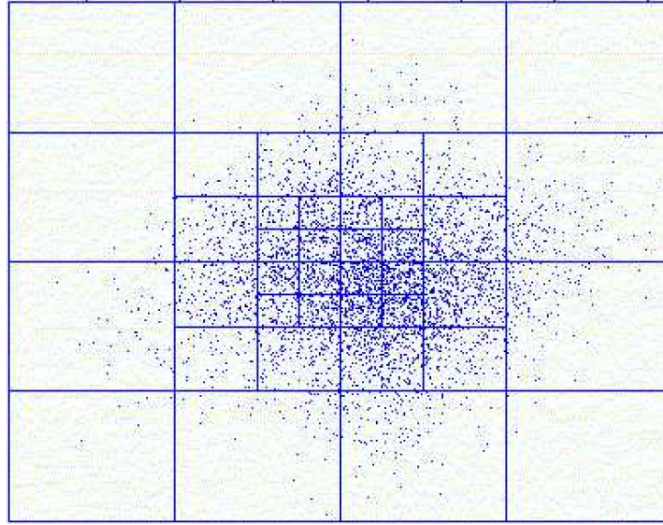


Figure 3.3: Eigentaste clusters

### 3.3 Hybrid/Combined

Instead of choosing for a memory or model-based similarity function another option is to combine them, hoping to get the best from both worlds. An example of a hybrid approach is described by Wang et al.[31], where similarity fusion is suggested. This algorithm combines results from a user-item and a item-item similarity function to improve performance. It uses a binary values to prefer one or the other. The resulting weight calculation is then:

$$p_{a,j} = \sum_{i=0}^m Pr(v_{a,j} = i | SUR, SIR, SUIR) i \quad (3.10)$$

this looks very much like the default model-based equation as described in Section 3.2. *SUR* is the user-item and *SIR* is the item-item based similarity function. And third table is introduced *SUIR*, this table is obtained by sorting the rows and columns according to dissimilarity. The active user/item is located in the top-left cell and moving further away from this cell results in less similar users/items. Hidden in this equation are  $\lambda$  and  $\delta$  which are weights that increase/decrease the importance of one of the matrices. This is typical for a model-based similarity function and are usually calculated by performing experiments on test sets. Experimental result shows that combining these algorithms improves performance when used on a sparse dataset. The computational complexity of combining functions has not received any attention.

Good et al. [8] propose combining collaborative filtering and personal agents. Four hypotheses are introduced in order to prove or disprove this:

1. The opinions of a community of users provide better recommendations than

a single personalized agent.

2. A personalized combination of several agents provides better recommendations than a single personalized agent.
3. The opinions of a community of users provides better recommendations than a personalized combination of several agents.
4. A personalized combination of several agents and community opinions provides better recommendations than either agents or user opinions alone.

It uses combinations of 23 different agents and the DBLens collaborative filtering engine created by the PocketLens research group. It then evaluates the hypotheses. Surprisingly it states that there are multiple personal agents which provide a better MAE than using collaborative filtering, thus the first hypothesis is discarded. Combining the agents and a CF with one neighbor results in a much improved MAE, but ultimately the best solution is the combination of all agents and a CF with an unlimited neighborhood. Again nothing is stated regarding the computational complexity and the tests were limited to 50 users which is a very small set.

### 3.4 Log-based

A problem with all similarity functions as described above is that they are based on ratings. These ratings typically range from 0 to 5, but sometimes no rating information is available. This is the case when recommendations have to be made based on web logs, but also a study by Wang et al. [30] has shown that users are very unlikely to provide explicit ratings and this should be avoided when possible.

Using only web logs several techniques can be applied to extract implicit information on a user. Sugiyama et al.[27] introduce a technique which uses the most important terms of a website a user visits to compose a profile for this user. These terms are retrieved using the TD/IDF algorithm, and stored in different columns. Each term is stored as a column and in the cells the frequency of this term for a website is stored. Additionally the whole row is normalized.

This profile then describes the preference of this user, or what type of information he/she searches for. Then using Pearson's similarity function similarity is computed between users. The read time of a website is also used in the computation, because this number says something about the value of a website. Then when searching for new websites, the similarity between this user and others can be used to suggest websites that are particularly useful to this user. Sugiyama calls this approach Adaptive Web Search.

Rafter et al.[17] use a different approach. They applied the CASPER project to extract data from their web logs of an online recruitment environment. The statistics that were extracted were read time, revisits and whether a user applied

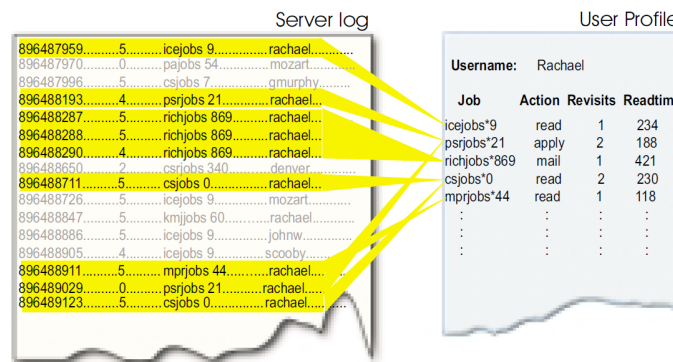


Figure 3.4: Implicit profile construction as done by CASPER[17]

for this job. CASPER collapses revisits that were due to slow connections (so-called irritation clicks), and uses sequential page requests to compute read time as shown in Figure 3.4. Read time is an important factor, because it is an indicator of interest for an item and thus can replace a rating for an item. To further optimize the read time factor the average read time of this user and this job is also stored, this helps to determine the significance. CASPER uses a very simple algorithm to calculate similarity which only uses the overlap between two users.

Tribler currently only stores download information of a peer, so no rating information is available. A database is created in which all information of peers is stored in a user-item style matrix. In addition of the download history, the recently introduced Clicklog<sup>1</sup> stores searches combined with the items that were clicked. It is possible that some additional information can be extracted from this data.

<sup>1</sup><http://www.tribler.org/trac/wiki/ClickLog>



## Chapter 4

# Spam and Duplicates

As described in the previous section sparseness is a big problem for similarity functions and using techniques such as default voting etc. improves the quality of the recommendations. In a distributed context (or in a P2P environment) the sparseness problem explodes because of the lack of a central authority which structures the items. The unstructured environment causes a single item to have multiple versions.

The cause of these versions can be separated into two categories: Spam and duplicates. Spam is intentional and is caused by companies/organizations that publish corrupt versions of a file. The goal of this is to frustrate users and cause them to look for other options to retrieve the file. This is a common technique against piracy, the record companies try to motivate people to buy cd's instead of downloading them.

Duplicates occur when users publish their own version of a file. This can have several reasons i.e. because their version has a better/worse quality, is a live recording, has subtitles etc.

Liang et al.[12] used the FastTrack network to determine the number of versions of seven songs which resided high in the charts. Table 4.1 shows that this behavior has a huge impact on the sparseness of a matrix.

Song Title	Number of Versions
Naughty Girl	26.715
Ocean Avenue	8.000
Where is the Love?	48.613
Hey Ya	46.926
Toxic	38.992
Tipsy	32.893
My Band	49.447

Table 4.1: The Number of Versions of seven songs as discovered by Liang et al.[12].

Dealing with Spam requires a trust function, this is an active field in research and these functions are still under active development. An example of a trust function is the EigenTrust algorithm as described by Kamvar et al.[10]. The EigenTrust algorithm relies on relationships. If person  $a$  trusts person  $b$  and person  $b$  trusts person  $c$  then person  $a$  could trust person  $c$  to some extent. As said progress is made in these functions but for now let's assume that a working trust function exists which helps to almost eliminate the Spam.

## Duplicates

When downloading a duplicate version the end-user will not notice any difference, because the file will work. But having many duplicates causes some problems.

One notable problem occurs when matching users. If two users downloaded almost the same file, the only difference being a different quality/bitrate, and did so for their complete download history. Then these two users should be matched as highly similar, but using the similarity functions as described in Section 3 they will not.

To match these users additional semantic data needs to be incorporated. The similarity between items could then be used together with a similarity function using a hybrid technique as described in Section 3.3. Several hybrid techniques have been especially developed to use semantic data as a way to improve similarity.

An example of such a hybrid technique is the Fab recommender as described by Balabanović et al.[5]. Fab is a recommender for the Web and incorporates both collaborative filtering and content-based filtering. The combination of these two techniques is used to solve the problems as described above, but then for the web. If one user likes the CNN weather page and another user likes the MSNBC weather page then, using only collaborative filtering, the two would not necessarily end up being nearest neighbors. This is in essence the same problem. Balabanović et al. solve this problem by using multiple agents. A collection agent which searches the web for pages containing a specific topic and a selection agent which filters all collected web pages using a single user's specific taste. Ratings from users are forwarded back to the nearest neighbors and to the collection agent. A high rating will cause the collection agent to find similar web pages based on content.

Another approach is described by Adomavicius et al.[2]. They incorporate additional contextual information into the user-item matrix by extending it. This model is called the multidimensional recommendation model. An example of contextual information that could be used is given: A recommender system may recommend a different movie to a user depending on whether she is going to see it with her boyfriend on a Saturday night or with her parents on a weekday. All rating information is stored in a multiple dimension matrix, but when similarity has to be calculated this matrix is reduced. The following formula is given to achieve this:

$$R_{User \times Content \times Time}^D(u, c, t) = R_{User \times Content}^{D[Time \in weekday]}(User, Content, AGGR(rating))$$

This reduction leads to a two dimensional recommendation space ( $User \times Content$ ) which contains the aggregation of all ratings made on weekdays instead of using i.e. only Mondays. The reduction is needed because when only Mondays would be used the recommendation would become inaccurate because of the lack of sufficient ratings. Also this reduction allows the use of traditional collaborative filtering using the similarity functions described in Section 3.1. But because using all weekdays is still more precise than just using all ratings (weekdays and weekend combined) the reduction based CF outperforms traditional CF. To test which different dimensions influence the ratings Adomavicius et al. calculated the averages per user of the different categories of an dimension (i.e. weekend or weekday for the time dimension). And then applied a t-test to determine if there was a significant difference between them. The performance of this segment-based CF was then compared to traditional CF and empirically showed that it improved performance. As a side note the input set contained only 1373 ratings and 117 users, but the same CF function was used in both datasets.

Melville et al.[15] introduces Content-Boosted Collaborative Filtering (CBCF). This is a variant of CF where the sparseness problem is reduced by using content similarity to create a pseudo user-ratings vector. This vector is then used to compute predictions. Several improvements were made which reduce the influence of users which rated only a few items and a ratio is introduced which gives more preference to the pure content based ratings. Comparing this approach to a pure CF solution, a pure content based one and a naive hybrid algorithm (which used the average of the CF and content based algorithm) the CBCF algorithm performed 4% better than traditional CF.

The final approach that will be discussed is described by Shepitsen et al.[24]. It tries to solve the problem of ambiguity of tags, this is a problem that occurs when a user specifies a tag that can be used to describe different things. I.e. when searching for java, did a user mean the programming language or the island? Shepitsen et al. split the process of recommendation into two steps.

The first step is to calculate similarity as usual. But in the second step the similarity is multiplied by another factor which determines the relevancy of this item to the user. Calculating the relevancy requires the computation of tag clusters by using a modified version of the Hierarchical Agglomerative Clustering algorithm. The algorithm first creates a cluster for all tags, then these clusters are combined using a similarity level. This level starts at 1 and is decreased until all clusters are combined. The result of this algorithm can be seen in Figure 4. More similar tags are clustered sooner, as shown in the Figure 4 as in the case of the Red sox and White sox.

When a search is made, the specified tag is selected. Then a predefined number of levels is traversed up (to widen the search) and the tags that are in this subtree are used as the relevant tags. Then for each item that is in the top-n of similarity function the closeness to each of those clusters is calculated. This results in the following formulas:

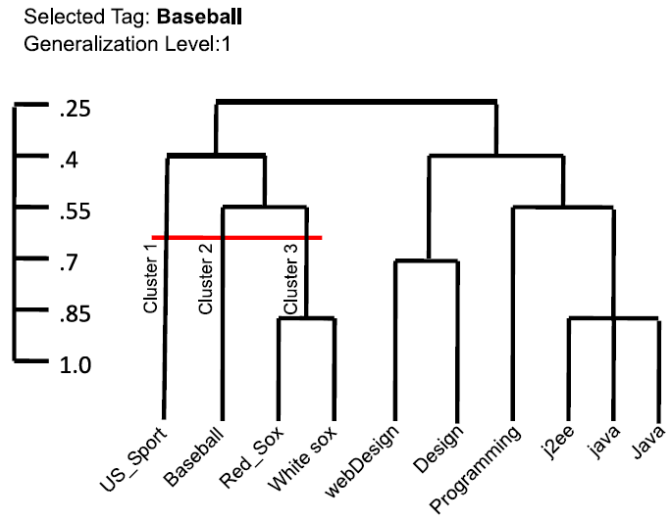


Figure 4.1: Tag clustering as done by Shepitsen et al.[24]

$$I(u, r) = \sum_{c \in C} uc_w(u, c) \cdot rc_w(r, c) \quad (4.1)$$

$$S'(u, q, r) = S(q, r) \cdot I(u, r) \quad (4.2)$$

where  $uc_w$  determines the user-cluster similarity,  $rc_w$  the item-cluster similarity and  $S'$  is the updated similarity for an user-item. By combining the original similarity and this new relevancy the ordering of the list changes. Empirical analysis was done on a subset of del.icio.us and last.fm profiles. The new approach led to an improvement of 0.35 on the del.icio.us set, but on the last.fm set the improvement was less. The researchers concluded caused by the sparsity level of the del.icio.us set, which was 99.9994% (the last.fm set had a lower sparcity level).

## Tribler

But how can these techniques be integrated into Tribler and what additional semantic data is available?

Within Tribler all items are annotated with their the filenames, these are included in the .torrent files thus available. Additionally ClickLog<sup>1</sup> information is available. This log annotates the items with search keywords and positions when clicked. Annotating an item with search keywords gives a more reliable source of tagging, because users are specifically linking a search keyword with an item. Using this additional implicit information user profiles can be constructed.

<sup>1</sup><http://www.tribler.org/trac/wiki/ClickLog>

Using the integrated player an approach resembling last.fm could be used to create a profile. A song which is submitted to last.fm needs to meet the following requirements<sup>2</sup>:

- Each song should be posted to the server when it is 50% or 240 seconds complete, whichever comes first.
- If a user seeks (i.e. manually changes position) within a song before the song is due to be submitted, do not submit that song.
- Songs with a duration of less than 30 seconds should not be submitted.

This process results in a profile which consists of the number of times each particular song has been played. Using this information similarity between users can be calculated.

---

<sup>2</sup><http://www.audioscrobbler.net/wiki/Protocol1.1>



## Chapter 5

# Testing Methodologies

When testing a similarity function datasets are used as input. These datasets are typically divided into two parts, a learning and a test set. The test set contains at least one rating for each user that needs to be predicted. A common division between learning and test set is 80% - 20% or 90% - 10%, but all thinkable combinations are possible.

To compare the performance of different algorithms consistent datasets should be used. However this is not the case, in this chapter some of the more common datasets and evaluation methods are discussed.

### 5.1 Datasets

A dataset is usually one or more databases containing users and items. Then for each user,item combination a rating could exists. The number of empty cells determine the sparseness of a dataset. Because sparseness is a big problem for similarity functions, it can be used to indicate the difficulty of a dataset. Usually sparseness is shown as the sparsity level, which can be calculated using:

$$Sparsity = 1 - \frac{NrOfRatings}{NrOfUsers \cdot NrOfItems} \quad (5.1)$$

The higher the sparsity level, the more empty the dataset. Resulting in predictions which are less accurate, because of less overlap between users.

### MovieLens

One of the most used datasets is the Movielens dataset. This continuously expanding dataset can be obtained from the GroupLens project. Currently three different datasets are available from the Movielens website <sup>1</sup>, the sparsity level is shown in brackets:

---

<sup>1</sup><http://www.grouplens.org/node/73>

- 100,000 ratings for 1682 movies by 943 users (0.9369)
- 1 million ratings for 3900 movies by 6040 users (0.9575)
- 10 million ratings and 100,000 tags for 10681 movies by 71567 users (0.9867)

But instead of just using these three datasets, some papers have compiled their own version. The different versions are shown in Table 5.1. The dataset used in the PocketLens paper[16] has the highest sparsity level (second row), but this level does not come close to the sparsity level of the del.icio.us dataset used in the previous chapter (99.9994%). The dataset used in the hybrid approach by Good et al.[8] has no indication of the number of items, thus sparsity could not be calculated. But because of the low number of users and the selection of users which had at least 120 ratings it can be expected that the sparsity level would be low.

### Alternatives

But if the sparsity levels of the Movielens dataset are not sufficient then which dataset should be used. Other papers use different datasets, but this is mainly because the Movielens dataset does not suit their needs. I.e. for log-based similarity, then different information is needed such as actions of users. Jet another dataset is needed when tagging information is required, the usually alternatives such as using last.fm or del.icio.us are used. But those datasets are not published thus need to be crawled in order to create a dataset. Resulting in a different dataset each after each crawl.

## 5.2 Evaluation

Two different techniques are used to evaluate the performance of the similarity functions. These techniques seem to be much more standardized then the datasets and both originate from the Information Retrieval field.

### MAE

When evaluating a prediction for a user, papers typically use the Mean Absolute Error (MAE) which defines the absolute average deviation for a user. A number which indicates the accuracy.

Dataset	Description	Sparsity	Used In
Movielens	100.000 Ratings, 943 Users and 1682 Items	0.9369	[9, 31, 11, 21]
Movielens	96.000 Ratings, 1000 Users and 3775 Items	0.9745	[16]
Movielens	50 Users which had at least 120 ratings	?	[8]

Table 5.1: Different variations of Movielens dataset.

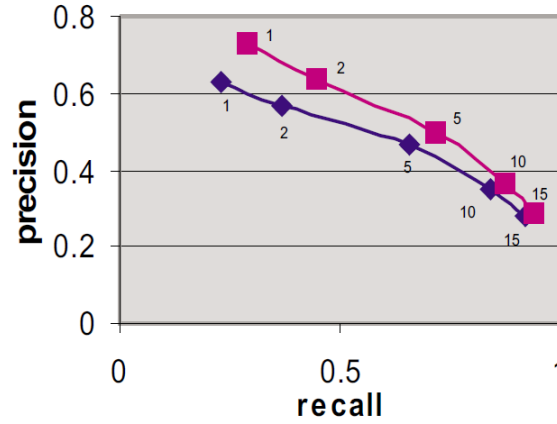


Figure 5.1: Comparing Recall and Precision, from Rafter et al.[17]

$$S_a = \frac{1}{m_a} \sum_{j \in P_a} |p_{a,j} - v_{a,j}| \quad (5.2)$$

In this formula  $m_a$  is the number of predicted ratings for a user,  $p_{a,j}$  the predicted rating of an item and  $v_{a,j}$  the actual value. A number of 0.8 is frequently achieved by a number of algorithms. A lower number indicates better performance, because the similarity functions predicted ratings which were more accurate.

### Precision and Recall

Log-based similarity functions do not predict a rating, thus need to be evaluated in a different way. Then usually precision and recall are used as a statistic. Those number are calculated using:

$$precision = \frac{no.predictedrelevantitems}{no.predicteditems} \quad (5.3)$$

$$recall = \frac{no.predictedrelevantitems}{no.relevantitems} \quad (5.4)$$

Precision is the percentage of relevant items compared to the number of predicted items. Recall is the percentage of relevant items compared to the overall relevant items in the test set that were returned in the top-n list. When the top-n list increases (or the number of returned predicted items increases) recall increases, but precision decreases. This is because more items are returned, resulting in more relevant items in this list, but also more items are returned that are not in that list/relevant. An example a typical precision/recall ratio is shown in Figure 5.1.

## **Convergence and Hitrate**

In order to evaluate the semantic overlay, Voulgaris et al.[29] introduced the convergence speed. The convergence speed is indicative of the performance of neighborhood selection. After each cycle of the gossip protocol, for each peer the average number of common files between a peer and its neighbors is recorded. These numbers are then plotted in a graph, to visualise the difference in performance.

A similarity function that performs better, should show a steeper graph. But for similarity functions using more advanced semantic data, the graph could display erroneous data. I.e. when trying to merge different versions of a single file.

Hitrate is also used by Voulgaris. A randomly selected file was removed from each node, which then after initializing the overlay, was searched for. A semantic overlay that is better matched should have a higher hitrate. Again due to multiple versions, how this statistic behaves needs to be looked at.

## Chapter 6

# Conclusions and Future Work

State of the art solutions such as Gnutella, Kazaa and Kad are not sufficiently scalable. Caused mainly by using an overlay which is not developed to use as a way to search. Optimizing a flooding structure by introducing superpeers which index underlying peers and reduce network requirements only delays an existing problem. Using a DHT and publishing that a user has files to share is equally bad, the publish messages need to be resend after a period. And multiple messages are needed for each file.

Using an additional semantic overlay keeps the transport of data and meta-data separate, this leads to two networks which can be specifically designed for their needs. The data overlay can be optimized to improve data throughput and the semantic overlay improves search. Additionally the use of a semantic overlay allows the P2P field to use proven techniques from the Information Retrieval field. These techniques have proven themselves in e-commerce, search engines etc. So why not use them? As a result the overall structure of the P2P network as implemented by Tribler is by far one of the more optimal. And an opportunity to evaluate the performance improvement of a similarity function in practice as it is one of the few projects that was actually implemented.

### Improving

But the current similarity function as implemented can be improved using an array of different techniques. As a first step it would be interesting to see how the current function compares too the most used similarity functions as described in this research. Because Tribler uses binary data (did a user download a file yes/no), a possible replacement could be the Cosine function as described in Section 3.1.2. Pearson's correlation function is not particularly useful because of the averages it includes in scaling the ratings (which is always 1).

Item-item similarity also sounds very promising, especially the PocketLens algorithm as described by Miller et al.[16]. The iterative aspects of this algorithm and the organization of the item-item matrix which is single user oriented could

be easily transformed into the Tribler setup, where the matrix is also single user oriented.

But because this is an Item-Item matrix, it would be difficult to create the user-user similarities needed for the overlay construction. Also it should be noted that the largest benefit of using an item-item approach is the reduction in computation complexity, because of a lower number of items compared to users (as described by Linden et al.[14] for Amazon.com). But in the P2P environment, this could very well be the other way around. In Tribler each user exchanges his last 50 downloaded items and because of the many different versions the number of items could be a magnitude larger than the number of users.

Incorporating semantic data would also be an interesting possibility. A first option is applying a string distance technique to the file/swarm names of the items and see how it performs. Using this distance as a measure of the similarity of two items, a default voting/pseudo vector could improve similarity for matching users. No research could be found on matching filenames, but several techniques from the IR domain should be efficiently able to do this.

Also the additional information gathered by the Clicklog is also very interesting. The search behavior of an user could describe this user in the same way that Sugiyama et al.[27] used the content of visited websites. Then a much more complete user profile could be constructed resulting in a much improved similarity function. Also the search keywords could be used as tags and then open up possibilities to cluster them etc.

The tags provided by the Clicklog could also be used in calculating similarity between files. Analyzing the Clicklog could also reveal a hidden gem, which is hidden inside the log. This gem could be uncovered using the techniques as described by Adomavicius et al.[2]. Which determine if a category influences the ratings of users by running a t-test on the averages of each user.

## Evaluating

Equally difficult as improving the similarity function is proving that this improvement actually improved search noticeably. All of the techniques show an improvement of the similarity function, but how noticeable is this improvement? And how do we test this improvement? A reduction of the MAE by 0.01 is an improvement but at what cost do we accept this improvement. And by how much will search improve?

Precision and recall, convergence and hitrate are also statistics that give insight into the performance of the overlay. But how should we deal with the large number of different versions of a file. And when a different version is recommended is that by definition wrong, because of not taking into account the preferences of a user (language, quality etc.)? Possibly comparing a sanitized version of a dataset and comparing that to the original dataset (including multiple versions) and looking at the difference in precision and recall, the influence of these duplicates could be

determined. Also the recommendation has to include a filter to recommend only one version of the same file.

The MAE statistic gives insight in the performance of the predicted ratings. But is not used for log-based evaluation, which is needed to evaluate the performance of the Tribler similarity function. Also plotting all the differences instead of just averaging them reveals more information about the performance of an algorithm. If an algorithm has a more consistent behavior, this algorithm should be preferred above an algorithm that has a lot of differences in performance.

The usage of a dataset which is more resembling to the P2P setting, in terms of sparsity levels, is also recommended. Because testing with different sparsity levels influences the performance of the algorithms and quite possible paints a nicer picture. But what is the sparsity level of a default Tribler dataset?

The computation aspect of finding an optimal similarity function should not be forgotten. The ultimate goal should be to try and find an optimal solution, that is both better at matching peers and has an acceptable computation complexity.

## **Future research**

Summarizing the future research should include:

- Replacing current function with Cosine (Section 3.1.2).
- Evaluating the performance using precision and recall, giving attention to the multiple version problem.
- Improving performance of similarity function with possible improvements as described in Section 3.1.4.
- Using Clicklog data to construct profile of interest as done by Sugiyama et al.[27].
- Using Clicklog data to create tag clusters as done by Shepitsen et al.[24].

,which should result in a very interesting Master of Science project.



# Bibliography

- [1] Tribler.org wiki. <http://www.tribler.org>.
- [2] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145, 2005.
- [3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. 17(6):734–749, June 2005.
- [4] A. Bakker, J.J.D. Mol, J. Yang, L. dAcunto, J.A. Pouwelse, J. Wang, P. Garbacki, A. Iosup, J. Doumen, J. Roozenburg, Y. Yuan, M. ten Brinke, L. Musat, F. Zindel, F. van der Werf, M. Meulpolder, J. Taal, and B. Schoon. *Tribler Protocol Specification*.
- [5] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, 1997.
- [6] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. pages 43–52. Morgan Kaufmann, 1998.
- [7] Ken Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. Technical Report UCB/ERL M00/41, EECS Department, University of California, Berkeley, 2000.
- [8] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.
- [9] Jon Herlocker, Joseph A. Konstan, and John Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.*, 5(4):287–310, 2002.
- [10] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM.
- [11] George Karypis. Evaluation of item-based top-n recommendation algorithms. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254, New York, NY, USA, 2001. ACM.
- [12] Jian Liang and Rakesh Kumar. Pollution in p2p file sharing systems. In *IEEE INFOCOM*, pages 1174–1185, 2005.
- [13] Tsungnan Lin and Hsinping Wang. Search performance analysis in peer-to-peer networks. In *Proc. Third International Conference on Peer-to-Peer Computing (P2P 2003)*, pages 204–205, 1–3 Sept. 2003.

- [14] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *7(1):76–80*, Jan.–Feb. 2003.
- [15] Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth national conference on Artificial intelligence*, pages 187–192, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
- [16] Bradley N. Miller, Joseph A. Konstan, and John Riedl. Pocketlens: Toward a personal recommender system. *ACM Trans. Inf. Syst.*, 22(3):437–476, 2004.
- [17] R. Rafter and B. Smyth. Passive profiling from server logs in an online recruitment environment. In *Proceedings of IJCAI Workshop on Intelligent Techniques for Web Personalization (ITWP2001)*, Seattle, Washington, U.S.A., August 2001.
- [18] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proc. First International Conference on Peer-to-Peer Computing*, pages 99–100, 27–29 Aug. 2001.
- [19] John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: search methods. *Comput. Netw.*, 50(17):3485–3521, 2006.
- [20] Nima Sarshar. Percolation search in power law networks: Making unstructured peer-to-peer networks scalable. In *In Proceedings of IEEE P2P04*, pages 2–9. IEEE Computer Society, 2004.
- [21] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA, 2001. ACM.
- [22] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Min. Knowl. Discov.*, 5(1-2):115–153, 2001.
- [23] Hendrik Schulze and Klaus Mochalski. Internet study 2007. <http://www.ipoque.com/resources/internet-studies/internet-study-2007>, 2007.
- [24] Andriy Shepitsen, Jonathan Gemmell, Bamshad Mobasher, and Robin Burke. Personalized recommendation in social tagging systems using hierarchical clustering. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 259–266, New York, NY, USA, 2008. ACM.
- [25] Moritz Steiner, Wolfgang Effelsberg, and Taoufik En-najjary. Load reduction in the kad peer-to-peer system. In *In Fifth International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2007.
- [26] D. Stutzbach and R. Rejaie. Improving lookup performance over a widely-deployed dht. In *Proc. 25th IEEE International Conference on Computer Communications INFOCOM 2006*, pages 1–12, April 2006.
- [27] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *Proceedings of the 13th international conference on World Wide Web*, pages 675–684. ACM Press, 2004.
- [28] Lyle H. Ungar, Dean P. Foster, Ellen Andre, Star Wars, Fred Star Wars, Dean Star Wars, and Jason Hiver Whispers. Clustering methods for collaborative filtering. AAAI Press, 1998.
- [29] Spyros Voulgaris and Maarten van Steen. *Epidemic-Style Management of Semantic Overlays for Content-Based Searching*. 2005.
- [30] Jun Wang, Arjen P. de Vries, and Marcel J.T. Reinders. A user-item relevance model for log-based collaborative filtering. In *Proc. of European Conference on Information Retrieval (ECIR 2006)*, London, UK, 2006.

- [31] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508, New York, NY, USA, 2006. ACM.