

IT Licentiate theses  
2003-006

# Bootstrapping and Decentralizing Recommender Systems

TOMAS OLSSON

UPPSALA UNIVERSITY  
Department of Information Technology

SWEDISH  
INSTITUTE OF  
COMPUTER  
SCIENCE

**SICS**







UPPSALA  
UNIVERSITET

**Bootstrapping and Decentralizing  
Recommender Systems**

BY  
TOMAS OLSSON

June 2003

COMPUTING SCIENCE DIVISION  
DEPARTMENT OF INFORMATION TECHNOLOGY  
UPPSALA UNIVERSITY  
UPPSALA  
SWEDEN

Dissertation for the degree of Licentiate of Technology in Computer Science  
at Uppsala University 2003

# Bootstrapping and Decentralizing Recommender Systems

*Tomas Olsson*

Tomas.Olsson@sics.se

*Computing Science Division  
Department of Information Technology  
Uppsala University  
Box 337  
SE-751 05 Uppsala  
Sweden*

<http://www.it.uu.se/>

© Tomas Olsson 2003  
ISSN 1404-5117

Printed by the Department of Information Technology, Uppsala University, Sweden

## Abstract

This thesis consists of three papers on recommender systems.

The first paper addresses the problem of making decentralized recommendations using a peer-to-peer architecture. Collaborating recommender agents are connected into a network of neighbors that exchange user recommendations to find new items to recommend. We achieved a performance comparable to a centralized system.

The second paper deals with the bootstrapping problem for centralized recommender systems. Most recommender systems learn from experience but initially there are no users and no rated items to learn from. To deal with this problem we have developed the Genesis method. The method bootstraps a recommender system with artificial user profiles sampled from a probabilistic model built from prior knowledge. The paper describes how this was done for a k-nearest neighbor recommender algorithm in the movie domain. We were able to improve the performance of a k-nearest neighbor algorithm for single predictions but not for rank ordered lists of recommendations.

The third paper identifies a new domain for recommendations – product configuration – where new recommender algorithms are needed. A system that helps users configuring their own PCs is described. Recommendations and a cluster-based help system together with a rule-based configurator assist the users in selecting appropriate features or complete PC configurations. The configurator ensures that users cannot choose incompatible components while the recommender system adds social information based on what other users have chosen. This introduces new complexity in the recommendation process on how to combine the recommendations from the configurator and the recommender system. The paper proposes (1) three new recommender algorithms on how to make recommendations in the domain of product configuration, (2) a method for adding social recommendations to a rule-based configurator and (3) a method for applying the Genesis method in this domain. In this case the Genesis method is implemented by a Bayesian belief net that captures the designers' prior knowledge on how to configure PCs. Then instances of complete configurations are sampled from the model and added to the recommender algorithm.



## Acknowledgements

I would like to thank all my co-workers at SICS, Åsa Rudström, Rickard Cöster and Martin Svensson for their valuable input to this thesis, Niclas Finne and Joakim Eriksson for many nice lunches, and Sverker Janson for giving me the opportunity to work at SICS. Special thanks to Kristina Höök for her encouragement and help in the writing process. I also would like to thank my supervisor Arne Andersson at Uppsala University for his guidance in writing this thesis.

Among the people dearest to me I want to thank my wife Elisabeth for her patience with this thesis, my son Joel for giving me joy, and my parents Leif and Maire for helping me looking after Joel.

Finally, I would like to thank my Lord and Savior Jesus Christ for giving me a life that matters.



# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>Recommender Systems</b>	<b>7</b>
2.1	Content-based filtering . . . . .	9
2.2	Collaborative filtering . . . . .	9
2.3	Economic filtering . . . . .	10
2.4	Knowledge-based recommendations . . . . .	10
2.5	Demographic-based recommendations . . . . .	11
<b>3</b>	<b>Open and addressed research issues</b>	<b>11</b>
3.1	Hybrid Systems . . . . .	11
3.2	Building community . . . . .	13
3.3	Explaining recommendations . . . . .	13
3.4	Validating recommendations . . . . .	13
3.5	Preserving privacy . . . . .	14
3.6	Mobile systems, Ubiquitous Computing and Peer-to-peer . . . . .	15
<b>4</b>	<b>Research Methodology</b>	<b>16</b>
<b>5</b>	<b>Reading Guide and Contributions</b>	<b>17</b>
5.1	A Peer-to-Peer-based Recommender System for a News Headline Viewer . . . . .	17
5.2	Genesis: A method for bootstrapping recommender systems using prior knowledge . . . . .	18
5.3	Enhancing Web-Based Configuration with Recommendations and Cluster-Based Help . . . . .	20
<b>6</b>	<b>Concluding remarks</b>	<b>21</b>
<b>A</b>	<b>Papers</b>	
A.1	Paper I . . . . .	
A.2	Paper II . . . . .	
A.3	Paper III . . . . .	



# 1 Introduction

This thesis consists of three papers on recommender systems.

The first paper addresses the problem of making decentralized recommendations using a peer-to-peer architecture. Collaborating recommender agents are connected into a network of neighbors that exchange user recommendations to find new items to recommend. We achieved a performance comparable to a centralized system.

The second paper deals with the bootstrapping problem for centralized recommender systems. Most recommender systems learn from experience but initially there are no users and no rated items to learn from. To deal with this problem we have developed the Genesis method. The method bootstraps a recommender system with artificial user profiles sampled from a probabilistic model built from prior knowledge. The paper describes how this was done for a k-nearest neighbor recommender algorithm in the movie domain. We were able to improve the performance of a k-nearest neighbor algorithm for single predictions but not for rank ordered lists of recommendations.

The third paper identifies a new domain for recommendations – product configuration – where new recommender algorithms are needed. A system that helps users configuring their own PCs is described. Recommendations and a cluster-based help system together with a rule-based configurator assist the users in selecting appropriate features or complete PC configurations. The configurator ensures that users cannot choose incompatible components while the recommender system adds social information based on what other users have chosen. This introduces new complexity in the recommendation process on how to combine the recommendations from the configurator and the recommender system. The paper proposes (1) three new recommender algorithms on how to make recommendations in the new domain of product configuration, (2) a method for adding social recommendations to a rule-based configurator and (3) a method for applying the Genesis method in this domain. In this case the Genesis method is implemented by a Bayesian belief net that captures the designers' prior knowledge on how to configure PCs. Then instances of complete configurations are sampled from the model and added to the recommender algorithm.

First we will give a background to recommender systems, second we list some open and addressed research problems, third we present the used research methodology and last we present a reading guide and the contribution of each paper.

## 2 Recommender Systems

The rise of the Internet has led to information overload problems whether you want to find something using a search engine, buy something at an online store or read your own e-mail. In addition, many online retailers want to give their customers a good service to keep

them coming back but also to cross-sell products that give high revenues. Recommender systems are one way to tackle these problems.

**Definition 1** *A recommender system is a system that helps a user to select a suitable item among a set of selectable items using a knowledge-base that can be hand coded by experts or learned from recommendations from the users.*

This definition implies that the notion of recommendation holds the idea that it has been recommended by another human being (an expert or another user). Predictions induced only from the active user's previous interactions are excluded.

A learning recommender system typically works as follows: (1) the recommender system collects all given recommendations at one place and (2) thereafter it applies a learning algorithm. Predictions are then made either with a model learned from the dataset (model-based predictions) using, for example, a clustering algorithm [1, 7] or on the fly (memory-based predictions) using, for example, a nearest neighbor algorithm [1, 11]. A typical prediction can be a list of the top N recommendations or a requested prediction for a single item.

Memory-based methods store training instances during training that are then retrieved when making predictions. In contrast, model-based methods generalize into a model from the training instances during training and the model need to be updated regularly. The model is then used to make predictions. Memory-based methods learn fast but make slow predictions, while model-based methods make fast predictions but learn slowly.

The origin of recommender systems can be traced back to Malone et al. [3]. They proposed three forms of filtering: cognitive filtering (now called content-based filtering), social filtering (now called collaborative filtering) and economic filtering. They also suggested that the best approach is probably to combine these approaches (now called hybrid systems).

The term collaborative filtering was introduced by Goldberg et al. in their work with Tapestry [4]. Tapestry is regarded as the first recommender system [12].

Another early paper from 1990 by Karlgren describes an algebra for combining user opinions to make recommendations [31].

The term *collaborative filtering* was used for recommender systems until Resnick et al. in [12] introduced the latter and broader term. The reasons for the new term were two: (1) users do not necessarily collaborate explicitly with each other and (2) to include both filtering (removal of uninteresting items) and suggestions of interesting items. The new term includes any of the filtering approaches suggested by Malone et al. In addition, the focus is moved from the algorithm to include the user interface [2].

Burke [16] adds two other ways of making recommendations to the three already mentioned: knowledge-based and demographic-based recommender systems (as well as utility-based recommender systems

but they are regarded as a special case of knowledge-based recommender systems).

Since the term was coined in 1997 the field of recommender systems has grown a lot. There are systems for recommending movies [13], newspaper articles [8], Usenet news [11], humor [14], food recipes [9] and many others. A good overview of many commercial recommender systems can be found in [10].

## 2.1 Content-based filtering

Content-based filtering uses the content of the recommended items as basis for recommendations. This is the oldest approach to filtering [6]. For instance, when recommending news articles the words in the article might constitute the content attributes.

This approach suffers from the shortcoming of only being able to recommend items with content previously encountered (the serendipity problem). To give an example, when a user has only rated papers containing the term *collaborative filtering* and not *recommender systems*, only papers containing the previous term can be recommended. As in case for all systems learning from experience content-based systems take time to learn to make good recommendations (the cold-start problem or the bootstrapping problem). For instance, not until the user has rated a paper containing the term *recommender systems* can other papers containing that same term be recommended.

The systems in the first and the third paper describe recommender systems using content-based methods. In the first paper term vectors are used to match users' interests in news articles recommended by other users. The system in the third paper uses the attributes of PC configurations to match the current user with completed configurations from previous users.

## 2.2 Collaborative filtering

Collaborative filtering differs from content-based filtering in that user opinions are used instead of content. Each item is represented by the opinions given by users. Opinions can be expressed as explicit user ratings on some scale ranging from bad to good, or as implicit ratings given by logging users actions. As an example of the latter, viewing or skipping items could be interpreted as positive and negative ratings respectively. In addition, opinions can be expressed as text annotations attached to the items, allowing content-based methods to be applied. Thus for collaborative filtering the recommendation process is a social activity – collaborative filtering tries to automate word-of-mouth recommendations: you are recommended items that other people with similar taste have recommended [5].

The main advantage over content-based methods is that any items regardless of content can be recommended. Movies, images, art and text items are all represented by the users' opinions and thus can be recommended by the same system. If a content-based method was applied, only items with comparable content could be recommended.

This also means that collaborative filtering can recommend things from different genres. Assume that there is a user that likes thrillers but not science fiction. In a content-based approach the user will only get thrillers recommended. Whereas in a collaborative approach, a science fiction movie can be recommended as well given that there are enough science fiction lovers that also like thrillers.

Collaborative filtering suffers from different shortcomings than content-based filtering. There are foremost two bootstrapping problems: (1) if nobody has rated an item, it cannot be recommended (the early rater problem) and (2) if each user has rated very few items, users cannot be matched (the sparsity problem).

None of the system presented in this thesis uses pure collaborative filtering for making recommendations. All systems are hybrids. The system closest to using pure collaborative filtering is presented in the second paper. In this system, the actual recommendations are made by pure collaborative filtering with explicitly captured opinions. However, the system is bootstrapped with content-based artificial user profiles. The system of the first paper uses the user opinions to select which neighbors an agent should keep but not when making recommendations.

### 2.3 Economic filtering

Economic filtering moves the focus of filtering from the receivers to the senders by introducing cost to send recommendations. For instance, a user could be charged for the length of the sent recommendation (assuming that the user actively recommends something to another user). This form of filtering is not much used for recommender systems: the problem is usually not to prevent users from recommending, but to encourage them to do it. Thus instead economic means or some other reward could be used to create incentives for users to recommend. An instance of a system that uses virtual money called “chit” as payment and reward for receiving and giving recommendations is the Knowledge Pump described in [36].

### 2.4 Knowledge-based recommendations

Knowledge-based recommender systems use prior knowledge on how the recommended items meet the users’ needs. An instance of a knowledge-based recommender algorithm with collaborative filtering is presented in [15, 16].

The main advantage using a knowledge-based system is that there is no bootstrapping problem. Because the recommendations are based on prior knowledge there is no learning time before making good recommendations. This is also a drawback since pure knowledge-based systems can only make pre-coded recommendations and not adapt to the individual user or changing domains.

The closest to a knowledge-based recommender system is described in the third paper. The system uses a Bayesian belief net to represent uncertain prior knowledge on how people tend to put PCs together and

a rule-based configurator containing knowledge on how components can and cannot be combined.

## 2.5 Demographic-based recommendations

Demographic-based recommender systems use prior knowledge on demographic information about the users and their opinions for the recommended items as basis for recommendations. There are not many recommender systems using demographic data because this form of data is difficult to collect [16]. People are reluctant to share phone numbers, physical addresses and similar information with a system. This problem is somewhat easier for commercial systems that, for instance, collect credit card numbers when users make their orders. An example of a demographic-based recommender where information about users is taken from their homepages to get around this problem is described in [26]. No paper in this thesis makes use of demographic information.

## 3 Open and addressed research issues

Although much research has been conducted there are still many unsolved problems and open research questions. In this section we list current and future research problems and where our work makes contributions.

Schafer [17] identifies six research directions to be further investigated: hybrid systems, building community, explaining recommendations, validating recommendations, preserving privacy and mobile systems. In addition to these we would like to add peer-to-peer systems and systems in the context of ubiquitous computing, and the old research issue on how to bootstrap a new recommender system. The presented work contributes to the area of hybrid systems, peer-to-peer systems, bootstrapping, and partially to protecting user privacy. All presented systems in this work are hybrid systems. The system of the first paper is a peer-to-peer system and it is claimed that decentralization in itself helps to protect the user privacy. The other two papers address the problem of how to bootstrap a new recommender system. The bootstrapping problem was covered in the previous section that describes the different basis for recommendations and it is not listed here.

### 3.1 Hybrid Systems

Hybrid recommender systems combine two or more recommender algorithms. The reason is to make use of their combined strengths and to level out their corresponding weaknesses [16]. Most frequent is to combine collaborative filtering with content-based filtering.

Burke lists a number of hybridization methods to combine pairs of recommender algorithms [16]. There are weighted hybrids, switching hybrids, mixed hybrids, feature combination hybrids, cascade hybrids, feature augmented hybrids and meta-level hybrids. Each method is

Hybridization method	Description
Weighted	The scores (or votes) of several recommendation techniques are combined together to produce a single recommendation.
Switching	The system switches between recommendation techniques depending on the current situation.
Mixed	Recommendations from several different recommenders are presented at the same time.
Feature combination	Features from different recommendation data sources are thrown together into a single recommendation algorithm.
Cascade	One recommender refines the recommendations given by another.
Feature augmentation	Output from one technique is used as an input feature to another.
Meta-level	The model learned by one recommender is used as input to another.

Table 1: Different methods to make hybrids of pairs of recommender algorithm. The table is taken from [16], page 340.

shortly described in Table 1 (the table is taken from Burke). According to Burke’s classification there are totally 53 possible hybrids combining pairs of recommender algorithms with hybridization methods. Only 14 of these hybrids seem to be explored. Thus there are still 39 hybrids to be investigated. In addition, one can combine more than two recommender algorithms and then the number of possible hybrids increases even more.

The presented work contributes to this research problem by introducing several new combinations of recommender algorithms. The first paper describes a system that is a cascade hybrid. First coarse-grain filtering using opinions is applied to select who to keep as neighbor and thereby selecting who that will receive recommendations. Then fine-grained content-based filtering is applied to select what is actually recommended.

The second paper describes a system that uses collaborative filtering that is seeded with artificial user profiles. The profiles are sampled from a model built using content-based methods and prior knowledge. Thus this is a hybrid that combines collaborative filtering with a combination of content-based and knowledge-based filtering. The first combination corresponds to feature augmentation hybridization because the collaborative filtering uses the ratings of the artificial profiles. The second combination corresponds to weighted hybridization because the ratings of the artificial profiles consist of the multiplication of the rating contributions from the content-attributes and the rating contributions from the prior knowledge.

The third paper describes a system that combines collaborative filtering, content-based filtering and (two times) knowledge-based filtering. The content features are used to match the current user profile (configuration) with previous profiles (completed configurations) to generate recommendations. In addition, the recommender is seeded with artificial profiles generated from a Bayesian belief net built from prior knowledge. Thus the system uses a meta-level hybridization method when combining collaborative with content-based filtering and feature augmentation to add the prior knowledge. Finally the output from the recommender is combined with the output from the configurator using mixed hybridization presenting both outputs alongside each other.

### 3.2 Building community

Most recommender systems rely heavily on a large community of users for good performance. Without users there are no opinions and no recommendations. If users do not come back to use the system again, the system will only be able to make shallow predictions for ephemeral users. Thus building a durable community is very important and it is an essential research issue that should be further investigated.

Recommender systems were originally used to recommend items, but we can also imagine them recommending people. This will effectively turn a recommender system into a community-building tool, for instance, to find new friends or experts in some field. As well, in a community, groups of users might be interested in getting recommendations for the entire group based on the different group members' interests. Early examples of the former for finding experts are presented in [19, 20] and PolyLens, an instance of the latter is described in [21].

### 3.3 Explaining recommendations

Another important issue is how to explain a recommendation to a user. Explaining why an item was recommended to a person might be as important as getting the recommendation. The major work in this area is Herlocker's thesis where several different ways for explaining recommendations are described and tested on real users [18].

For hybrid recommender systems explaining recommendations is complicated. For instance in the second and last paper the system is seeded with artificial profiles. Thus a recommendation might not only come from another user but also from a randomly created profile. How is such a recommendation explained? In the third paper we briefly touch this research question.

### 3.4 Validating recommendations

Yet another issue is how to know whether a recommendation is valid. Does a recommendation meet the user's need? This is quite easy for a

system like MovieLens <sup>1</sup> that uses explicit ratings. You watch a movie and then you rate it. However, for systems using implicit ratings this is a hard problem. To give an example, when a rating corresponds to whether an item is bought or not – how does the system know that the customer really liked the bought item? One solution is to let the customer rate the item when returning to buy some more as done at Amazon.com.

This research problem was not addressed in this thesis though the issue is very relevant for the last paper describing a system for configuring and selling PCs. How can the system obtain the information on whether a PC configuration really meet the user’s need?

### 3.5 Preserving privacy

Most recommender systems require that some information about users is stored and used for making recommendations. This makes recommender systems a serious risk for violating the privacy of users. The information might be sold to a third party, or be used unexpectedly, for instance, for promoting other products [22].

Even though the data itself might not be misused, somebody can, by cleverly asking for recommendations, find the opinions of users that are differing significantly from most other users. As long as a user is similar to the other users he is safe but when making serendipitous ratings he is exposed. This problem is described and investigated in [23].

Kleinberg et al. make a game theoretical analysis of the value of sharing private information for recommender systems and for collaborative filtering in [25]. They believe the principle behind privacy is simply that users should only share their private information when they are fairly compensated.

An attempt to secure privacy for users of a decentralized match-making system is described by Foner in [24]. Foner argues that decentralization in itself helps preserving privacy.

Another decentralized, peer-to-peer system for preserving privacy is presented by Canny [37]. He describes a network of peers that uses a clever recommender algorithm. The algorithm consist of two parts: First user opinions are gathered by an aggregator that computes a model and then the resulting model is spread among the peers that use it to make recommendations. This is done in a clever way using encryption and other techniques that make no user aware of any other’s opinions.

Though privacy is not addressed directly in this thesis the first paper makes arguments taken from Foner that decentralization in itself makes it easier to preserve privacy.

---

<sup>1</sup>[www.movielens.org](http://www.movielens.org)

### 3.6 Mobile systems, Ubiquitous Computing and Peer-to-peer

In recent years mobile devices have become properties of each and everyone. In Sweden alone nearly 90 percent of the population have their own cellular phones [27]. Many also have notebook computers, PDAs and similar devices. The next step has been to connect these devices together to stationary PCs, local networks and other local resources. One such proposed approach for personal use is Bluetooth<sup>2</sup> and another complementing approach for local area use is WLAN<sup>3</sup>. The small size of most mobile devices places high demands on the user interface. It is harder to present and explain recommendations to the user on a small area. Two examples of recommender systems for mobile systems are presented in [38] and [39].

The introduction of mobile computers is only the beginning of a trend towards ubiquitous computing where computers are situated anywhere and everywhere [28, 29]. Ubiquitous computing is about making the computers invisible and part of the ordinary life in a similar way as it is with written text. Items with written text surround us but we are not bothered by it and we regularly use the text without thinking much about what we are doing. The aim is that the computer should be a tool that makes people focus on what they want to do rather than on the computer itself.

In ubiquitous computing the physical location of devices is of utmost importance. Users should interact with computer systems without even noticing, such as is already the case with for example washing machines, cars, and microwave ovens. By using the users' contexts such as their location these kinds of systems open up for new applications using location as base for adapting system behavior [34, 33]. In the case of mobile systems there are already some positioning systems in use, for instance, GPS. An example of a system using the user's location as well as content-based and collaborative filtering to filter digital notes is GeoNotes [30]. Another example is a shopping agent that only shows the items from the stores closest to the user's location [32].

Note however that the notion of having an intelligent assistant that can interrupt to tell me something is contrary to the notion of ubiquitous computing that tries to remove the focus from the computer.

Peer-to-peer is an alternative to the dominant client-server architecture now prevailing on the Internet. Though not necessarily a mobile or ubiquitous, peer-to-peer systems have some advantages in an environment penetrated by computers. One advantage is for instance to preserve privacy [24, 37]. In a peer-to-peer system, the peers' computers are connected in a network working together to achieve some common goal. No peer is in essence considered different from any other peer. Examples of peer-to-peer systems are ICQ<sup>4</sup> which is a chat

---

<sup>2</sup><http://www.bluetooth.com/>

<sup>3</sup>Wireless Local Area Network (<http://www.wlana.org/>)

<sup>4</sup><http://www.icq.com>. ICQ is an instant-messaging program that lets the users set up a buddy list with friends that can be contacted directly with an instant message.

system, and Kazaa<sup>5</sup>, Gnutella<sup>6</sup> and Napster<sup>7</sup> which are file-sharing systems. A prominent feature of a peer-to-peer system is that the peers can communicate directly with each other and not via a server. Gnutella and Kazaa work without any central server while Napster and ICQ use a central server to help the peers find each other. The first paper describes a peer-to-peer-based recommender system based on a multi-agent system.

Some inherent advantages with multiple agents are according to [35]:

- Speed through parallel computing
- Robustness through agents with redundant functions
- Scalability because of the modular approach.

The proposed peer-to-peer-based system will be robust to disappearing agents. If an agent disappears it stops making recommendations and thus it will soon be forgotten. In addition, if there are a large number of agents, another agent that share same interests can probably replace the disappeared agent. The scalability of the system is secured because it is easy to add a new user and by adding a new user, a new computer and its resources are added as well.

We believe that parallelism, robustness, scalability and privacy preserving are all important features for recommender systems in a mobile environment or in an environment of ubiquitous computing. In ubiquitous computing there will be many possible ways of detecting the user's actual context by communication with nearby devices. Thus collecting all user information at one single location to generate recommendations will not be feasible. In a decentralized system this will be much easier to handle.

## 4 Research Methodology

The research methodology of this work consists of three steps:

**Literature study:** First, interesting research problems are identified by literature studies and cross-fertilization between different research fields.

**Design and implementation:** Second, the identified research problems are described and followed by design and implementation of systems addressing the problems.

**Experimental evaluation:** Last experimental evaluations are performed to test whether the problems are solved. In the first paper the system is evaluated in a simulator with simulated users and in the second paper experiments are performed on an available dataset to test relevant aspects.

---

<sup>5</sup><http://www.kazaa.com>

<sup>6</sup><http://www.rixsoft.com/Knowbuddy/gnutellafaq.html>

<sup>7</sup><http://napstermp3.com/Webopedia.htm>

The three steps have of course had impact on each other. For instance, the identification of potential research problems naturally leads to more literature study on that specific topic. In addition, experimental evaluation of systems motivates better design and implementation.

## 5 Reading Guide and Contributions

This section consists of short presentations of papers I-III in Appendix A. The first paper can be read separate from the other two while the last two papers should be read together.

Paper I – “A Peer-to-Peer-based Recommender System for a News Headline Viewer” – is submitted to the workshop on Cooperative Information Agents (CIA), Helsinki, August 2003.

Paper II – “Genesis: A method for bootstrapping recommender systems using prior knowledge” – is submitted to a special issue of ACM Transactions on Information Systems, 2004.

Paper II is written together with Åsa Rudström at SICS<sup>8</sup>. I developed the initial ideas, performed the experiments and wrote the first draft. Åsa helped me to rewrite the paper and to develop the initial ideas into the Genesis method.

Paper III – “Enhancing Web-Based Configuration with Recommendations and Cluster-Based Help” – was presented at the Workshop on Recommendation and Personalization in eCommerce at the 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems, May 2002.

Paper III is written together with Rickard Cöster, Andreas Gustavsson and Åsa Rudström at SICS. My contributions to this paper are foremost the three recommender algorithms and the bootstrapping approach. The Genesis method was born out of this work.

### 5.1 A Peer-to-Peer-based Recommender System for a News Headline Viewer

Most of today’s recommender systems, as well as any web-based service, use a client/server architecture. The first paper describes a recommender system for information monitoring that uses an alternative peer-to-peer architecture. Predictions of recommendations are done locally at the user’s computer.

The proposed recommender system works as follows. Each user has a program that presents subscribed new headlines and information updates to the user as well as recommendations from a personal recommender agent.

The recommender agent learns to recommend based on what the user rates as interesting (using content-based filtering) and collaborates with other users’ recommender agents to find new items to recommend. Each recommender agent has a limited number of neighbors

---

<sup>8</sup><http://www.sics.se>

(users represented by their agents) to which it sends the user's recommendations. Received recommendations are then: (1) recommended to the user based on content similarities and (2) forwarded to the neighbors. The number of times a recommendation is forwarded is limited to two. Recommendations are used both as a means to find new items to recommend and to find new neighbors.

To shorten the path between similar users the agents try to form clusters with like-minded users. The clustering is done by keeping a set of good neighbors and set of potentially good neighbors. When a potentially good neighbor becomes good enough it replaces one of the good neighbors and vice versa.

A simulator was implemented to test the system. The users and news sources were simulated but the recommender agents were pretty much fully functioning recommenders. The news items were taken from the CLEF collection<sup>9</sup> and 10 news sources were randomly drawn from the collection. The search queries of the CLEF collection were used as user interests.

The peer-to-peer system was compared to a similar but centralized system with only one recommender agent serving all users. Three parameters were varied to test the *scalability*, the *clustering effect* and the *exploration effect*.

The scalability was tested by running the system with 100 and 400 users. The result indicated that the system scales.

The effect of clustering was tested by varying the number of kept good neighbors. We did not get much clustering of users with similar interests. This was probably because the neighbors were exchanged very fast making it hard to learn whether a neighbor in fact was good. Consequently, the clustering had little effect on the performance.

The effect of exploration was tested by varying the probability of adding a new neighbor regardless of having given a good recommendation. The ability to explore showed to be very important for the performance. No exploration meant worse performance than for the centralized system.

For all parameters it was easy to get a performance comparable with the centralized system.

## 5.2 Genesis: A method for bootstrapping recommender systems using prior knowledge

All recommender systems based on content-based filtering or collaborative filtering suffer from a bootstrapping problem. Initially there are no users and no rated items thus there is nothing to learn from. The prominent way of dealing with this problem is to combine content-based filtering and collaborative filtering using one of Burke's hybridization methods. However, the combination of content-based filtering and collaborative filtering still suffers from a bootstrapping problem since it still takes time to learn what to recommend.

---

<sup>9</sup><http://trec.nist.gov/data.html>

Recall that a knowledge-based recommender system does not have a bootstrapping problem. Thus a good idea would be to combine a learning recommender system with domain knowledge.

In order to deal with this problem we have developed the Genesis method. The method bootstraps a recommender system with artificial user profiles sampled from a probabilistic model built from prior knowledge on the recommender domain.

The Genesis method consists of three steps: domain analysis, model building, and user sampling. In the domain analysis, we first analyze the recommended items. Second, we investigate the user profiles. Third, we consider what kind of prior knowledge there is. The results from the domain analysis are used to build a set of probabilistic models reflecting the working of system users. A recommender system using any collaborative recommender algorithm might finally be bootstrapped with artificial user profiles sampled from the model. The paper describes how this was done for collaborative filtering in the movie domain using a k-nearest neighbor recommender algorithm.

Four different models of users' rating schemes were tested: Two basic rating schemes (uninformed bootstrapping), the random rating scheme and the constant rating scheme, and two extensions adding prior knowledge to the basic schemes (informed bootstrapping), the rank order-based extension and the cluster-based extension. In the random rating scheme each user was modeled as having a uniform randomly drawn rating for each movie, while in the constant rating scheme each user had a constant rating for all movies. In addition all users were modeled to have randomly drawn rating preferences, where some users might rate movies on the full scale from 0 to 5 (from low to high rating) while other users might rate movies only from 1 to 3 or even from 4 to 5.

The rank order-based extension adds the influence of rank order from three available top lists for the most popular movies from the Internet Movie Database (IMDb)<sup>10</sup> – the female top 50 list and the male top 50 list and the overall top 250 list. Movies with low rank order in the lists were given higher ratings from the users on average.

The cluster-based extension assumes that users like similar movies. Therefore the movies were clustered based on the text-content attribute plot (also available from the IMDb). A user was then modeled to like movies in two randomly drawn clusters.

The accuracy of the system was measured for single predictions and for ranked lists of recommendations. The system was tested with and without artificial user profiles.

**Uninformed bootstrapping.** Surprisingly we were able to improve the performance for single predictions using only the random rating scheme though the performance for the ranked recommendations was significantly lowered.

The constant rating scheme did not differ significantly from having no artificial profiles.

---

<sup>10</sup><http://www.imdb.com>

**Informed bootstrapping.** The addition of the rank order-based extension (for either of the two basic rating schemes) did not either affect the performance probably due to too few rank ordered movies in the top lists.

However, by adding the cluster-based extension to the constant rating scheme we were able to slightly improve the performance for single predictions and at the same time improve the performance for ranked recommendations over the random rating scheme. Though, the performance for the ranked recommendations is still significantly lower than for the system without artificial profiles.

### 5.3 Enhancing Web-Based Configuration with Recommendations and Cluster-Based Help

Most recommender systems recommend single-dimensional items such as books, movies and other products. The user makes one single selection of a complete item for which an explicit or implicit rating is given (thereby the term single-dimensional). In the last paper we have identified a new domain with multi-dimensional items – product configuration – where new recommender algorithms are needed.

Instead of selecting the complete item the user selects values for multiple components before finally selecting the complete item (thereby the term multi-dimensional). This resembles the multi-dimensional ratings used for the restaurant recommender Entree though most of our dimensions constitute real attributes of the product while Entree’s are entirely human defined ratings [16].

We have investigated the multi-dimensional item domain of PC configuration where users can choose what type of computer they want, how fast the computer should be, the size of the hard disk and many more features or components. The basic approach to assist users configuring their PCs is to have a rule-based configurator that constrains what selections are compatible. The configurator is extended with recommendations and a cluster-based help system. The recommender system adds social information based on what other users have chosen. The combination of the recommendations from the configurator and the recommender system introduces new complexity in the recommendation process.

The third paper makes the following contributions: (1) three new recommender algorithm on how to make recommendations in the new domain of product configuration, (2) a method for adding social recommendations to a rule-based configurator and (3) a method for applying the Genesis method in this domain. A prototype of the system was implemented as well.

The three recommender algorithms are (1) the Weighted Majority Voter, which predicts the value of a component on the basis of a weighted majority vote of  $k$  nearest neighbors using the number of common component features as vote function (2) the Naïve Bayes Voter that is similar to Weighted Majority Voter but uses the probabilities of component values given by a Naïve Bayes Classifier as vote function

and (3) the Most Popular Choice (a modified Naïve Bayes Classifier that takes the user’s current selections into count) that predicts the most popular entire configuration of the  $k$  nearest neighbors.

The recommendations from the configurator and the recommender system are mixed to use Burke’s vocabulary. Recommendations are presented alongside but not in the same way. Users select values for attributes from popup-menus showing all available values. In case of the configurator, recommended selections are marked with different colors depending on the meaning of the recommendation. For instance, red colored values indicate that they are not compatible with other selected values. In case of the recommender system, single predictions are given on demand at an area to the right by clicking on a button beside the popup-menu. Recommendations of entire configurations are given by clicking on a separate button that opens a new window that shows a full configuration. The attribute values can then be selected all at once.

The system is bootstrapped with the Genesis method. In this case Genesis is implemented by a Bayesian belief net that captures the designers’ prior knowledge on how to configure PCs. Then instances of complete configurations are sampled from the model and added to the recommender algorithm.

## 6 Concluding remarks

In this thesis we have given a background to recommender systems and listed a number of research questions. We have also listed our contributions to each research question.

In Paper I we presented a decentralized peer-to-peer-based recommender system that achieved a performance comparable to a similar but centralized system. We also argued that decentralized systems are very suitable for ubiquitous computing and mobile systems.

Papers II and III address the bootstrapping problem when the recommender system lacks experience to learn from. As a solution we proposed the Genesis method that consist of three steps: (1) domain analysis, (2) model building and (3) user sampling. First a probabilistic model of the users’ rating behavior is built from prior knowledge. Then a recommender system is bootstrapped with artificial user profiles drawn from the model. In Paper II, the Genesis model was applied to the movie domain. We were able to improve the average prediction accuracy for single movies but for rank list of recommendations the average accuracy was decreased. In Paper III, we described how to apply the Genesis method for PC configuration using a Bayesian belief net.

## References

- [1] Breese, J. S., Heckerman, D. and Kadie, C. 1998. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. MSR-TR-98-12. May (revised October 1998).
- [2] Soboroff, I., Nicholas, C., and Pazzani, M. 1999. Workshop on recommender systems: algorithms and evaluation. ACM SIGIR Forum, Volume 33 , Issue 1, pp. 36 - 43, Fall 1999.
- [3] Malone, T.W., Grant, K.R., Turbak, F.A., Brobst, S.A. and Cohen, M.D. 1987. Intelligent Information Sharing Systems. Communications of the ACM, 30, 5, pp.390-402.
- [4] Goldberg, D., Nichols, D., Oki, B. and Terry, D. 1992. Using Collaborative Filtering to Weave an Information Tapestry. Communications of the ACM, Vol. 35, No. 12, pages 61-70, December.
- [5] Shardanand, U. and Maes, P. 1995. Social Information Filtering: Algorithms for Automating "Word of Mouth". CHI '95 Proceedings: Conference on Human Factors in Computing Systems: Mosaic of Creativity Sponsored by SIGCHI May 7 -11 1995, Denver, CO. ISBN: 0-89791-694-8.
- [6] Oard, D.W. 1997. The State of the Art in Text Filtering, User Modeling and User Adapted Interaction, 7(3)141-178, 1997.
- [7] Ungar, L.H. and Foster, D.P. 1998. Clustering Methods for Collaborative Filtering. AAAI Workshop on Recommendation Systems.
- [8] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D. and Sartin, M. 1999. Combining Content-Based and Collaborative Filters in an Online Newspaper. ACM SIGIR Workshop on Recommender Systems, Berkeley, CA, August 19.
- [9] Svensson, M., Laaksolahti, J., Höök, K. and Waern, A. (2000) A Recipe Based On-line Food Store, In Proceedings of the 2000 International Conference on Intelligent User Interfaces (IUI'2000), New Orleans, Louisiana, USA, 260 - 263.
- [10] Schafer, J.B., Konstan, J.A. and Riedl, J. 2001. E-commerce recommendation applications. Data Mining and Knowledge Discovery , vol. 5 nos. 1/2, pp 115-152.
- [11] Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P., and Riedl, J. 1994. GroupLens: An open architecture for collaborative filtering of netnews. Proceedings of the 1994 Computer Supported Collaborative Work Conference.
- [12] Resnick, P. and Varian, H. 1997. Recommender Systems. Communications of the ACM, 1997, 40(3), pp. 56-58.
- [13] Good, N., Schafer, J., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J. and Riedl, J. 1999. Combining collaborative filtering with personal agents for better recommendations. In Proceedings of the Sixteenth National Conference on Artificial Intelligence.

- [14] Goldberg, K., Roeder, T., Gupta, D. and Perkins, C. 2001. Eigentaste: A Constant Time Collaborative Filtering Algorithm, *Information Retrieval Journal*,4(2), pp. 133-151. July.
- [15] Burke, R. 2000. Knowledge-based Recommender Systems. In A. Kent (ed.), *Encyclopedia of Library and Information Systems*. Vol. 69, Supplement 32. New York: Marcel Dekker.
- [16] Burke, R. 2002. Hybrid Recommender Systems: Survey and Experiment User Modeling and User-Adapted Interaction 12(4): 331-370; Nov 2002
- [17] Schafer, J.B. 2002. Emerging Internet Applications. DoDEA/UNI Computer Science Institute, July 22-26, University of Northern Iowa, Cedar Falls, IA.  
URL: <http://www.cs.uni.edu/~schafer/courses/dodea/>  
Accessed: 28 February, 2003
- [18] Herlocker, J. 2000. Understanding and Improving Automated Collaborative Filtering Systems. Ph.D Dissertation, University of Minnesota, 2000
- [19] Kautz, H., Milewski, A. and Selman, B. 1996. Agent Amplified Communication. In the Proceedings of AAAI-96.
- [20] Kautz, H., Selman, B. and Shah, M. 1997. Referral Web: combining social networks and collaborative filtering. *Communications of the ACM*, Volume 40, Issue 3 (March 1997). pp. 63 - 65.
- [21] O'Connor, M., Cosley, D., Konstan, J. A., and Riedl, J. 2001. PolyLens: A Recommender System for Groups of Users. In Proceedings of ECSCW 2001, Bonn, Germany, pp. 199-218.
- [22] Riedl, J. 2001. Personalization and Privacy. *IEEE Internet Computing Online*, November/December 2001 (Vol. 6, No. 6).
- [23] Ramakrishnan, N., Keller, B.J., Mirza, B.J., Grama, A.Y. and Karypis, G. 2001. Privacy Risks in Recommender Systems. *IEEE Internet Computing Online*, November/December 2001 (Vol. 6, No. 6). pp 54-63.
- [24] Foner, L.N. 1999. Political Artifacts and Personal Privacy: The Yenta Multi-Agent Distributed Matchmaking System. Ph.D. Thesis. Massachusetts Institute of Technology.
- [25] Kleinberg, J., Papadimitriou, C.H. and Raghavan, P. 2001. On the value of private information. In Proc. 8th Conf. on Theoretical Aspects of Rationality and Knowledge (TARK).
- [26] Pazzani, M. 1999. A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review*. 13(5-6) 393-408
- [27] MTB Mobiltelebranschen. 2003. Mobiltelebranschens prognos för 2003. Press release 030226.  
URL: <http://www.mtb.se/Anslagstavla/Innehall/pressrelease.cfm?AnslID=736> Accessed: 4 March, 2003.
- [28] Weiser, M. 1991. The Computer for the Twenty-First Century. *Scientific American*, pp. 94-10, September 1991

- [29] Weiser, M. 1993. Some Computer Science Problems in Ubiquitous Computing. *Communications of the ACM*, July 1993. (reprinted as "Ubiquitous Computing". *Nikkei Electronics*; December 6, 1993; pp. 137-143.)
- [30] Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E. and Bylund, M., *GeoNotes: Social and Navigational Aspects of Location-Based Information Systems*, in Abowd, Brumitt & Shafer (eds.) *UbiComp 2001: Ubiquitous Computing*, International Conference Atlanta, Georgia, September 30 - October 2, Berlin: Springer, 2001, p. 2-17.
- [31] Karlgren, J. 1990. *Syslab Working Paper 179*, Department of Computer and System Sciences, Stockholm:Stockholm University
- [32] Fano, A.E. 1998. *Shopper's eye: using location-based filtering for a shopping agent in the physical world*. In *Proceedings of the second international conference on Autonomous agents*, Minneapolis, Minnesota, United States
- [33] Korkea-aho, M. 2000. *Context-Aware Applications Survey*. *Internetworking Seminar (Tik-110.551)*, Spring 2000. Helsinki University of Technology
- [34] Chen, G. and D. Kotz. 2000. *A Survey of Context-Aware Mobile Computing Research*. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.
- [35] Stone, P. and Veloso, M. 2000. *Multi-agent Systems: A survey from a Machine Learning Perspective*. *Autonomous Robots*, 8(3):345-383, Kluwer Academic Press.
- [36] Glance, N., Arregui, D. and Dardenne, M. 1999. *Making Recommender Systems Work for Organizations*. *Proceedings of PAAM'99*, London, UK, April 19-21.
- [37] Canny, J. 2002. *Collaborative Filtering with Privacy*, *IEEE Conf. on Security and Privacy*, Oakland CA, May.
- [38] Tveit, A. 2001. *Peer-to-peer based recommendations for mobile commerce*. *Proceedings of the first international workshop on Mobile commerce 2001*. pp 26 - 29. ACM Press New York, NY, USA.
- [39] Miller, B.N., Albert, I., Lam, S.K., Konstan, J.A. and Riedl, J. 2003. *MovieLens Unplugged. Experiences with Recommender Systems on Four Mobile Devices*. In *Proceedings of the 2003 international conference on Intelligent user interfaces*, ACM Press New York, NY, USA

## A Papers



## A.1 Paper I



# A Peer-to-Peer-based Recommender System for a News Headline Viewer

Tomas Olsson  
Swedish Institute of Computer Science  
Box 1263, SE-164 29 Kista, Sweden  
tol@sics.se

April 28, 2003

## Abstract

Today most recommender systems use a centralized client-server architecture where there is one service provider and many users. However, this is not the only possible architecture. In this paper we propose a decentralized peer-to-peer architecture with recommender agents serving a community of users. The peer-to-peer-based recommender is compared with a similar but centralized recommender in terms of precision, recall and utility.

In a news domain we were able to show that the decentralized approach performs equally well with respect to recall and utility but not with respect to precision.

## 1 The Scope of a Decentralized Recommender

Today most recommender systems use a centralized client-server architecture where there is one service provider and many users. Generally a recommender system collects all user opinions at one place and thereafter it applies a learning algorithm. Predictions are made either with a model learned from the data set (model-based predictions) using for example a clustering algorithm [3], [10] or on the fly (memory-based predictions) using for example a nearest neighbor algorithm [3], [11]). Model-based algorithms learn to predict prior to making predictions while memory-based algorithms predicts on the basis of some retrieved training examples at the time of prediction. However, a centralized architecture is not the only possible or the most suitable solution.

For inherently decentralized domains such as WWW or when the users' privacy is of great importance a centralized approach might not be the best choice. By storing all user preferences at one place the recommender system might easily yield all of the data to unauthorized intruders or even more likely, lead to unauthorized access and use by the service provider [1].

Yet another problem for centralized recommender systems is that the users are solely dependent on one service provider. In contrast the service of a decentralized approach will, if designed well, exist even if the original provider vanishes.

In response to these issues we propose a peer-to-peer recommender system consisting of recommender agents serving a community of users. A decentralized solution: (1) Will be running independent of a single service provider, (2) will protect the privacy of the users

by distributing the information about the users between the peers (each peer gets only a part of the information, not all information).

Though privacy is a major reason for building decentralized solutions it is not explicitly addressed in this paper. Instead we will focus on the question of how well a peer-to-peer recommender performs compared to a similar but centralized version. The system is tested in a hypothetical news domain with simulated users but with quite realistic recommender agents. We show that the performance is comparable for recall and utility but not for precision.

First the background of the proposed systems is presented, second the ideas behind the proposed recommender system is described, third the system implementation is presented and last experimental results are presented and discussed.

## 2 Background and Related Work

The proposed system intersects mainly with three areas of research: (1) it is a *recommender system* (2) with a decentralized *peer-to-peer architecture* (3) that uses distributed *software agents*. In the following each area is shortly described and related research is presented.

**Recommender systems.** A recommender system is a program that helps a user to select an item among many selectable items by means of some knowledge base either handcrafted or learned using some machine learning algorithm or a statistical model. Recommender systems have been implemented for several domains. There are recommender systems for recommending movies [7], newspaper articles [13], Usenet news [11], humor [12], food recipes [14] and many others. A good overview of many recommender systems can be found in [8]. All these systems use client-server architectures.

**Peer-to-peer systems.** A recent alternative to the dominating client-server architecture is a peer-to-peer architecture where peers act as both servers and clients. No peer is in essence considered different than any other peer. Examples of peer-to-peer systems are ICQ<sup>1</sup> which is a chat system, and Kazaa<sup>2</sup>, Gnutella<sup>3</sup> and Napster<sup>4</sup> which are file sharing systems. The prominent feature of a peer-to-peer system is that the peers can communicate directly with each other and not via a server. Gnutella and Kazaa work without any central server while Napster and ICQ use a central server to help the peers find each other.

**Software agents.** A software agent (or agent for short) is a software program that proactively or autonomously performs its work. A good introductory paper to the concept of software agents is [15] and [16] gives a good overview of the current research.

The recommender agents in the proposed system can be called intelligent information agents or interface agents because they use learning algorithms to handle information on the behalf of their users and they adapt to and interact directly with their users. They also constitute a so-called multi-agent system because they interact with each other and they are collaborative agents because they proactively cooperate to make good recommendations.

---

<sup>1</sup>ICQ is a trademark of ICQ Inc. ([www.icq.com](http://www.icq.com)). ICQ is an instant-messaging program that lets the users set up a buddy list with friends that can be contacted directly with an instant message.

<sup>2</sup>©Sharman Networks 2002 - All Rights Reserved ([www.kazaa.com](http://www.kazaa.com)).

<sup>3</sup><http://www.rixsoft.com/Knowbuddy/gnutellafaq.html>

<sup>4</sup><http://napstermp3.com/Webopedia.htm>

## 2.1 Related Work

The idea of using collaborating agents in a filtering task has been proposed in [6]. This paper describes a system with interface agents that help users sort e-mail. The agents' advice is taken either from learned user models or from asking other agents for advice. A problem not addressed is how the agents are supposed to find each other.

The issue of finding other peers was addressed in Yenta [1]. Yenta is a decentralized matchmaking system where agents search for users with similar interests using referrals from other agents. Both Yenta and our approach address the problem of how agents can find each other without any central control. In both approaches agents are expected to self-organize into clusters representing users with similar interests. In both approaches, this self-organization is achieved in the same manner humans find other people with similar interests — by referrals based on knowledge about others. However, Yenta assumes that it is non-problematic to directly compare user models while the agents of our system can have non-comparable user models because they implicitly share user models by exchanging recommendations. In addition, instead of explicitly asking other agents for referrals we use forwarding of recommendations as implicit means to find new neighbors.

A peer-to-peer recommender system is described in [2]. In this system the complete user profiles of rated items are sent between agents as queries in a Gnutella-like fashion, whereas we only send a single item with corresponding rating. Another difference is that their recommendations are computed locally using a nearest neighbor algorithm based on opinions, while we use content-based predictions. In addition, the agents of our system use opinions to actively select which neighbors to keep while they use the rather static infrastructure of Gnutella.

## 3 The Proposed Recommender

The present work is based on a hypothetical scenario of introducing a software program for information monitoring at all levels of an organization.<sup>5</sup> The software resides on a user's computer and keeps the user updated on changes in online newspapers and web pages. The information is shown as headlines in a small configurable window. All processing, for example checking updates, is done locally by the client. Since there is no central server the domain is well suited for a decentralized recommender approach. The advantage compared to using a client-server solution is that no more than the information monitoring software in combination with the recommender agent has to be installed (the former is installed anyway).

### 3.1 The Recommender System

To solve the problem of recommending news items in a decentralized domain we propose a recommender system consisting of interconnected peers of recommender agents. The agents watch over the shoulders of their users (called the monitored users) and learn what news articles the monitored users find relevant. That knowledge is then used to find other relevant articles in collaboration with the other agents. However, each agent can only collaborate directly with its closest neighbors and hence to reach agents beyond the closest

---

<sup>5</sup>The described system is intended to be incorporated with the BotBox Personal Assistant for information monitoring developed by BotBox AB ([www.botbox.com](http://www.botbox.com))

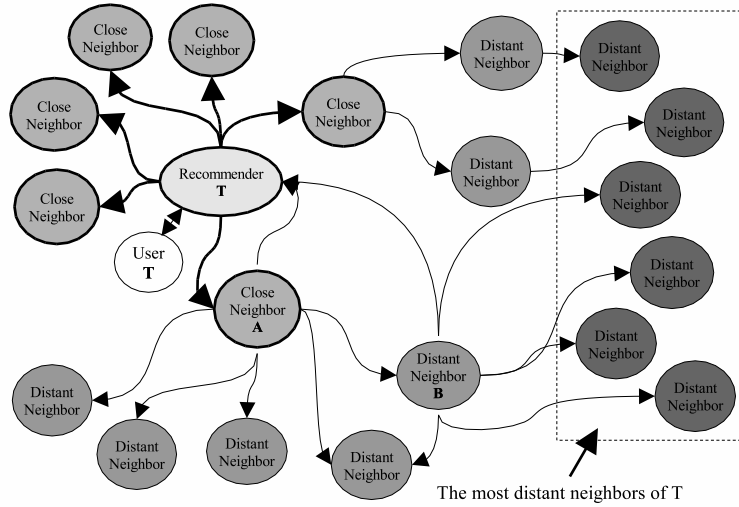


Figure 1: A part of a completely decentralized recommender of interconnected peers of recommender agents. In this figure each recommender agent (the ellipses) corresponds to a single user. An agent can communicate directly with a couple of (close) neighbors whereas more distant neighbors can be reached via the close neighbors. The arrows indicate the direction of communications and thus which agent that is modeling which neighbor.

neighborhood they have to forward recommendations for each other. This is illustrated in Figure 1.

We also want the agents to form clusters representing users with overlapping interests. The paths between agents representing similar interests should be shorter within a cluster than outside it. Thus recommendations should be spread faster with less communication and at the same time reach more interested users. This is expected to increase the performance compared to having no clusters.

Another important reason for agents to form clusters is that the resources of an agent are limited because most users' computers have limited resources. No agent should have to (or be able to) model all other users and thereby have all users as its neighbors. This becomes even more important when considering cell phones or other handheld devices.

This setup leads to the following problems that must be solved to get a working peer-to-peer recommender:

1. The peers have to collaborate,
2. The peers should be able to filter the user recommendations,
3. The peers must be able to find each other,
  - Initially (the bootstrapping of neighbors),
  - When the system is running (the clustering of peers),
  - While still keeping the network of peers connected.

Given that these implementation issues are solved we need to test the system to make sure

that this communication between agents will not affect performance too much. In the rest of this section will show how to solve these problems given a set of assumptions

**Peer-to-Peer Collaboration.** For the system to work it is assumed that users have distinct (Boolean) interests and that they are willing to share what documents (news items in our domain) they find interesting. The agents collaborate by exchanging the users' recommendations. The agents send recommendations to each other not on request but whenever a user has shown interest in a document. Thus only positive feedback is used. This means that only what the users like is forwarded to their agents and thereby to their agents' neighbors. A motivation for only using positive relevance feedback in a recommender system is that the meaning of negative feedback is unclear. It is hard to know why somebody did not like a certain item.

**The Filtering Mechanism.** The described system has three levels of filtering. The first level of filtering is user-based. Only documents that a user finds interesting are forwarded to its agent. The second level of filtering is between the agents based on their models of each other, both when deciding whom should receive a recommendation and whom to keep as a neighbor. The third level is when an agent decides whether to recommend a document to its user. This three-leveled filtering guarantees the user that a document is only recommended if another user has deemed it relevant.

**Initial Bootstrapping.** The agents are bootstrapped with neighbors by a peer-to-peer mechanism similar to ICQ with a buddy list and the possibility to send instant messages. The idea is that users usually have a social network of friends and co-workers whom they can add to their buddy lists. The buddies can then be used as the agents' neighbors.

**Finding New Neighbors.** Additional neighbors can be found either by receiving forwarded recommendations from other agents or whenever new buddies are added to the buddy list. However, the latter mechanism is not used in this paper.

**Clustering Peers.** It is also assumed that the interests of the users overlap. This implies that the agents by performing some type of hill climbing can find new neighbors more similar to their monitored users. The same assumption is made in Yenta that uses a similar algorithm. In the current system, the agents perform random search to find better neighbors.

## 3.2 Experiments

To test the system we ran several experiments. In the experiments the performance of a decentralized approach is compared to a similar but centralized one. The two systems differ in that the centralized system only has one recommender agent serving all users while the decentralized version has several agents, each only serving one user.

In addition, the number of neighbors per agent is limited to prevent the decentralized version from practically becoming identical to the centralized version where all agents serve all users. This implements the idea from above that no node should have access to all user data and that the agents have limited amount of resources.

## 4 The Recommender System Implementation

Recommender agents are implemented with a learning algorithm to model a set of monitored users and with a set of neighbor users (differing from the monitored users and initially taken from their buddy lists). Note that internally agents can use different algorithms. But by using the same algorithm we make it easier to evaluate the results. The learned models are used to predict the relevance of documents for the monitored users but also to choose

whom to keep as neighbors. The agents also work as brokers for each other making it possible to find new users by exchanging recommendations. The number of agents reached by the recommendations is limited.

#### 4.1 The Recommender Agent

The recommender agents are implemented to be fully functioning recommenders but with a rather simple learning algorithm for building user models. In addition, restricting the number of neighbors to maximally six simulates the limited resources of the agents. In a real setting the number of neighbors should be larger but not necessarily unlimited.

**Keeping users.** As previously mentioned, each recommender agent has a set of monitored users and a set of neighbors. An agent works as a personal recommender for its monitored users while a neighbor is a user using another agent as personal recommender. The neighbor relation is one way (see direction of arrows in Figure 1). However, the exchange of recommendations with a neighbor is always done via its recommender agent and not directly with the neighbor. In a completely decentralized system each agent has only one monitored user and in a completely centralized system there is only one agent that monitors all users.

**Building user models.** Each monitored user and each neighbor is represented by a user model. The user model of a monitored user consists of three parts: the vector space model (described below), a relevance threshold and a memory of relevant document as well as a memory of seen documents. The model of a neighbor only consists of a memory of relevant documents.

**Finding neighbors.** An agent can only send its monitored users' recommendations and forward other users' recommendations to its neighbors. The only way to receive recommendations is from agents having at least one of the monitored users of the receiving agent as a neighbor. In addition agents send recommendations to tell other agents of their existence and they use received recommendations forwarded from users not among the neighbors as means to find new neighbors. This means that it might be better for agents to have neighbor users that share the same interests as the agent's own, monitored users. As a result, the agents search for good neighbors and discard bad neighbors and thereby (hopefully) form clusters of like-minded individuals. This is supposed to shorten the paths for forwarded recommendations and hence improve the system performance. The choice of neighbors is thus very important for the individual agent but even more important for the entire network. The network should stay fully connected and not become divided into smaller parts that hinder the agents to communicate with distant agents. However, by using a naive strategy of only keeping good neighbors, the network might, given enough time, split into small, closely coupled clusters of interconnected neighbors. Therefore a clever selection mechanism must be chosen to keep the network connected while still letting the agents form clusters.

**Forming clusters.** We try to solve the problem of finding better neighbors by letting each agent have two sets of neighbors, a set of good neighbors and a set of potentially good neighbors. When an agent receives a forwarded recommendation, the recommending user might be added to the set of potentially good neighbors and replace an old potentially good neighbor if the set is full. As soon as a potentially good neighbor is better than one of the good neighbors, they can change places. This will hopefully solve the problem of finding and keeping good neighbors.

**Staying connected.** To partly solve the second problem of keeping the network connected we let the neighbors represent groups of similar neighbors instead of single users. The assumption is that two neighbors having similar user models are probably connected

via other agents too. Thus, when adding a new neighbor to the full set of potentially good neighbors, instead of just removing a random neighbor one of the two most similar neighbors are discarded. However, instead of throwing away all information about the discarded neighbor its user model is merged with the model of the other neighbor. Thus the resulting model will represent both of them. This procedure keeps the set of neighbors diverse while still letting agents keep good neighbors and thus form clusters of like-minded users.

Nevertheless, this approach only reduces the probability for the network to fall apart. To ensure that the network really stays connected another solution is needed such as a central repository where the agent can register to be found. Though this is contrary to the notion of decentralization.

## 4.2 The Document Memory

The document memory of a neighbor or a monitored user  $a$  for agent  $i$  is a column vector  $\mathbf{d}_a^i$  of relevant documents where  $d_{a,j}^i = 1$  and  $d_{a,j}^i = 0$  means that the document  $j$  is relevant for user  $a$  respectively not rated by user  $a$ . For each monitored user  $a$  there is also a memory of seen documents. The set of seen documents is denoted  $\mathcal{D}_a^i$ .

## 4.3 The Vector Space Model

The vector space model is well known from Information Retrieval (Baeza et al. 1999). We have used a variant with only positive examples (news articles considered relevant by a user) and with tf.idf vectors. The tf.idf is a technique to decrease the influence of common words in a word frequency vector by multiplying each word weight (= term frequency, tf) with the inverse document frequency (= idf).

For an agent  $i$  we denote  $\mathbf{M}_a^i$  the diagonal matrix with the idf of each term for the model of user  $a$  in the diagonal row. As idf of a term we use  $\ln(\frac{N_a^i}{n_w})$  where  $\ln$  is the natural logarithm,  $N_a^i$  is the total number of documents in the collection and  $n_w$  is the number of documents in the collection containing the given term  $w$ . We consider the set of relevant documents of each user  $a$  as a collection of its own.

$\mathbf{m}_a^i = \mathbf{M}_a^i \times \mathbf{s}_a^i$  is the vector space model of user  $a$  for agent  $i$ .

$\mathbf{s}_a^i = \sum_{d_{a,j}^i=1} \mathbf{v}_j$  is the sum of the word vectors of all relevant documents of user  $a$  known by agent  $i$ .

$\mathbf{v}_j$  is the word frequency vector of document  $j$ .

The vector space model is thus easily updated by adding the word vector of a new document to the previous model. The resulting vector is known to be an approximation to the best separator between relevant and irrelevant documents for query expansion (Baeza et al. 1999).

## 4.4 The Relevance Prediction

A document is deemed relevant for user  $a$  when the cosine similarity of an incoming document's word vector  $\mathbf{v}_j$  with idf and a vector space model  $\mathbf{s}_a^i$  is greater than a decision threshold  $t_a^i$ , that is,

$$\text{sim}(\mathbf{M}_a^i \times \mathbf{v}_j, \mathbf{m}_a^i) > t_a^i$$

The cosine similarity measure is the cosine of the angle between two vectors defined as:

$$\text{sim}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v}^T \mathbf{w}}{|\mathbf{v}| |\mathbf{w}|}$$

$\mathbf{v}$  and  $\mathbf{w}$  are column vectors and  $|\mathbf{v}| = \sqrt{\mathbf{v}^T \mathbf{v}}$  where  $T$  is the transpose operator.

When a user deems a document relevant, the threshold is updated by adding the old threshold together with the difference between the document's cosine measure in the relevance prediction and the old threshold weighted with a learning rate. The decision threshold for user  $a$  with a learning rate of 0.5 is thus updated by:

$$t_a^i \leftarrow \min(0.5, t_a^i + 0.5 \times (\text{sim}(\mathbf{M}_a^i \times \mathbf{v}_j, \mathbf{m}_a^i) - t_a^i))$$

The decision threshold is initially set to zero and the threshold will not get greater than 0.5 to prevent the relevant decision to be too specific and hinder the exploration of new documents. This means that in the beginning no documents will be recommended because  $\mathbf{m}_a^i = \mathbf{0}$  that leads to a  $\text{sim}(\cdot) = 0$  that is not larger than the threshold  $t_a^i = 0$ . As soon as a document has been rated relevant both the thresholds and the model are updated and after that recommendations can be given.

## 4.5 The Set of Neighbors

The neighbors of an agent  $i$  are divided into two sets, the set of good neighbors  $\mathcal{G}_i$  and the set of potentially good neighbors  $\mathcal{P}_i$ . The set of users monitored by agent  $i$  is denoted  $\mathcal{A}_i$ .

A new user is added to the neighbor set either (1) when giving a recommendation deemed relevant for any monitored user or (2) randomly with a certain probability of exploration. The probability of exploration lets an agent add neighbors that have not given a relevant recommendation and thus explore these neighbors neighborhood for additional better neighbors.

As long as there is room for more neighbors, any new neighbor  $l$  is added, first to the set of good neighbors and when it is full to the set of potentially good neighbors. However, because the number of neighbors is limited it is necessary to replace an old neighbor. This is done in three steps:

1. **Moves between sets.** The good neighbor least similar to and the potentially good neighbor most similar to any of the monitored users are found. If the potentially good neighbor is more similar to the monitored users than the good neighbor, then the potentially good neighbor will replace the good neighbor and vice versa in each neighbor set. The similarity comparison between the models is done by using the cosine measure with the number of common documents as the dot product.
2. **Merging neighbors.** The two most similar potentially good neighbors are found and one of them is randomly selected to be merged with the other one. The remaining potentially good neighbor will represent both of them. The merge is done by adding all the relevant documents of the selected user to the model of the other user. This is done as follows,

$$\{p_1, p_2\} = \text{argmax}_{n_1, n_2 \in \mathcal{P}_i} (\text{sim}(\mathbf{d}_{n_1}^i, \mathbf{d}_{n_2}^i))$$

$X \sim R(0, 1)$ , a uniform random number over  $[0, 1]$ .

**if**  $X \leq 0.5$  **then**

$$\mathcal{P}_i \leftarrow \mathcal{P}_i - p_1, \mathbf{d}_{p_2}^i \leftarrow \mathbf{d}_{p_2}^i + \mathbf{d}_{p_1}^i \text{ and } \mathbf{d}_{p_1}^i \leftarrow \mathbf{0}$$

**else**

---

**Algorithm 1 – rememberShownDocument( $a, j$ )**

---

**Comment:** The agent  $i$  is told that document  $j$  has been seen by the monitored user  $a$  (see section 4.2)

**Require:**  $i$  is the called agent

**Require:**  $j$  is a seen document

**Require:**  $a$  is a monitored user ( $a \in \mathcal{A}_i$ )

$$\mathcal{D}_a^i \leftarrow \mathcal{D}_a^i \cup \{j\}$$

---

$$\mathcal{P}_i \leftarrow \mathcal{P}_i - p_2, \mathbf{d}_{p_1}^i \leftarrow \mathbf{d}_{p_2}^i + \mathbf{d}_{p_1}^i \text{ and } \mathbf{d}_{p_2}^i \leftarrow \mathbf{0}$$

**end if**

3. **Adding neighbor.** The new neighbor  $l$  is added to the set of potentially good neighbors and a new user model is created.

$$\mathcal{P}_i \leftarrow \mathcal{P}_i - l$$

$$\mathbf{d}_i^j \leftarrow \mathbf{0} \text{ and } d_{i,j}^i \leftarrow 1 \text{ (} j \text{ is the recommended relevant document).}$$

---

**Algorithm 2 – relevantDocument( $a, j$ )**

---

**Comment:** The agent  $i$  is told that document  $j$  is relevant for the monitored user  $a$

**Require:**  $i$  is the called agent

**Require:**  $j$  is a relevant document

**Require:**  $a$  is a monitored user ( $a \in \mathcal{A}_i$ )

- 1: Update the user model of user  $a$  {see section 4.2, section 4.3 and section 4.4}
  - 2: **for all**  $u \in (\mathcal{A}_i - a)$  **do** {Recommend the document to the monitored users that probably deem document  $j$  relevant and have not already seen it}
  - 3:   **if**  $\text{sim}(\mathbf{M}_u^i \times \mathbf{v}_j, \mathbf{m}_u^i) > t_u^i$  **and**  $j \notin \mathcal{D}_u^i$  **then**
  - 4:     Add  $j$  to the unread document queue of user  $u$ .  
    {The users will read the document in next iteration of the main algorithm. In a completely decentralized system this step is never reached because each agent only has one monitored user.}
  - 5:   **end if**
  - 6: **end for**
  - 7: **for all**  $n \in \mathcal{G}_i \cup \mathcal{P}_i$  **do** {Forward the recommendation to all neighbor agents}
  - 8:   **call** agent of user  $n$  : **receiveRecommendation**(<  $i, a, j, n, 2 - 1$  >)  
    {This step is never reached in a centralized system since there is only one agent with no neighbors.}
  - 9: **end for**
- 

## 4.6 The Recommender Agent Algorithm

A recommender agent can communicate with its monitored users and with other recommender agents. In the centralized version, there is no communication with other agents because there is only one single agent.

---

**Algorithm 3 – receiveRecommendation**( $\langle s, a, j, n, ttl \rangle$ )

---

**Require:**  $i$  is the called agent.

**Require:**  $s$  is the sending agent.

**Require:**  $a$  is the recommending user

**Require:**  $j$  is the recommended document

**Require:**  $n$  is a receiving, monitored user ( $n \in \mathcal{A}_i$ )

**Require:**  $ttl$  is an integer ( $ttl \geq 0$ )

```
1: if  $a \in \mathcal{G}_i \cup \mathcal{P}_i$  then {The recommending user is a neighbor}
2:   Update the user model of user  $a$  {see section 4.2, section 4.3 and section 4.4}
3: else if  $a \notin \mathcal{A}_i$  then {The recommending user is not a monitored users}
4:   if  $(\exists u \in \mathcal{A}_i \text{ and } (\text{sim}(\mathbf{M}_a^i \times \mathbf{v}_j, \mathbf{m}_a^i) > t_a^i) \text{ or } X \sim R(0, 1) > \text{Exploration Probability})$  then {The document is relevant for any monitored user or by chance}
5:     Add user  $a$  to the neighbors (replace an old neighbor if necessary) {See section 4.5}
6:   end if
7: end if
8: if  $ttl > 0$  then
9:   for all  $k \in \mathcal{G}_i \cup \mathcal{P}_i$  do {Forward the recommendation to all neighbors}
10:    call agent of user  $k$  : receiveRecommendation( $\langle i, a, j, k, ttl - 1 \rangle$ )
    {This step lets the agent work as a broker for the neighbors. Notice that this step is never reached in a centralized system since there is only one agent with no neighbors}
11:   end for
12: end if
13: for all  $u \in (\mathcal{A}_i - \{a\})$  do {Recommend the document to the monitored users that probably deem document  $j$  relevant and have not yet seen it}
14:   if  $\text{sim}(\mathbf{M}_u^i \times \mathbf{v}_j, \mathbf{m}_u^i) > t_u^i$  and  $j \notin \mathcal{D}_u^i$  then
15:     Add  $j$  to the unread document queue of user  $u$ .
     {The users will read the document in the next iteration of the main algorithm}
16:   end if
17: end for
18: if  $\text{sim}(\mathbf{M}_n^i \times \mathbf{v}_j, \mathbf{m}_n^i) > t_n^i$  then {The document is deemed relevant for the receiving user  $n$ }
19:   call agent  $s$  : receiveFeedback( $\langle i, a, j, a, 0 \rangle$ )
   {We lessen the amount of communication by only sending positive feedback}
20: end if
```

---

---

**Algorithm 4 – receiveFeedback**( $\langle s, a, j, a, 0 \rangle$ )

---

**Require:**  $i$  is the called agent.

**Require:**  $s$  is the sending agent.

**Require:**  $a$  is the user for which it is given feedback

**Require:**  $j$  is the recommended document

Update the user model of user  $a$  {see section 4.2, section 4.3 and section 4.4}

---

**Definition:** We define a recommendation as a tuple  $\langle s, a, j, n, ttl \rangle$  where  $s$  is the sending agent,  $a$  is the recommending user,  $j$  is the recommended document,  $n$  is the receiving user and  $ttl$  is the time-to-live (the number of hops left) for the recommendation. The initial  $ttl$  is 2. A  $ttl = 2$  means that the recommendation should be forwarded two times — the first time from the recommending user’s agent to its neighbor agents and then the second time from the neighbors to the neighbors’ neighbors (the distant neighbors of user T in Figure 1).

A monitored user might call its recommender agent  $i$  with the functions defined in Algorithm 1 and Algorithm 2. The functions are used to tell the agent that a document has been shown respectively that a document is relevant. These calls are simulated in the main algorithm presented in Section 5.2.

Recommender agents can call other recommender agents with the functions defined in Algorithm 3 and Algorithm 4. The functions are used to send a recommendation respectively to give explicit feedback. Observe that the latter is not the same as the previously mentioned feedback from the users. Instead it is a feedback between agents based on their models of the neighbor users. The reason is to give fast feedback and thereby speed up the model building.

## 5 The Simulator Implementation

In order to study the proposed recommender system we have implemented a simulator with variable parameters: the number of users, the agents tendency to explore to find new neighbors, the number of kept good neighbors and the degree of decentralization. In this paper only two degrees of decentralization are used: completely with one agent per user or with a single agent for all users.

The components of the simulator are some parts of the recommender system and the users. The simulated parts of the recommender system are documents including their presentation and information sources. The recommender agents are implemented as would be done in a real system. In a real setting, the information sources and the presentation of documents would be handled by the information monitoring software.

**Simulated users.** A user is modeled as having some interests, subscribing to some information sources, having some buddies (some other users) and having a monitoring recommender agent. As previously mentioned the system is supposed to be located in an organization. This is simulated by initializing the users with buddies taken from an artificially created hierarchical structure.

**Information Sources.** The information sources are sets of documents and interests are Boolean functions indicating whether a user likes a document or not. A user can receive documents both from its information sources and from its recommender agent. All received documents are put in a queue for unread documents until the user reads them. The buddies are initially used to bootstrap the recommender agent with some neighbors.

The rest of this section is structured as follows. First the implementation of the simulated parts such as documents, information sources and users is described and last the main algorithm of the simulation is described.

## 5.1 Documents, Information Sources and Users

For the simulation to be fairly realistic the documents, interests and information sources are based on a collection of articles from a real newspaper. The CLEF collection from TREC-9 contains all articles of the year 1994 edition of the Los Angeles Times and a set of 40 queries with some of the articles classified as relevant or irrelevant for each query.<sup>6</sup> The classification is sparse and does not cover all articles for each query. However, it is assumed that the articles classified as relevant are the only relevant articles in the collection.

### 5.1.1 Documents

As documents we use the news articles which are represented by word frequency vectors where the words are the elements and the number of occurrences of each word is the weight. Thus it is a simple document representation without any preprocessing.

### 5.1.2 Information Sources

As information sources we use the set of articles in the CLEF collection. However, the simulations require more than one source to be realistic. To solve this problem an information source corresponds to randomly drawn subset of the CLEF collection. Thus the information sources will partly overlap which will be a problem especially when the subsets are large enough. Each overlapping article is viewed as a unique instance for the same topic and not as a duplicate. Despite of the overlap, this setup is at least partially realistic since most newspapers report the same stories at the same time and the users might want to cover all articles from all sources for a certain story. In particular this is the case if we look at the problem as an information-monitoring problem when people often want to know all news items related to their organization and their work.

### 5.1.3 Users

For the users we have to define their interests and what buddies they have.

**Interests.** As interests we use Boolean functions corresponding to the query classification from the CLEF collection. This means that each interest corresponds to a set of articles relevant to a query. In the current setting we let each user have six different interests.

**Explicit feedback.** The sparse number of classified news articles makes it reasonable to think that the queries are explicitly captured interests where the users actively indicate whether an article is relevant and not implicitly captured from the user behavior. This is reasonable since real users are normally reluctant to give explicit feedback while it is fairly easy to get implicit feedback because explicit feedback requires extra work from the users.

**Buddies.** We assume that the system is to be introduced in an organization from scratch. A natural way to do this is to initially connect the users to their closest co-workers for example their superiors and subordinates. Thus we simulate the introduction in an organization by creating a hierarchical tree structure that connect the users. Conceptually, parents in the tree correspond to superiors and children correspond to subordinates. Thus as buddies of the users we use each user's parent and children. The hierarchical tree is created as follows: (1) The first user (the root) gets five other users as children, (2) Each child, not having own

---

<sup>6</sup><http://trec.nist.gov/data.html>

children, gets five users (not yet in the tree) as children<sup>7</sup>, (3) The previous step is repeated until there are no more users left outside the tree.

This results in a fully connected tree with at least six buddies for each non-leaf user except for the first user who will have only 5 buddies and the last user that might be missing some children. The leaf users will only have one buddy each. To get a structure more similar to what one would expect in a network of real buddies, one could also start out with a small world network as described in [17].

Remember that the users' buddies will be the agents' initial neighbors. Additional neighbors will be found by the exchange of recommendations. Thus it does not matter much whether all agents get six initial neighbors.

---

**Algorithm 5** The Main Algorithm of the Simulation

---

```
1: Create the Information Sources by randomly draw 60 days of publications in order out of
   365 from the CLEF collection for each source.
2: Create the users. Each user is initialized with:
   • Six different interests, randomly drawn.
   • One randomly drawn information source.
   • One recommender agent.
   • Maximally six buddies chosen as described above.
3: Let the recommender agents get the buddies of their users as initial neighbors.
4: for all simulated days do
5:   for all information sources do
6:     Add today's documents from the source to every subscribing user's unread document
       queue (if not already in the queue).
7:   end for
8:   for all user a do {In random order}
9:     Let a read all documents in the unread document queue
10:    for all document j read by a do
11:      if a likes j then
12:        call the agent of a : relevantDocument(a, j)
13:      end if
14:      call the agent of a : rememberShownDocument(a, j)
15:    end for
16:  end for
17: end for
```

---

## 5.2 The Simulator Algorithm

The main algorithm for the simulator in which the users are executed and the recommender agents are called is shown in Algorithm 5. Every call results in executing an agent that

---

<sup>7</sup>Or less than five if there are no more users left outside the tree.

might call other agents and so on. First the simulated information sources and users are created as described in Section 5.1. Second each day of publications from each source are presented to and read by each user. Last, the agents are called and executed as described in Section 4. The agents update user models and forward recommendations to other agents that in turn might present the recommendations to their users.

The execution is done in sequence but by executing users in random order we simulate parallelism. The system is run with 60 days of publications from each source.

## 6 Experimental Setting

### 6.1 Method

We compared the performance of a decentralized recommender system with the performance of a similar but centralized system. In the former, each user has a unique recommender agent, though with a limited number of neighbors, while in the latter, a shared recommender agent serves all users. All setups were run 30 times initialized with unique random seeds.

### 6.2 Metrics

The performance was measured with three different measurements, the average precision, the average recall and the average utility. Each measure is computed for every simulation day based on all the documents presented to the users so far with respect to each user’s interests. The system simulations were run over 60 simulated days.

All significance tests (two sided with  $p = 0.01$ ) were computed, if not otherwise stated, for the average sum of the point values of the curve (conceptually, that is the area below the curve). The null hypothesis was that there is no difference between the compared mean values  $\bar{v}_1$  and  $\bar{v}_2$ . The used test statistic was  $\frac{\bar{v}_1 - \bar{v}_2}{\sqrt{(s_1)^2/n_1 + (s_2)^2/n_2}}$  where  $s_i$  is the sample standard deviation of  $v_i$  and  $n_i$  is the number of measured values of  $v_i$ . The test statistic was assumed to be Student-t distributed with degrees of freedom estimated with the Welch-Satterthwaite approximation [18].

**Precision.** The precision for each user is the number of recommended relevant documents divided by the number of all recommended documents.

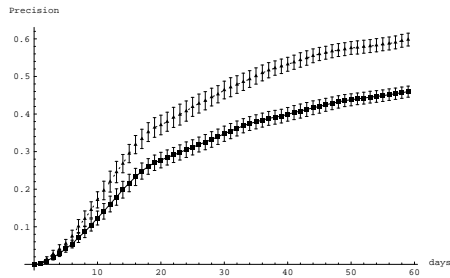
**Recall.** The recall for each user is the number of recommended relevant documents divided by the total number of relevant documents in the system originating from the other information sources (not from its own).

**Utility.** The utility of each user is similar to the measure used in the filtering track of TREC-9 (Robertson et al. 2000). We use  $utility = 2 \times n_+ - n_-$  where  $n_+$  is the number of recommended relevant documents and  $n_-$  is the number of recommended irrelevant documents.

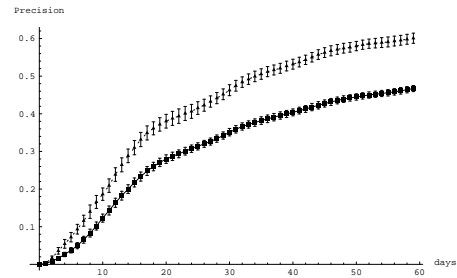
### 6.3 Evaluated Parameters

We have evaluated the system by varying three different parameters to test the scaling, the clustering effect and the effect of exploration.

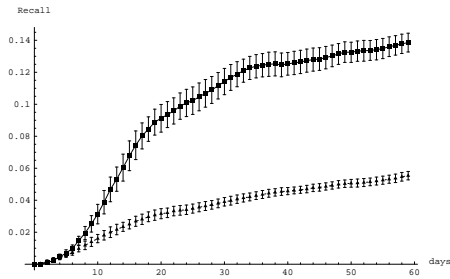
**Scaling.** To see whether the algorithm scales, we tested the systems with 100 users and with 400 users. The simulation results are shown in Figure 2a-c and Figure 2d-f respectively.



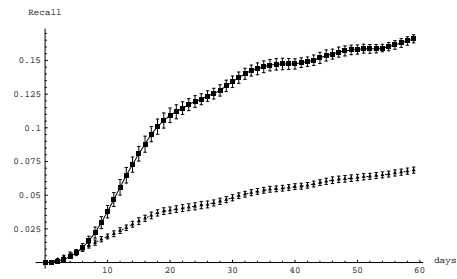
a) Average precision for 100 users



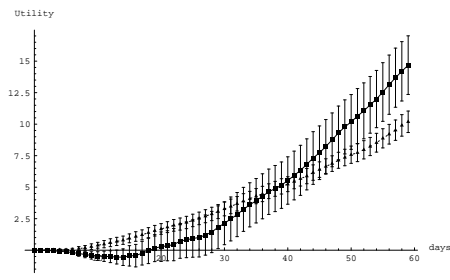
d) Average precision for 400 users



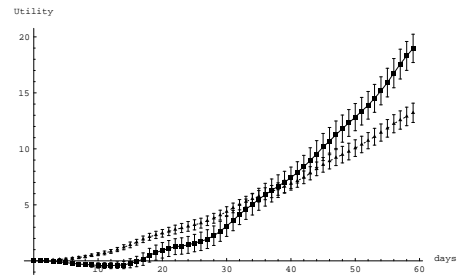
b) Average recall for 100 users



e) Average recall for 400 users



c) Average utility for 100 users



f) Average utility for 400 users

Figure 2: The performance comparison between a decentralized system (boxes) and a centralized system (triangles) with 100 simulated users (a-c) respectively 400 users (d-f). The maximum number of potentially good neighbors was four and the probability of exploration was 0.3. The horizontal axis shows the number of simulated days passed since the start of the simulation. Both systems were run 30 times and each time initialized with a new random seed.

Compared setups	Avg. num. of com. interests	Average precision	Average recall	Average utility
4 vs. 6 of 6	NO	NO	NO	NO
4 vs. 2 of 6	NO	NO	YES for 4	NO
2 vs. 6 of 6	YES for 2	NO	YES for 6	NO
6 of 6 vs. CS	—	YES for CS	YES for 6	NO
4 of 6 vs. CS	—	YES for CS	YES for 4	NO
2 of 6 vs. CS	—	YES for CS	YES for 2	YES for CS
2 of 3 vs. CS	—	YES for CS	NO	YES for CS

Table 1: The table shows the performance comparison between different number of potentially good neighbors and the centralized system (denoted CS). The number of users was 100 and the probability of exploration was 0.3. The maximum number of neighbors was six for the first two sections and 3 for the last section. “YES for X” means “Significantly better for X”, “NO” means “No significant difference”.

The vertical line at each point plots the confidence interval that contains the real value with a probability of 0.99 (using Student-t with 29 degrees of freedom).

**Clustering.** We also tested whether varying the number of potentially good neighbors among the neighbors (and thereby the number of kept good neighbors because the maximum number of neighbors is six) made any difference to the performance. The results are shown in Table 1.

**Exploration.** The next test we performed was to vary the probability of exploration while keeping the number of potentially good neighbors constant. The results are shown in Table 2.

## 7 Results and Discussion

### 7.1 Scaling

In the case of 100 users (Figure 2a-c), the average precision is significantly better for the centralized system while the average recall is significantly better for the decentralized system and the average utility differs insignificantly. However, on the last simulation day the average utility differs significantly in favor of the decentralized system. This means that more articles are presented to the users in the decentralized system than in the centralized one.

The same is true when scaling the system to 400 users (Figure 2d-f) though the standard deviation is much smaller.

### 7.2 Clustering

By splitting the neighbors into two sets we hoped to form clusters of users with overlapping interests that would improve the system performance. This was not indicated by the results shown in Table 1. The average number of common interests does not differ significantly when comparing the use of four (4 of 6) with two (2 of 6) respectively six (6 of 6) potentially

Compared setups	Avg. num. of com. interests	Average precision	Average recall	Average utility
<b>0.1 vs. 0.0</b>	<b>YES for 0.0</b>	<b>YES for 0.1</b>	<b>YES for 0.1</b>	<b>YES for 0.1</b>
<b>0.2 vs. 0.1</b>	<b>NO</b>	<b>NO</b>	<b>YES for 0.2</b>	<b>NO</b>
<b>0.3 vs. 0.1</b>	<b>NO</b>	<b>NO</b>	<b>YES for 0.3</b>	<b>NO</b>
<b>0.3 vs. 0.2</b>				
<b>0.3 vs. 0.5</b>				
<b>0.3 vs. 0.75</b>	<b>NO</b>	<b>NO</b>	<b>NO</b>	<b>NO</b>
<b>0.3 vs. 1.0</b>				
<b>0.75 vs. 1.0</b>				
<b>4 of 4 &amp; 0.3 vs. 6 of 6 &amp; 1.0</b>	<b>YES for 4 &amp; 0.3</b>	<b>YES for 4 &amp; 0.3</b>	<b>YES for 4 &amp; 0.3</b>	<b>YES for 4 &amp; 0.3</b>
<b>0.0 vs. CS</b>	—	<b>YES for CS</b>	<b>YES for 0.0</b>	<b>YES for CS</b>
<b>0.1 vs. CS</b>				
<b>0.2 vs. CS</b>	—	<b>YES for CS</b>	<b>YES for p2p</b>	<b>NO</b>
<b>0.3 vs. CS</b>				
<b>0.5 vs. CS</b>				
<b>0.75 vs. CS</b>				
<b>1.0 vs. CS</b>				
<b>6 of 6 &amp; 1.0 vs. CS</b>	—	<b>YES for CS</b>	<b>YES for p2p</b>	<b>YES for CS</b>

Table 2: The performance comparison between different probabilities of exploration, and in the last two rows, compared to the centralized system as well (denoted CS). The number of users was 100. The maximum number of potentially good neighbors was four in all but one test. The maximum number of neighbors was six. “YES for X” means “Significantly better for X”, “NO” means “No significant difference”.

good neighbors. This means that clustering is not very much affected by the number of good neighbors. However, when comparing the use of two respectively six potentially good neighbors there is a significant difference in favor for the former indicating that the ability to keep good neighbors at least has some effect. Though, in the case of two potentially good neighbors the decentralized system performs worse than the centralized for all measures but recall.

When comparing the decentralized system with varying number of potentially good neighbors and the centralized version it seems better to keep many potentially good neighbors. Too many good neighbors give worse performance.

The last row in Table 1 shows the result of using maximally three neighbors instead of six with two potentially good neighbors and one good neighbor. The average recall for the decentralized system is in this case similar to the average recall for the centralized system. Thus the forwarding of recommendations between peers seems to be important for the recall but gives a low precision compared to the centralized system.

### 7.3 Exploration

As seen in Table 2, the worst performance (for precision, recall and utility) is achieved for a probability of 0.0. This means that having no exploration is very bad for the performance. A probability of 0.3 is also better than having only 0.1 but there seems to be no improvement or lack of improvement when using other values. Notable is that the insignificant difference between a probability of 0.3 and 1.0 and a probability of 0.75 and 1.0. Thus it seems more important to be changing neighbors than keeping neighbors that have recommended relevant documents. Though in the last section probabilities above 0.5 lowers the performance for utility.

When having a probability of 1.0 while not keeping any good neighbors (6 of 6) differs significantly from having a probability of 0.3 while only keeping two good neighbors (4 of 6). Thus the combination of exploring a lot and not keeping any good neighbors seems to lower the performance but still the recall is significantly better for the decentralized system (p2p).

When looking at the average number of common interests the best is of course achieved for a probability of 0.0 but when instead the performance is very bad.

Nevertheless, the last section in Table 2 implies that without the probability of exploration we would end up with a decentralized approach that is significantly worse than the centralized version. Yet, with a probability of only 0.1 we will still get a system with a better performance for recall and comparable utility (the utility is significantly better on the last day) though the average precision differs significantly. The same is true for an exploration probability of 1.0.

## 8 Summary and Conclusions

In this paper we have shown that a decentralized recommender system with 100 users seems to perform equally well compared to a centralized approach with respect to recall and utility but worse with respect to precision. This is a good result given that the decentralized system has agents with a limited number of neighbors and many other tunable parameters.

The system was tested with a simulator. The simulation model started out with a hierarchical connected network of users with recommender agents. The users read news articles and give explicit indication whether an article is relevant thus we use only positive feedback. The recommender agent forwards the relevant articles to its neighbors. The neighbors in turn, work as forward the recommendations to their neighbors and maybe to their users. By selecting which users to keep as neighbors both based on relevant recommendations and with a certain probability, the agents hope to form cluster and thereby shorten the path between good neighbors. In this way the articles should be spread among the users, not requiring any central processing.

The ability to keep good neighbors seems to neither improve the ability to form clusters nor the performance. This is probably because the neighbors change quickly. The agents do not have time to build detailed models of the neighbors and thus it is hard to choose good neighbors. In contrast, the ability to explore the neighborhood of neighbors giving irrelevant recommendations is essential for the performance. Though the forwarding of recommendations between peers seems to improve the recall but lowers the precision while the utility is similar at least after 60 simulated days. The tests with 400 users indicate that the decentralized approach would perform well even for larger populations.

In the experiments the used recommender algorithm (the vector space model) was the same for both the decentralized and the centralized system. To get similar performance for

both systems it seems not enough to vary the probability of exploration or the parameters for the clustering of peers. Though by lowering the maximum number of neighbor to three the recall became similar but the precision and utility was lowered as well. Thus it seems reasonable to assume that the only way to achieve similar performance is by using different recommender algorithms or using different parameter values of the current one.

## 9 Acknowledgments

This work was funded by the Intelligent Agents program funded by VINNOVA 1996 - 2002 and the Social Computing program supported by SITI 1999 - 2003. The simulator was implemented in the RePast simulation toolkit for multi-agent systems<sup>8</sup>.

I want to thank my supervisor Professor Arne Andersson at CSD, Uppsala University for his good advice on how to run experiments and how to write this paper, and Professor Kristina Höök at the IT-University in Kista (formerly head of the HUMLE laboratory at SICS) for her help and encouragement to write this paper. I would also want to thank all co-workers at SICS for their valuable comments and discussions.

## References

- [1] Foner, L.N. 1997. Yenta: A Multi-Agent, Referral-Based Matchmaking System. In Proceedings of The First International Conference on Autonomous Agents, 301-307. ACM Press.
- [2] Tveit, A. 2001. Peer-to-peer based recommendations for mobile commerce. Proceedings of the first international workshop on Mobile commerce 2001. pp 26 - 29. ACM Press New York, NY, USA.
- [3] Breese J.S., Heckerman David and Kadie Carl. 1998. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. MSR-TR-98-12. May (revised October 1998).
- [4] Sarwar, B.M., Joseph A.Konstan, Al Borchers, Jon Herlocker, Brad Miller, and John Riedl. 1998. Using filtering agents to improve prediction quality in the GroupLens research collaborative filtering system. In Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW).
- [5] Baeza, R. and Neto, B. 1999. Modern Information Retrieval. ACM Press New York, Addison Wesley.
- [6] Lashkari, Y., Metral, M. and Maes, P. 1994. Collaborative Interface Agents. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94), Vol 1. AAAI Press/The MIT Press. pp 444-450
- [7] Good, N., Schafer, J., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J. and Riedl, J. 1999. Combining collaborative filtering with personal agents for better recommendations. In Proceedings of the Sixteenth National Conference on Artificial Intelligence.
- [8] Schafer, J.B., Konstan, J.A. and Riedl, J. 2001. E-commerce recommendation applications. Data Mining and Knowledge Discovery , vol. 5 nos. 1/2, pp 115-152.

---

<sup>8</sup><http://repast.sourceforge.net/>

- [9] Robertson, S. and Hull, D.A. 2000. The TREC-9 Filtering Track Final Report. In Proceedings of the Ninth Text REtrieval Conference (TREC 9). NIST Special Publication 500-249. pp 25-40.
- [10] Ungar, L.H. and Foster, D.P. 1998. Clustering Methods for Collaborative Filtering. AAAI Workshop on Recommendation Systems.
- [11] Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P. and Riedl, J. 1994. GroupLens: An open architecture for collaborative filtering of netnews. Proceedings of the 1994 Computer Supported Collaborative Work Conference.
- [12] Goldberg, K., Roeder, T., Gupta, D. and Perkins, C. 2001. Eigentaste: A Constant Time Collaborative Filtering Algorithm, *Information Retrieval Journal*,4(2), pp. 133-151. July.
- [13] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D. and Sartin, M. 1999. Combining Content-Based and Collaborative Filters in an Online Newspaper. ACM SIGIR Workshop on Recommender Systems, Berkeley, CA, August 19.
- [14] Svensson, M., Laaksojahti, J., Höök, K. and Waern, A. (2000) A Recipe Based Online Food Store, In Proceedings of the 2000 International Conference on Intelligent User Interfaces (IUI'2000), New Orleans, Louisiana, USA, 260 - 263.
- [15] Nwana, H.S. 1996. Software Agents: An overview. In: *Knowledge Engineering Review*, Vol. 11, No 3. Cambridge University Press. Pp 1-40.
- [16] Luck, M., McBurney, P., Preist C. and Guilfoyle, C. 2002. The AgentLink Agent Technology Roadmap Draft, AgentLink.
- [17] Watts, D. and Strogatz, S. Collective dynamics of small-world networks. *Nature*, 393:440– 442.
- [18] NIST/SEMATECH e-Handbook of Statistical Methods, <http://www.itl.nist.gov/div898/handbook/>, 2002 December 12.
- [19] Belkin, N.J., Perez Carballo, J., Cool, C., Kelly, D., Lin, S., Park, S.Y., Rieh, S.Y., Savage-Knepshield, S.Y. and Sikora, C. 1998. Rutgers' TREC-7 Interactive Track Experience. In: D. Harman, ed. Proceedings of the Sixth Text Retrieval Conference (TREC-7), pp 275. Washington, D.C.: GPO.

## A.2 Paper II



# Genesis: A method for bootstrapping recommender systems using prior knowledge

Tomas Olsson  
tol@sics.se

Åsa Rudström  
asa@sics.se

April 28, 2003

Swedish Institute of Computer Science, SICS, Box 1263  
SE-164 29 Kista, Sweden

In the beginning God created the heavens and the earth. Now the earth was formless and empty, darkness was over the surface of the deep, and the Spirit of God was hovering over the waters. And God said, “Let there be light,” and there was light.

Genesis 1:1-3 (NIV)

## Abstract

Bootstrapping is an intrinsic problem for recommender systems and any other system that learns from experience. When there is no data, there is no experience and hence nothing to recommend. We propose a general bootstrapping method that uses prior knowledge. The knowledge is represented as a set of probabilistic models of user behavior in a domain. Artificial user profiles are created by instantiating specific user models drawn from the general model set. The system is then bootstrapped with a number of such profiles. The method was tested in the domain of movie recommendation. A  $k$ -nearest neighbor algorithm was bootstrapped with different sets of artificial user profiles. The system was then trained and tested on real user profiles from the EachMovie data set. Performance was measured for accuracy using Mean Absolute Error (MAE) and Rank Score. Comparing with no bootstrapping at all, results showed a significant improvement of the performance in terms of MAE for the first 20 real training profiles.

## 1 Introduction

When starting up a recommender system every developer will be faced with the problem that without any stored users it is hard to make good recommendations. Furthermore, if there are many users but few opinions there might be a problem to find matches between different users. The ordinary approach to tackle this problem is to combine collaborative filtering – which is entirely based on user opinions – with content-based methods, expecting the two approaches level out each other’s weaknesses.

In this paper, we propose a method that uses prior knowledge to bootstrap a collaborative recommender algorithm. Prior knowledge is acquired by asking experts, by analyzing the

recommended items, and by using preconceived notions about the world together with good guesses. The prior knowledge is represented as a set of probabilistic models, making it possible to bootstrap any used algorithm with instances of sampled user profiles. By using prior knowledge, the recommender is expected to give better recommendations to early users compared to a system lacking the prior knowledge. The method was tested in the domain of movie recommendation using the EachMovie data set.

The proposed method has three phases: domain analysis, model building, and user sampling. In the domain analysis, we first analyze the recommended items. Second, we investigate the user profiles. Third, we consider what kind of prior knowledge there is. The results from the domain analysis are used to build a set of probabilistic models reflecting the working of system users. A recommender system using any collaborative recommender algorithm might finally be bootstrapped with artificial user profiles sampled from the model.

This paper is organized as follows. We start with an overview of previous work in using prior knowledge for bootstrapping systems, and then describe the k-nearest neighbor algorithm. Next, the Genesis method is discussed, followed by its implementation in the domain of movie recommendations. The system performance is tested with and without prior knowledge, and conclusions are drawn and further work is outlined.

## **2 Background and previous work**

The purpose of a recommender system is to make a personal prediction of a user's opinion on a specific item. There are essentially two sources of knowledge to base this prediction on: the opinion of other users similar to the current user, or some set of rules describing how user preferences are related to item properties.

By basing the recommender on user opinions there is no need to analyze and understand exactly what makes a certain user prefer a certain item. Users are simply described in terms of the items they have expressed opinions on, and items may simply be named or enumerated. User opinions may be captured either by explicitly asking the user to rate items, or implicitly by observing user behavior. Buying a book at Amazon.com is for example interpreted as giving a positive rating on that book. Combinations of explicit and implicit rating are also used. Recommendations are made by matching the current user with similar stored users, suggesting items that these users have been positive about.

One method or set of methods that makes predictions for items based on given opinions is collaborative filtering [21, 11]. Collaborative filtering suffers from many bootstrapping problems, foremost from the early-rater problem: when there are no opinions for an item it cannot be recommended; and the sparsity problem: when there are few opinions from each user, users cannot be matched.

Another approach to the recommendation of items is content-based filtering. Here, the content of the items is used to make predictions. Content-based systems suffer from the cold-start problem where learning to filter in a good way takes time and the serendipity problem where only items with already encountered content can be recommended.

### **2.1 Combining content-based and collaborative filtering**

The prominent way of tackling bootstrapping problems for recommender systems has been to combine collaborative filtering with content-based algorithms, in order to level out their corresponding weaknesses. For example, collaborative filtering cannot make predictions

for items with no opinions, while content-based filtering still can. Various systems have taken this approach.

One approach has been to bootstrap a pure collaborative filtering system with agents, using content-based algorithms, acting together with the ordinary users. The GroupLens project (a recommender system for Usenet news) used agents with rather simple rating schemes such as ratings based on news length and number of misspelled words [20]. The mentioned examples implicitly use the prior knowledge that people prefer well-written news. In MovieLens (a recommender system for movies) the work was extended with agents using more complex rating strategies, for example a doppelganger that uses learning methods to mimic a real user but can of course rate many more items [9]. The system was also bootstrapped with millions of real user data from an earlier recommender system.

Another approach has been used for Fab (recommending web pages) where the collaborative filtering is entirely based on content [1]. The user models are content based and matched to infer similar interests.

A pure machine learning strategy is applied in [2] where an inductive learner system, Ripper, learns how to recommend movies based on a combination of user opinions and content attributes.

In [17] two approaches that use text content to decrease the sparseness of user opinions are described. The first approach computes predictions based on the number of co-occurrences between users and items (for example, the number of times a user accessed an item). Missing co-occurrences are filled in with the average cosine similarity between an item and a user's already known co-occurred items. This approach differs from the previous agent approach in that opinions are artificially created for real users while in the agent-based approach the agents are seen as extra users. The second approach is to count co-occurrences between users and words instead of users and items. This reduces the sparsity drastically when words are contained in many text items. The result indicates that the second and more purely content-based approach performs better than the first approach.

## 2.2 Knowledge-based recommender systems

We hardly ever have complete or correct prior knowledge of every single user's preferences. Nor do we completely understand why people like what they like, and the taste usually varies from user to user, and sometimes from time to time. Nevertheless, we might have some understanding for certain domains. In the movie domain, we might expect that users like movies of a certain director or with a famous actor if they like other movies with that same person. For both movies and books, we might expect the genre to be a good predictor of user opinions. Even in domains where such naive, informal prior knowledge might be sparse, it should be possible to make use of whatever prior knowledge we have, as suggested in [23]. The prior knowledge might be acquired by analyzing the recommended items (e.g. by means of data mining), by asking experts, or by simply using preconceived notions and good guesses – they might at least be partially correct. If we know that the taste of people usually splits over a certain item, this information should be possible to use in the recommender system. In addition, if some potential users have already been analyzed, it should be possible to transfer that knowledge into the design of the recommender system. However, not all users behave as expected and hence the system must be able to adapt to the individual by learning.

The idea of combining background knowledge with collaborative filtering to improve recommendation is not new. In Entree [4, 5], a knowledge-based recommender system for restaurants is modified to use collaborative filtering to refine its final suggestions. Given

some user preferences, Entree uses its knowledge base to retrieve a small, unordered set of matching restaurants, which is then ordered by collaborative filtering. The best restaurant, chosen randomly if no order is available, is presented to the user. If not satisfied, the user can ask for a restaurant that is cheaper, livelier, more exotic, etc. to further refine the preferences. The process is then repeated with the refined user preferences.

When the prior knowledge is correct, Entree performs well despite the performance of the collaborative filtering but Entree lacks the ability to handle incomplete or incorrect prior knowledge. In comparison, our knowledge base is refineable and not necessary complete or correct and we want to be able to use any available prior knowledge. Thus, our approach is more related to machine learning than to knowledge-engineered systems.

### 2.3 Prior knowledge in Machine Learning

In machine learning research, the benefit of initializing a learning system with a domain theory of explicit prior knowledge has been shown in the effort of combing analytical and inductive learning ([14, 15]). One such approach is the knowledge-based artificial neural network (KBANN) described in [23]. The KBANN starts with a domain theory of propositional horn clauses that is mapped into a neural network. The neural network is then trained on real data. The resulting network might finally be transformed back into horn clauses. Tests of the KBANN suggest that one should use all knowledge one has to improve learning even when the knowledge is not completely correct. However, the tests also suggest that it is bad to add completely incorrect knowledge.

A couple of other algorithms for combing analytical and inductive learning (as well as the KBANN) are described in [14]. In all of these algorithms, the inductive learning might refine and correct a given domain theory while the domain theory still influences the induction with a search space bias.

Though background knowledge has been used to improve machine learning algorithms it has not explicitly been used for recommender systems. Our primary contribution is the proposed method of bootstrapping any recommender system with explicit background knowledge. The secondary contribution, related to machine learning, is to bootstrap an instance based algorithm, k-nearest neighbor. Previous approaches have been applied to model-based algorithms such as neural networks.

## 3 Bootstrapping for k-nearest neighbor

The k-nearest neighbor algorithm is one of the most widely used algorithms for collaborative filtering, as discussed in [13] and [3]. The k-nearest neighbor is a so-called instance based method (sometime called memory based) because training instances are stored (memorized) during training and retrieved when making predictions. In contrast to instance-based approaches, model-based methods generalize into a model from the training instances during training. The model is then used to make predictions. Examples of model-based methods are Bayesian belief nets and artificial neural networks [14]. Instance based methods learn fast but make slow predictions, while model based methods make fast predictions but learn slowly. The k-nearest neighbor algorithm works such that for any active user the k most similar stored instances are retrieved and then the predictions are made based on them. The contribution of each instance might also be weighted according to some similarity, distance, or correlation measure.

The k-nearest neighbor algorithm for collaborative filtering used here is taken from [3].

In the following there are  $n$  stored instances of user profiles. A user profile consists of one or more items with corresponding opinions. An opinion is a value taken from a numerical scale defined by the system designers. The predicted opinion of an active user  $a$  for item  $j$  is then

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

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

$v_{i,j}$  = the opinion of user  $i$  for item  $j$

$I_i$  = the set of items for which user  $i$  has given opinions

$\omega(a, i)$  is the weighted contribution of user  $i$  to the prediction for user  $a$ .

The weight function is the Pearson correlation coefficient.

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

$\kappa$  = a normalizing factor such that the absolute weights sums to unity.

It is not obvious how to incorporate prior knowledge in the classical k-nearest neighbor algorithm. One approach is to use a more complicated similarity measure or weight function that encodes the prior knowledge. For instance by adding a comparison measure  $\beta$  in Equation 2. This approach is related to the heuristic similarity measures used in case-based reasoning [5]. If instances have content attributes such as movie director, actors, etc., it would be easy to encode prior knowledge such as “equal values for the director attribute are more important than equal date of release” into the  $\beta$  function.

$$\omega(a, i) = \frac{\sum_j \sum_l (v_{a,j} - \bar{v}_a)(v_{i,l} - \bar{v}_i) \times \beta(j, l)}{\sqrt{\sum_j \sum_l (v_{a,j} - \bar{v}_a)(v_{a,l} - \bar{v}_a) \beta(j, l) \sum_j \sum_l (v_{i,l} - \bar{v}_i)(v_{i,l} - \bar{v}_i) \beta(j, l)}}$$

If  $\beta(j, l) = \begin{cases} 1 & \text{if } j = l; \\ 0 & \text{if } j \neq l \end{cases}$  we get the original weight function.

In addition, we can easily incorporate content into the algorithm. If the items were text documents the item-to-item weight function  $\beta$  could be the cosine similarity between the word frequency vectors of the items in the vector space model of information retrieval [19]. Nevertheless, this encoding of the prior knowledge requires quite large changes to the algorithm, redefining the correlation coefficient. We aim for a more general approach, where the bootstrapping algorithm is separate from the filtering algorithm.

## 4 The Genesis method

A bootstrapping approach should of course vary with what sort of prior knowledge and user profiles there are, what is recommended, and what algorithm used. However, the most general input of knowledge there is, working for any algorithm, is the set of instances (in our case user profiles) used for training. If we could model users and produce a set of artificial profiles, the system could simply be trained on these profiles. No extra efforts or changes to the recommender algorithm would be needed.

The Genesis method aims at creating a set of general, probabilistic models of the future users of a recommender or collaborative filtering system that needs to be bootstrapped. This is done by careful analysis of the domain, followed by creative model building, and finally, by instantiating selected models to create artificial users.

## 4.1 Data analysis

**Item attributes.** The first step in the data analysis is to analyze the items to recommend. If the items lack attributes e.g. content, it might be hard to model prior knowledge. However, known relations between items could be used without having explicit access to the content. In the movie domain, movies and stereotype users could be linked in a graph based on expert knowledge on movie categories and the type of person who likes them. This is not the approach taken in this paper. We assume that items have content that can be extracted for prior knowledge.

**User profiles.** In classical collaborative filtering a user profile consists of a list of items paired with the user's opinions on these items. This enables users to be matched based on their opinions as opposed to content. Usually the profile is built over time, with users coming back to rate new items. An example of a recommending system where tracing is possible is the recipe recommender Kalas [22]. Kalas requires users to login. In contrast, the recommender of PC configurations presented in [7] is designed for ephemeral users that are not traced. In the latter, the user profiles consist of selected PC configurations with no connection to any individual user. However, the users could easily be traced using the information from the order form.

**Prior knowledge.** Third we consider other available sources of prior or background knowledge. Basically, we need knowledge about how users relate to items. What do users like? What schemes do they use when rating items? Are there different user types? The content of the set of recommended items will certainly contain usable prior knowledge. User attributes not directly used for recommendation could possibly be used. Data from different sources such as marketing surveys might be available, making it possible to extract prior knowledge e.g. with data mining methods.

## 4.2 Model building

An underlying assumption or inductive bias behind the learning ability of existing recommender systems is the existence of correlations between items or attributes. In other words, people who liked a certain item are expected to also like a second item or items with certain attributes. Many recommender systems for e-commerce are based on this assumption, for example Amazon's "Customers who bought this book also bought" other books [13]. This leads to a probability model with dependencies between some or many of the items or attributes.

A second underlying assumption behind many recommender systems is the existence of groups of closely correlated users, which implicitly assumes ability to track individual users. Especially instance based collaborative filtering and cluster-based algorithms make explicit use of this assumption [24, 3]. Consequently, it seems reasonable to bootstrap the system with user profiles sampled from a mixture of probabilistic models that capture both the variation and similarity among real users. The k-nearest neighbor algorithm used for collaborative filtering is insensitive to non-correlated user profiles [20], and thus it does not matter very much if we have many faulty models in the mixture. However there is always a risk of getting coincident correlations by chance (where there are no real correlation) and

therefore we cannot randomly chose a large mixture model. This is especially true when the instances (the user profiles) have very few items or attributes.

### 4.3 User sampling

The last step is to create a set of sampled, artificial profiles to be used for bootstrapping. As discussed above, the probabilistic model set created from prior knowledge will consist of a mixture of models, reflecting the assumption that real users belong to different user types or clusters. The creation of a sampled user profile is thus performed in two steps. First, a specific user model is drawn or selected from the model set. Second, the probabilities of the model are instantiated, resulting in a sampled user profile.

The user models may also be mixed. For example, in the model set for the movie domain described in the next section, models for user preferences are combined with models of rating habits.

## 5 Applying Genesis to the movie domain

One of the most explored domains for recommender systems is the movie domain, for which both commercial and research systems have been built. Two commercial systems are Amazon.com<sup>1</sup> and The Internet Movie Database<sup>2</sup>, and two well-known research systems are EachMovie<sup>3</sup> and MovieLens<sup>4</sup>.

We tested our approach on the EachMovie data set that is available for research purposes. The EachMovie recommendation service was run by the Compaq Systems Research Center for 18 months to experiment with a collaborative filtering algorithm. During this time, users entered around 2.8 million numeric ratings for 1628 different movies (films and videos).

In addition to the EachMovie data, there is also the Internet Movie Database (IMDb) containing attributes for movies. The 1628 movies in the EachMovie data set are all included in the total set of movies in the IMDb.

### 5.1 Data analysis

In the EachMovie data, individual users are tracked. A user profile consists of a number of movies rated on a scale of 0 to 5. The dataset contains well over 65000 such profiles.

The EachMovie data contains very little information about the 1628 movies rated by its users. However, content attributes for movies can be found in the IMDb. movies.

In an experimental situation such as this we have to be careful when defining our prior knowledge. There is a definite risk that the prior knowledge is biased by too closely examining the real user data. Such data is of course not available in a realistic setting.

As prior knowledge we assume that users often tend to like or dislike movies with the same actors, same directors, similar plot, etc. This is reflected in the model by dividing the movies into clusters with similar movies. To speed up the clustering we have chosen to only cluster with respect to the plot.

---

<sup>1</sup><http://www.amazon.com>

<sup>2</sup><http://www.imdb.com>

<sup>3</sup><http://www.research.digital.com/SRC/>

<sup>4</sup><http://www.movielens.umn.edu/>

In addition, it could be argued that male and female users differ on what movies they like. Fortunately, instead of relying only on our own prejudices we have access to some “expert knowledge” in form of the top-ranking movie lists of the Internet Movie Database. There are the top 50 lists of their male and female voters as well as the top 250 list for all their voters. We use the rank to increase the probability that highly ranked movies (and movies in the same clusters) get high scores. The number of movies of the male top 50 list and the female top 50 list in the EachMovie data are 34 and 26 respectively. Among the top 250 only 131 are contained in the EachMovie data.

The notation  $\Pr(\dots|\dots, K)$  will be used to clearly show that the probability is based on the prior knowledge  $K$  as specified below. As shorthand we will use  $\Pr(Cluster = k_i|\dots) \equiv \Pr(k_i|\dots)$ ,  $\Pr(Gender = g|\dots) \equiv \Pr(g|\dots)$ ,  $\Pr(Movie = m_i|\dots) \equiv \Pr(m_i|\dots)$  in the following when the equations get too messy.

## 5.2 Model Building part I: Movie Clustering

Using information from the IMDb, movies are divided into clusters with similar movies, and each cluster gets a distribution of rating scores. The variable Cluster represents the clusters.

As clustering tool we use AutoClass [6], which is a Bayesian variant of EM clustering [8]. Instead of clustering complete movies where each training instance consists of many attributes, one for each movie attribute-value, we cluster movie-attribute-value pairs. The pairs are modeled as two multinomial distributions. This is the same clustering principle as for the two-way aspect-model presented in [12] for IR and [17] for recommender systems. The advantage compared to clustering instances with many attributes is first and foremost that each movie can belong to many clusters at the same time while otherwise, if given enough data, each movie would tend to belong to just one cluster. The clustering results in partial probability belongings for each cluster and each movie such that  $0 \leq \Pr(Movie = m_i|Cluster = k, K_c) \leq 1$  and  $0 \leq \Pr(Cluster = k|Movie = m_i, K_c) \leq 1$ . In addition, we get the  $\Pr(Cluster = k|K_c)$  for all clusters. The  $K_c$  indicates that the probabilities are computed by the cluster algorithm in contrast to  $\Pr(Cluster = k|K)$  that indicates that the probability is given by all our prior knowledge.

A training instance consists of a movie identity and an attribute value. The attribute values are taken from the movie plot in the IMDb. The plot is a text attribute that is pre-processed before being used in the clustering. Stop-words<sup>5</sup> are removed and the remaining words are stemmed with the Porter stemmer [18]<sup>6</sup>. All attribute values occurring in less than 2 or more than 500 movies are removed.

## 5.3 Model Building Part II: Sampling Models for User Profiles

All users rate movies on a scale from 0 to 5. However, their ways and preferences in rating movies may differ which is reflected in randomly drawn preferences for the rating range.

On top of the rating preferences we have investigated two basic rating schemes to model the rating of a user: the random rating scheme and the constant rating scheme. These two basic rating schemes are then extended with the prior knowledge of the rank order or the clusters.

---

<sup>5</sup>We use the stop-words of SMART Information Retrieval System at <ftp://ftp.cs.cornell.edu/pub/smart/>

<sup>6</sup>An implementation can be found at <http://www.tartarus.org/~martin/PorterStemmer/>

### 5.3.1 Rating Range Preferences

Regardless of the rating scheme, a user is modeled to have a probability of liking (giving the highest score to) or disliking (giving the lowest score to) each movie. When a user  $a$  rates a movie  $j$ , she will rate it with the score closest to the expected rating score:

$$v_{a,j} = \operatorname{argmin}_{s \in S} (|s - E[s_{a,j}|K]|), \text{ the rating score given by user } a \text{ to movie } j \quad (3)$$

$S = \{0, 1, 2, 3, 4, 5\}$ , the set of all scores

$$E[s_{a,j}|K] = s_{a,high} \times \Pr(a \text{ likes } j|K) + s_{a,low} \times \Pr(\neg(a \text{ likes } j)|K)$$

$E[s_{a,j}|K]$  is the expected score of movie  $j$  for user  $a$

The available scores are from 0 to 5 inclusive, yet users might differ when applying these scores. One user might give the best movies the score of 4 while another might give them the score of 3 and yet another user the score of 5. This is modeled by constraining the output of a user by randomly drawing a mean score  $\bar{s}$  and a mean deviation  $\bar{s}_{dev}$ . To keep the model simple  $\bar{s}$  and  $\bar{s}_{dev}$  are drawn from uniform distributions,  $\bar{s} \sim U(0.5, 4.5)$  and  $\bar{s}_{dev} \sim U(0.5, \min(\bar{s}, 5 - \bar{s}))$ . The highest and lowest score for user  $a$  are then:

$$s_{a,high} = \bar{s} + \bar{s}_{dev}$$

$$s_{a,low} = \bar{s} - \bar{s}_{dev}$$

### 5.3.2 The Random Rating Scheme

In the random rating scheme a user model is created by giving each movie a uniformly drawn ‘‘probability’’ of getting the highest rating score. Thus there is no dependency between any two probabilities. Because score is linearly related (Equation 3) this is equal to drawing a rating from the uniform distribution  $U(s_{a,low}, s_{a,high})$ .

### 5.3.3 The Constant Rating Scheme

The constant rating scheme is a simple scheme where each probability has a constant value:  $\Pr(a \text{ likes } j|K) = 0.5$ . This means that a user profile will have the same rating score for each movie; thus this rating will be equal to the profile’s mean score. However, because of users’ different rating preferences the mean score will vary from profile to profile.

### 5.3.4 Extension 1: Rank Order-based Rating

By extending the model using the prior knowledge of the rank order from IMDb top lists, a dependency on gender is added. Thus each user is modeled to have a gender  $g$ .

In the random rating and the constant rating schemes every user  $a$  gets a probability  $\Pr(a \text{ likes } j|K)$  for each movie  $j$ . This probability is refined to depend on the gender  $\Pr(a \text{ likes } j|Gender = g, K)$  by first calculating and refining the  $odds = \frac{\Pr(a \text{ likes } j|K)}{\Pr(\neg a \text{ likes } j|K)} = \frac{\Pr(a \text{ likes } j|K)}{1 - \Pr(a \text{ likes } j|K)}$  and then transforming the odds back to a probability:

$$\Pr(a \text{ likes } j|Gender = g, K) = \frac{odds_{new}}{1 + odds_{new}} \quad (4)$$

$$odds_{new} = \frac{\Pr(a \text{ likes } j|K)}{1 - \Pr(a \text{ likes } j|K)} \times \frac{1 - \omega_{g,j}}{0.5} \quad (5)$$

where  $\omega_{g,j} = \frac{rank_{g,j} + rank_{top250,j} + 1}{2 \times MaxRank + 2}$ , and  $rank_{g,j}$  is the rank corresponding to the rank when movies not in the EachMovie data set are removed from the rank lists. There is a “0.5” in the denominator of the second fraction because a user is assumed to be indifferent to a movie in the middle of the ranking lists. Movies in the data set but not in the top lists get the mean rank of the remaining movies. Thus for male users who have 34 movies in the top 50 list movies outside the list get  $rank_{male,j} = \frac{MaxRank + 34}{2}$ . The  $rank_{top250,j}$  is the rank for the movies in the top 250 list calculated in the same way as the rank for male and female movies.  $MaxRank$  is equal to the number of available movies (1628).

The probability of the user being male or female is defined using the top 50 lists and the top 250 list. According to the IMDb web pages the lists are computed using a true Bayesian estimate. Thus it seems reasonable to assume that the rank of each list has the following relationships:

$$\begin{aligned} r_{m,j} \Pr(Gender = m|m_j, K_{rank}) + r_{f,j} \Pr(Gender = f|m_j, K_{rank}) &= r_{top250,j} \quad (6) \\ \Pr(Gender = m|m_j, K_{rank}) + \Pr(Gender = f|m_j, K_{rank}) &= 1 \quad (7) \\ \Rightarrow \Pr(Gender = m|m_j, K_{rank}) &= (r_{top250,j} - r_{f,j}) / (r_{m,j} - r_{f,j}) \quad (8) \end{aligned}$$

where  $m$  and  $f$  stands for male/female gender.  $r_{g,j}$  is the rank of movie  $m_j$  in the top 50 male, top 50 female or the top 250 ranking lists (not modified as above where the rank is only computed in the context of the available movies of the EachMovie data). We assume that the probability of either gender is the mean value for all movies. After having removed all movies with missing rank values and all movies not satisfying Equation 6 and Equation 7 because they lead to probabilities not equal to 1, we get

$$\Pr(Gender = m|K) = \frac{1}{N} \sum_j \Pr(Gender = m|m_j, K_{rank}) \approx 0.7$$

$$\Pr(Gender = f|K) = 1 - \Pr(Gender = m|K) \approx 0.3$$

$N$  is the total number of used equations similar to Equation 8. The result is consistent with our prior belief that there are more males than females using this kind of service.

### 5.3.5 Extension 2: Cluster-based Rating

In the cluster-based extension the probabilities from the random rating and constant rating schemes are updated in a fashion similar to Extension 1.  $\Pr(a \text{ likes } j|K)$  is updated to  $\Pr(a \text{ likes } j|a \text{ likes } k_1 \wedge k_2, K)$  by randomly picking two clusters  $k_1$  and  $k_2$  and then modifying the odds:

$$\Pr(a \text{ likes } j|a \text{ likes } k_1 \wedge k_2, K) = \frac{odds_{new}}{1 + odds_{new}} \quad (9)$$

$$odds_{new} = \frac{\Pr(a \text{ likes } j|K)}{1 - \Pr(a \text{ likes } j|K)} \times \frac{\Pr(k_1 \vee k_2|m_j, K_c)}{1 - \Pr(k_1 \vee k_2|m_j, K_c)} \quad (10)$$

$$\Pr(k_1 \vee k_2|m_j, K_c) = \Pr(k_1|Movie = m_j, K_c) + \Pr(k_2|Movie = m_j, K_c) \quad (11)$$

The probabilities of the clusters  $\Pr(k_i|Movie = m_j, K_c)$  are given by the clustering algorithm.

## 6 Testing the movie model

### 6.1 Method

#### 6.1.1 Training data

The model was tested using real user profiles from the EachMovie data set. The data set was cleaned from duplicate votes using only the latest vote for each user. Users with less than two votes were removed, leaving a total number of 60087 distinct users.

#### 6.1.2 Parameters

The following parameters were considered:

##### Constant parameters

- The number of real user profiles for testing. For each experimental run, the EachMovie data set was split into two subsets, a test set and a training set. 1000 profiles were randomly drawn for testing, and the rest of the profiles were kept for training.
- The number of rated movies, i.e. the length of the user profiles:
  - The length of real profiles used for testing. Early users have usually rated only a few items. To reflect this, the system was tested using the Given5 protocol of [3]. The Given5 protocol implies that five rated movies are kept in each test profile and that predictions are then made for the rated but withheld movies. Therefore those profiles of the original test set not having enough rated movies (less than or equal to 5 movies) were removed leaving about 900 test profiles in each run.
  - The length of the real user profiles used for training. These profiles were used directly from the EachMovie dataset without taking profile length into consideration. The length would therefore vary.
  - The length of the artificial user profiles. The artificial user profiles were intended to reflect real users. However, the fact that real users only rate some movies was not modeled. All artificial user profiles contained ratings for all movies. The reason for this is that the artificial user profiles are tools for filling the system with experience, and not interesting as individual user models per se.
- The number of neighbors of the k-nearest neighbor algorithm. We could also have varied the parameters for the recommender algorithm. As the number of test runs required was already quite large, we choose to use  $k = 50$  for the k-nearest neighbor algorithm, which has been reported successful for movie recommendations in an earlier work [10].

##### Evaluated parameters

- The sampling model. The sampling models used are the two basic schemes: the random rating and the constant rating schemes, and the two extensions: the rank order-based and the cluster-based extensions.
- The number of movie clusters. We tested the system with 10, 50 and 75 clusters. Since there are 1628 movies, the average number of movies per cluster is about 162, 33 and 21 respectively.

To simplify the discussion, we assume that each cluster contains exactly the average number of movies and that  $k = 1$  for the k-nearest neighbor algorithm. Then, if a real user correlates in taste with an artificial profile that is based on one cluster

from a set of 10 clusters, the user will automatically be recommended 162 movies. In contrast, if the same happens when the number of clusters is 75, only 21 movies will be recommended. Thus, we expect that a high average number of movies per cluster is bad while a low number of movies is good for the performance since the recommendations are more fine-grained.

- The number of sampled, artificial user profiles. We tested the system with 50 and 200 profiles respectively. The choice of 50 profiles was because  $k=50$  for the nearest neighbor algorithm. The higher number, 200, was intuitively chosen to reflect a reasonably well-educated system.
- The number of real user profiles used for training. Since bootstrapping should improve the recommender for early users the training set was partitioned into sets of different sizes such that larger subsets contain all smaller subsets. The partitioning was repeated 5 times for different random permutations of the training set while using the same test set. This is similar to what was done in [23].

Initial tests showed that the performance curves converge somewhere between 100 and 300 real training profiles. Therefore, only partitions of sizes less than or equal to 100 profiles were considered for analysis.

- Recommender algorithm. We tested the bootstrapping algorithms mainly with the Pearson correlation. To test whether the same results could be seen although using a different weighting function we ran some tests with a modified Pearson correlation using default voting as described in [3]. The number of default items was 10,000 and the default rating was two. This means that when two profiles are compared it is assumed that both have votes for their compounded sets of rated items plus 10,000 additional items. If no rating has been given for an item the default vote is assumed to be two. With the modified Pearson less weight is given to negative correlations in Equation 1.

When testing with prior knowledge the system was first trained on the artificial profiles, and then on increasingly larger partitions of the real user profiles. When testing without prior knowledge the artificial profiles were withheld.

The system was run 36 times (15 times for Pearson with default voting) with different sampled profiles and with a different random test sets. The total number of runs in each experiment is thus equal to  $5 \times 36 = 180$  ( $5 \times 15 = 75$ ).

### 6.1.3 Metrics

Due to the fact that the employed k-nearest neighbor algorithm (Equation 1) uses the mean rating of a user as default rating, the coverage of the system was always 100. Performance was therefore measured only for accuracy.

The accuracy of the recommender predictions was measured with the Mean Absolute Error (MAE) and Rank Score measures discussed in [3]. The MAE treats all predictions equally by giving each prediction the same weight. The MAE is suitable for testing systems where each recommendation is equally important. The Rank Score measures the accuracy of recommending rank ordered lists of items where items further down in the list are less important.

All significance tests (two sided with  $p = 0.05$ ) were computed, if not otherwise stated, for pair wise point values of the curves. The null hypothesis was that there is no difference between the compared mean values  $\bar{v}_1$  and  $\bar{v}_2$ . The used test statistic was  $\frac{\bar{v}_1 - \bar{v}_2}{\sqrt{(s_1)^2/n_1 + (s_2)^2/n_2}}$

where  $s_i$  is the sample standard deviation of  $v_i$  and  $n_i$  is the number of measured values of  $v_i$ . The test statistic was assumed to be Student-t distributed with degrees of freedom estimated with the Welch-Satterthwaite approximation [16].

## 6.2 Test Runs: Results and Discussion

To see whether Genesis works we have compared the proposed bootstrapping schemes with and without extensions. The extensions corresponds to bootstrapping based on prior knowledge (informed bootstrapping) while the basic rating schemes are not based on any prior knowledge (uninformed bootstrapping).

In the first section is the random rating scheme with extensions tested. In the second section is the constant rating scheme with extensions tested and in the last section are the best schemes so far tested using the modified Pearson correlation with default voting. The unmodified Pearson correlation is used as weight function in the two first sections.

### 6.2.1 Random Rating Schemes

**Uninformed bootstrapping.** Figure 1 shows the performance of the system with no sampled user profiles compared to 50 and 200 profiles sampled with the random rating scheme. The MAE is significantly ( $p=0.05$ ) better for the random profiles up to the 20 first real user profiles (except for the first point with no real user profiles) where the curves converge. This indicates that the mean vote (the first plotted point) for a user is a good default recommender while adding only a few additional real user profiles degenerates the performance.

By adding randomly drawn user profiles the degeneration is stopped and we get a system that improves in terms of MAE for each additional training example. The improvement is most likely due to the fact that an arbitrary user  $a$  (that the predictions are made for) has a very different rating scheme than another arbitrary real user profile and thus the predictions based on a small set of real users do not work very well. In addition from inspection of the recommender algorithm in Equation 1 we can draw three conclusions: (1) when there are only random profiles the sum on the left hand side will be close to zero since each rating in each profile randomly varies around the profile's mean rating and each profile's contribution is weighted proportional to how close its mean rating is to user  $a$ 's mean rating leading to less contributions from uncorrelated profiles, (2) when adding real profiles they will compete with the random profiles such that the real profiles will only add anything to the predictions if they have more in common with the profile of user  $a$  than the randomly drawn profiles, and (3) thus the random profiles will draw the predicted rating closer to user  $a$ 's mean rating if they do not lose the competition to the real profiles.

In contrast to MAE, the Rank Score curve is much lower from start for the random profiles. The addition of sampled profiles seems to lower the ability to predict the relative order between movies. This is most likely due because random noise is added to each prediction (as described above) such that each prediction is improved compared to its absolute value but not compared to its relative value. Notably, the curve for the 50 sampled profiles crosses the curve with no sampled profiles around 60 real profiles and then they converge somewhere about 200 real user profiles (not shown in Figure 1). The curve for 200 sampled profiles rises much slower probably because adding more sampled profiles also adds more noise to the prediction of the relative order.

**Informed bootstrapping.** In an effort to improve the random rating scheme model the impact of rank order for both males and females was added (the rank order-based extension). This did not affect the performance, probably because there were not enough ranked movies

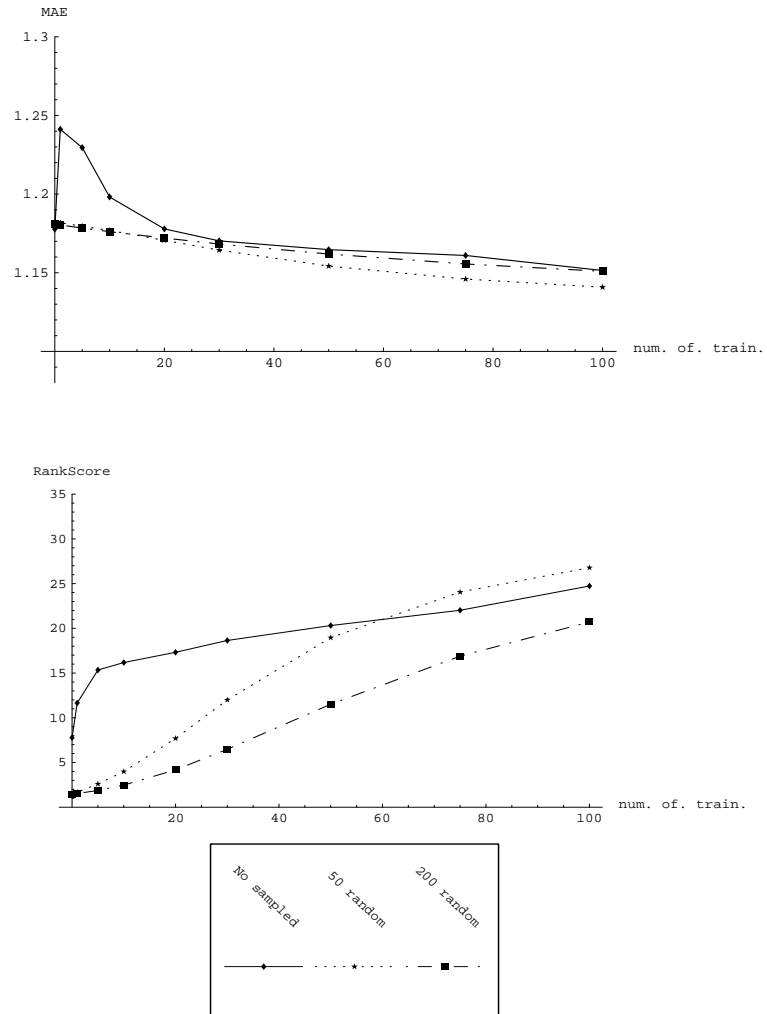


Figure 1: The Mean Absolute Error (MAE) and the Rank Score for the experiment with no sampled user profiles compared to 50 and 200 profiles sampled with the basic random rating scheme. The vertical axis shows the metric and the horizontal axis shows the number of real user profiles used for training. The differences between the curves are significant compared to the No sampled curve for the first 20 profiles excluding 0 and 20 profiles for MAE and for all points except those close to crossing points for Rank Score.

(only about 150 of 1628 movies had a rank). However, it also shows that the algorithm is insensitive to non-correlated profiles as was shown in [20].

Adding the assumption that users like movies in the same cluster where movies are more similar to each other (the Cluster-based Rating extension) did not influence the performance either. There is almost certainly too much noise in the basic random rating scheme.

### 6.2.2 Constant Rating Schemes

**Uninformed bootstrapping.** The performance of the constant rating scheme is not different from the performance of using no sampled profiles. This is not surprising since with the constant rating scheme each movie rating will be equal to the mean rating in each profile and thus the sum is equal to zero in Equation 1.

**Informed bootstrapping.** To improve the performance the impact of rank order was added (the Rank Order-based Rating extension). As in the case of the random rating scheme, this did not affect the result at all.

In contrast, when user profiles were modeled to only give high ratings to movies in the same cluster (the cluster-based rating extension) the result, shown in Figure 2, resembles the result for the random rating scheme. This is not surprising since there is a random element in choosing which clusters to rate high.

Figure 2 and Figure 3 show that only having 10 clusters generally decrease the performance for both MAE and Rank Score compared to using the random rating scheme with 50 sampled profiles. Using 200 sampled profiles decreases the Rank Score but seems to give slightly better MAE, comparable to the random rating scheme. However, when using 50 sampled user profiles the MAE does not differ significantly for either 50 or 75 clusters compared to the random rating scheme. In contrast, the Rank Score for the cluster-based rating scheme rises faster in the beginning than the random rating scheme but converges earlier, at about 80 real profiles, with the curve of no sampled profiles.

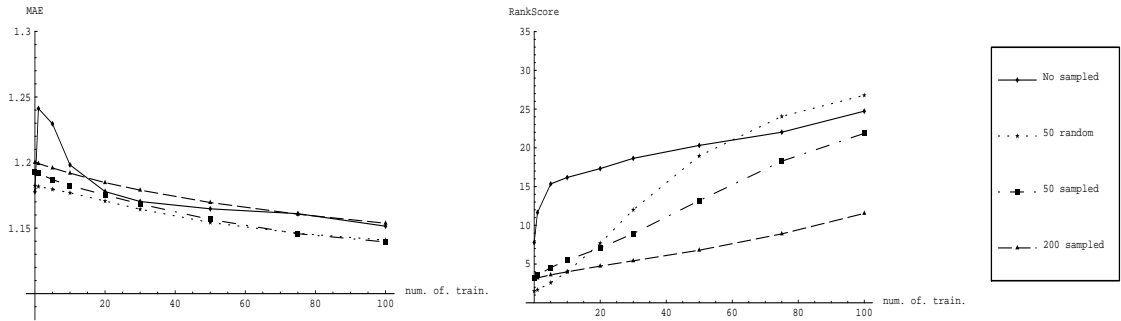
Thus the cluster-based rating scheme with 50 sampled profiles and 50 or 75 clusters seems to improve the performance over the random rating scheme though the MAE is slightly better for the latter.

### 6.2.3 Pearson with default voting

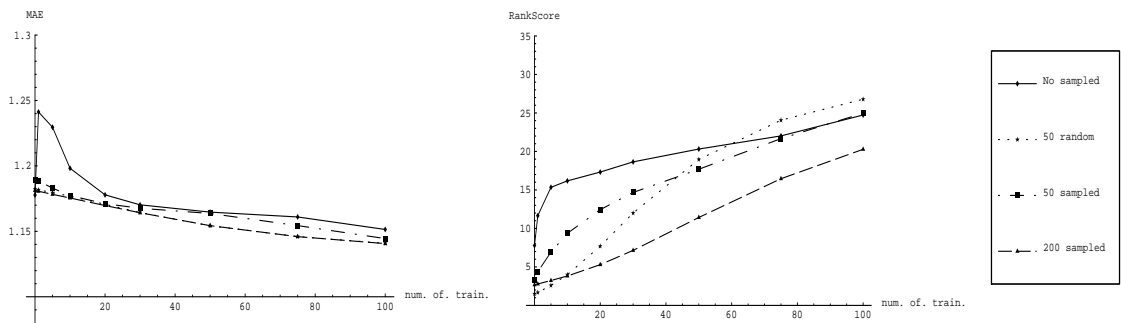
When changing correlation function to Pearson with default voting the Rank Score is greatly improved as shown in Figure 4. We note as well that the choice of correlation function is thus more important for the performance than the choice of bootstrapping scheme. However we still get similar results for the bootstrapping compared to using the original Pearson correlation. The constant rating scheme with the cluster-based extension (the Informed curve) is still better than the random rating scheme (the Random curve) for Rank Score. In addition the MAE is improved. While Random and No sampled 2 converge at about 20 real profiles, Informed still has a lower MAE though the difference between them are insignificant. However, while the difference of No sampled 1 compared to No sampled 2 or Random is not significant, the difference between No sampled 1 and Informed is in fact significant!

## 7 Concluding remarks

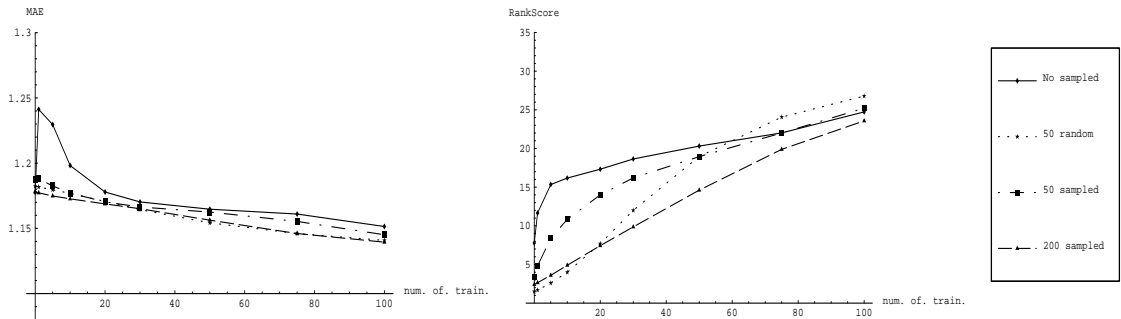
In this paper we presented the Genesis method for bootstrapping a recommender system with prior knowledge. The knowledge is represented as a probabilistic model of user be-



a) Bootstrapping based on 10 clusters.

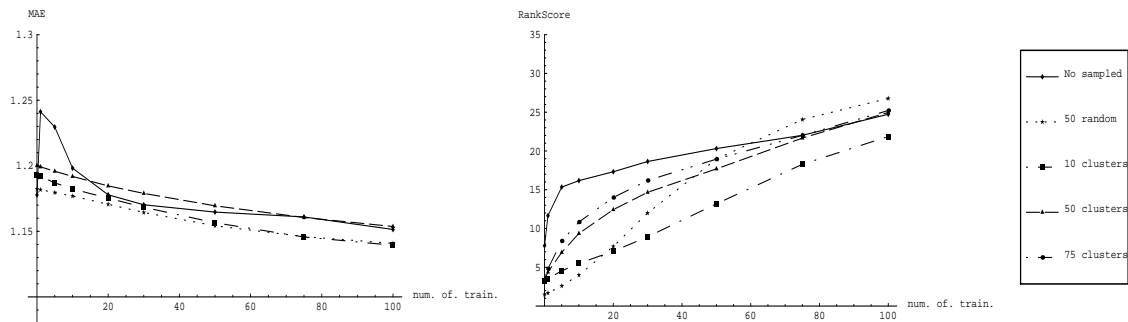


b) Bootstrapping based on 50 clusters.

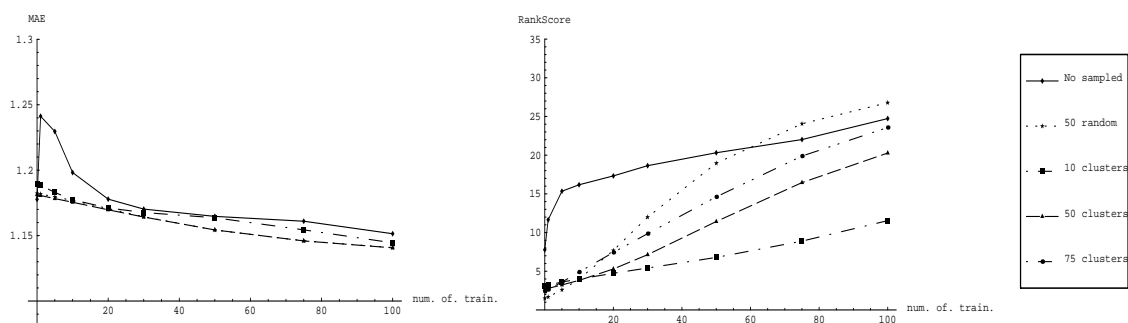


c) Bootstrapping based on 75 clusters.

Figure 2: The Mean Absolute Error (MAE) and the Rank Score for the experiment with no sampled user profiles compared to 50 and 200 profiles sampled with the constant rating scheme with the cluster-based extension. As comparison the curve for 50 profiles sampled with the random rating scheme is shown. The vertical axis shows the metric and the horizontal axis shows the number of real user profiles used for training. The differences between the curves are significant compared to the No sampled curve for first 20 profiles excluding 0 and 20 profiles for MAE and for all points except those close to crossing points for Rank Score.



a) Bootstrapping with 50 sampled profiles.



b) Bootstrapping with 200 sampled profiles.

Figure 3: The Mean Absolute Error (MAE) and the Rank Score for the experiment with no sampled profiles compared to profiles sampled with the cluster-based rating scheme for 10, 50 and 75 clusters respectively. As comparison the curve for 50 profiles sampled with the random rating scheme is shown. The vertical axis shows the metric and the horizontal axis shows the number of real user profiles used for training. The differences between the curves are significant compared to the No sampled curve for the first 20 profiles excluding 0 and 20 profiles for MAE and for all points except those close to crossing points for Rank Score.

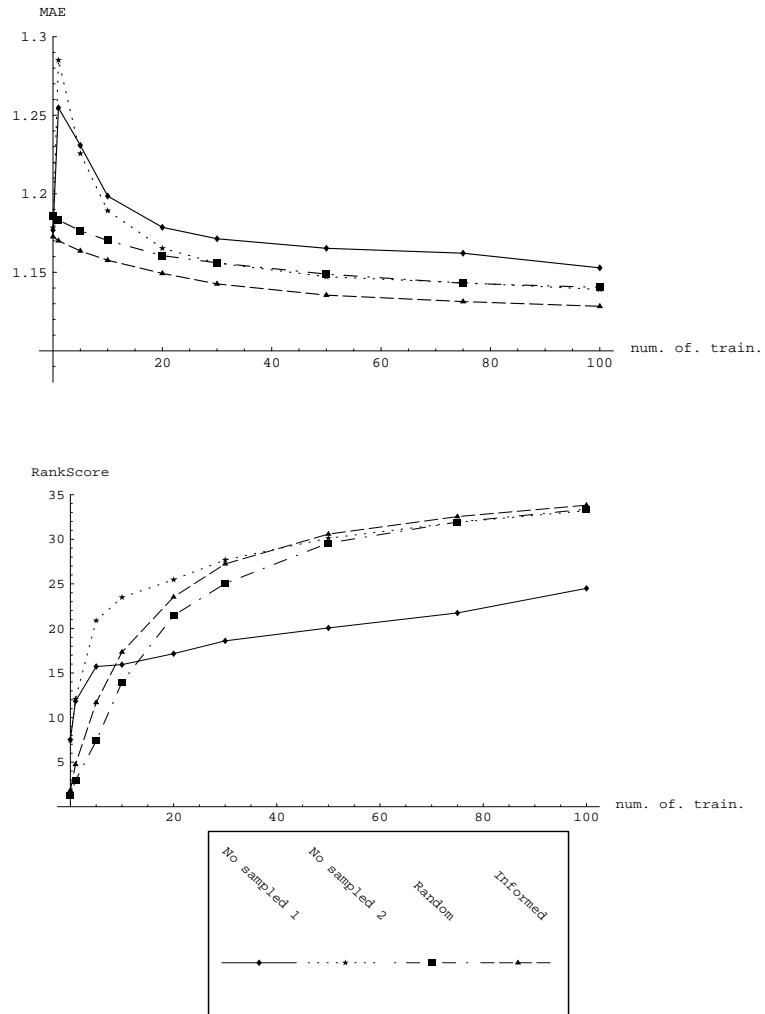


Figure 4: The Mean Absolute Error (MAE) and the Rank Score for the experiment with no sampled profiles for: Pearson correlation (No sampled 1) and Pearson correlation with default voting (No sampled 2), compared to 50 profiles sampled with the random rating scheme (Random) and to 50 profiles sampled with the constant rating scheme with the cluster-based extension using 75 clusters (Informed). The two last curves use default voting. The vertical axis shows the metric and the horizontal axis shows the number of real user profiles used for training. The differences between the Random and Informed curves are significant compared to (1) No sampled 1 for the first 10 profiles excluding 0 and 10 profiles for MAE and for all points after 20 profiles for Rank Score and (2) No sampled 2 for the first 10 profiles excluding 0 and 10 for MAE and after 20-30 profiles for Rank Score.

havior in a domain. Artificial user profiles are created by instantiating specific user models drawn from the general model. The system is then bootstrapped with a number of such profiles. The method was tested in the domain of movie recommendation. A k-nearest neighbor algorithm was bootstrapped with different sets of artificial user profiles. The system was then trained and tested on real user profiles from the EachMovie dataset. Performance was measured for accuracy using Mean Absolute Error and Rank Score. The bootstrapped system was compared to a system without sampled profiles.

A surprising result was that by modeling users with a completely random rating scheme we were able to improve the MAE significantly up to 20 real training profiles. However this also lowered the Rank Score significantly.

By clustering movies into groups of similar movies and then model users to like movies in the same group (the cluster-based rating extension to the constant rating scheme) we were able to improve Rank Score over the random rating scheme.

We have made the following conclusions with the respect to the proposed rating schemes:

- If high Rank Score is preferred, the proposed bootstrapping schemes are not needed.
- When MAE is the most important measure, the constant rating scheme with the cluster-based rating extension should be used.
- By extending the constant rating scheme with the cluster-based rating extension the MAE is low while at the same time Rank Score is higher for small number of real training profiles than for the random rating scheme.
- The number of sampled user profiles and the number of clusters seem to be crucial for the system performance with respect to a combination of both having low MAE and high Rank Score.

In other words, when predictions are made for: (1) individual items the constant rating scheme with the cluster-based rating extension should be used (thus decreasing the MAE) and (2) a list of the best recommended items none of the proposed bootstrapping schemes should be used (thus keeping the Rank Score high).

However, the best solution might be to get at least 20 users to rate many items in advance to get the system going or to combine these users and prior knowledge to produce clusters that more realistically reflects real users. In addition instead of using the plot there might be a content attribute, for instance actors or a combinations of attributes, that might better reflect quality demands and tastes of real users.

## References

- [1] Balabanovic, M. and Shoham, Y. 1997. Fab: Content-based, Collaborative Recommendation. In: Communication of the ACM, Vol 40, No 3.
- [2] Basu, C., Hirsh, H., and Cohen, W.W. 1998. Recommendation as Classification: Using Social and Content-Based Information in Recommendation. Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI98). AAAI Press/MIT Press.
- [3] Breese, J.S., Heckerman, D. and Kadie, C. 1998. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. MSR-TR-98-12. May (revised October 1998).
- [4] Burke, R. 2000. Knowledge-based Recommender Systems. In A. Kent (ed.), Encyclopedia of Library and Information Systems. Vol. 69, Supplement 32. New York: Marcel Dekker.

- [5] Burke, R.D. 2000. A Case-Based Reasoning Approach to Collaborative Filtering. EWCBR 370-379.
- [6] Cheeseman, P. and Stutz, J. 1996. "Bayesian Classification (AutoClass): Theory and Results", in *Advances in Knowledge Discovery and Data Mining*, Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, & Ramasamy Uthurusamy, Eds. AAAI Press/MIT Press.
- [7] Cöster, R., Gustavsson, A., Olsson, T. and Rudström, Å. 2002. Enhancing Web-Based Configuration with Recommendations and Cluster-Based Help. In *Proceedings of the AH'2002 Workshop on Recommendation and Personalization in eCommerce*, Málaga, Spain, May 28th. pp 30-39.
- [8] Dempster, M.M., Laird, N.M. and Jain, D.B. "Maximum likelihood from incomplete data via the EM algorithm," *J. Royal Stat. Soc., Series B*, vol. 39, pp. 1-38.
- [9] Good, N., Schafer, J., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J. and Riedl, J. 1999. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*.
- [10] Herlocker, J., Konstan, J., Borchers, A. and Riedl, J. 1999. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval*, August.
- [11] Hill, W., Stead, L., Rosenstein, M. and Furnas, G. 1995. Recommending And Evaluating Choices In A Virtual Community Of Use. *CHI '95 Proceedings: Conference on Human Factors in Computing Systems: Mosaic of Creativity Sponsored by SIGCHI* May 7 -11 1995, Denver, CO. ISBN: 0-89791-694-8.
- [12] Hofmann, T. 1999. "Probabilistic Latent Semantic Analysis," *UAI (99)*, Morgan Kaufmann Publishers, Inc., San Francisco.
- [13] Schafer, J. B., Konstan, J. A. and Riedl, J. 2001. E-commerce recommendation applications. *Data Mining and Knowledge Discovery* , vol. 5 nos. 1/2, pp 115-152
- [14] Mitchell, T.M. 1997. *Machine Learning*. McGRAW-HILL Companies, Inc
- [15] Mooney, R.J. 2000. Integrating Abduction and Induction in Machine Learning. Appears in *Abduction and Induction*, P. A. Flach and A. C. Kakas (Eds.), pp. 181-191, Kluwer Academic Publishers, Norwell, MA.
- [16] NIST/SEMATECH e-Handbook of Statistical Methods, <http://www.itl.nist.gov/div898/handbook/>, 2002 December 12.
- [17] Popescul, A., Ungar, L.H., Pennock, D.M. and Lawrence, S. 2001. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, to appear, Seattle, WA, August.
- [18] Porter, M.F. 1980. An algorithm for suffix stripping. *Program*, 14(3) :130-137.
- [19] Salton, G. and McGill, M.J. 1983. *Introduction to Modern Information Retrieval*. McGraw Hill, New York.
- [20] Sarwar, B.M., Konstan, J.A., Borchers, A., Herlocker, J., Miller, B. and Riedl, J. 1998. Using filtering agents to improve prediction quality in the GroupLens research collaborative filtering system. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*.

- [21] Shardanand, U and Maes, P. 1995. Social Information Filtering: Algorithms for Automating "Word of Mouth". In: Proceedings of 1995 Conference on Human Factors in Computing Systems (CHI'95). ACM Press.
- [22] Svensson, M. and Höök, K. "Social Navigation of Food Recipes: Designing Kalas". In Benyon, Hook, and Moore (Eds), "Designing Information Spaces: The Social Navigation Approach". Springer Verlag, 2003.
- [23] Towell, G.G. and Shavlik, J.W. 1994. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70:119-165.
- [24] Ungar, L.H. and Foster, D.P. 1998. A Formal Statistical Approach to Collaborative Filtering. Conference on Automated Learning and Discovery (CONALD)



### A.3 Paper III



# Enhancing web-based configuration with recommendations and cluster-based help

Rickard Cöster, Andreas Gustavsson, Tomas Olsson, and Åsa Rudström

Swedish Institute of Computer Science, SICS, Box 1263

SE-164 29 Kista, Sweden

{rick, angus, tol, asa}@sics.se

<http://www.sics.se/>

**Abstract.** In a collaborative project with Tacton AB, we have investigated new ways of assisting the user in the process of on-line product configuration. A web-based prototype, RIND, was built for ephemeral users in the domain of PC configuration.

Two mechanisms were added to a commercial configurator produced by Tacton: i) automated recommendations that display social trails associated with the configuration; and ii) a help system based on term clustering. Recommendations based on previous customer selections are made on separate attributes as well as full configurations, i.e. complete PCs. The early rater problem is solved using a probabilistic bootstrapping approach. The help system supports novice users browsing for help information, as well as experienced users able to pose exact queries.

## 1 Introduction

Configurators assist users in selecting attributes and features such as customer requirements and product attributes of a complex product. In a collaborative project with Tacton AB, a commercial company selling and marketing a constraint-based configurator, we have investigated new and complementary ways of assisting the user in the task of configuring complex products. The result of the project is a prototype, RIND, in which we have added i) automated, collaborative recommendations for displaying social trails associated with the configuration, and ii) a help interface. The prototype is built on top of the Tacton configurator in the domain of PC configuration.

By tagging the configuration process with social information we aim to give the user a better idea of which attributes or complete configurations that are common or not. The recommendations guide the user by displaying two types of social trails. First, the user can get recommendations on selected product attributes. Attributes that are recommended are those that other people have selected in similar situations. Second, the user can ask: Given my current selections, what is the most popular final product configuration?

The help system is designed to adapt to the current user selections in the configuration. For experienced users, detailed help can easily be found with keyword search. For users who are new to the product domain, the structure of the help system functions as a guide to the domain.

Using a rule-based knowledge system, the configurator calculates and displays which attributes are compatible with the user's previous selections. Whenever a choice is incompatible, the configurator will inform the user what needs to be changed in order to keep her most recent selection. The configurator can also select attributes that optimize some product variable, e.g. price. In this way, a user may select the most important product attributes and let the configurator select all the other.

The configurator is good at handling product attributes. Adding recommendations and cluster-based help takes us yet another step closer to the underlying customer needs.

## 2 The RIND Prototype

Several different user categories can be identified in the domain of on-line PC configuration: ephemeral/frequent user; novice/expert; buying for oneself or for others; customer or sales person (sales support, help desk) etc. In addition, the system could be designed to be stand-alone or a web client; to be accessed over the Internet/extranet/intranet; using a login procedure or not. The design of the system is fundamentally dependent on the choices made for these factors. The RIND prototype is a web client designed for users that only use the system

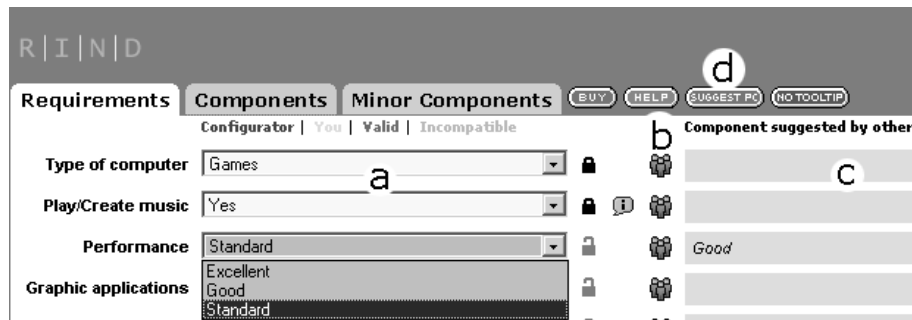


Fig. 1. Interface detail

once or twice. Part of the interface is depicted in Figure 1. Ephemeral users should not be required to log in to use the system until the time of purchase. In light of this, it is hard to anticipate the background of the user - novice or expert? Our hypothesis is that ephemeral users may be unfamiliar to the use of a configurator, but not necessarily unfamiliar with the domain. We also assume that users purchasing products over the Internet can be assumed to be reasonably familiar with computer applications as such.

We believe that it is in this context that recommendations and help interfaces increase the benefit to the user.

Our approach differs from other approaches combining knowledge based reasoning and collaborative filtering in two respects. Firstly, there is no explicit model of user preferences, ratings, or the like, such as discussed in e.g. [1]. Only complete configurations are stored and used as a basis for recommendation. Secondly, our system is not a hybrid system combining the two techniques, as suggested by [10] or [6]. Instead, the results from the recommender engine are presented alongside the results from the constraint-based configurator.

The PC configuration domain is expressed in 35 attributes, listed in Table 1. Each attribute has 2 to 18 values. 12 of the 35 attributes are not settable by the user (e.g. price excl. VAT), and are therefore not used in the recommendation process. 8 attributes represent user requirements (e.g. whether the user will play/create music), and the remaining 15 are computer components. For better overview, the 23 settable attributes are grouped into Requirements, Components, and Minor Components. This grouping is reflected in the interface, where each group is displayed on a separate tab. The demands on the interface are high.

<i>Requirements</i>	<i>Components</i>	<i>Minor components</i>	<i>Non-settable attributes</i>
Type of computer	Service	Mouse	Performance value
Play/Create music	Monitor	CD/DVD	Max days to delivery
Performance	Hard Drive	Keyboard	Price
Graphic applications	OS	Speaker	Price incl.VAT
Minimum component quality	Video Card	Network Card	Total RAM
Minimum RAM size	Extra Video Card	Mother Board	memory cards (5 attrrs.)
Minimum hard drive size	Cpu	Computer Case	Free Dimm slots
Delivery		Sound Card	Floppy Drive

**Table 1.** PC domain attributes

For each attribute, a large number of different possibilities need to be displayed. Each attribute can take on a number of values, displayed as a color coded pull-down menu (a in Fig. 1). The configurator's choice is presented first in the list (blue), followed by valid (green) choices and choices that are incompatible with the user's previous selections (orange). Once a value is selected (yellow), the lock icon to the right of the value list closes.

Next, recommendations can be made on single attributes as well as on full configurations. To maintain a reasonable complexity in the interface, recommendations are separated from the selection of values. Recommendations on single attributes are given on demand (clicking on the icon b in Fig. 1), and are displayed to the right (c in Fig. 1). The most popular full configurations are recommended in a separate window (clicking on icon d). This sub-window has a similar but simplified interface, with the possibility to select all attributes of one of the displayed configurations with a single click.

RIND is an HTML/JavaScript client, communicating with three logical servers: the configurator, the recommender engine, and the help system. Data is commu-

licated in XML format, and the client makes extensive use of the XML DOM provided by MS Internet Explorer 5.5 and higher.

### 3 Recommendations

For recommendations, we use a recommender engine developed at SICS. This engine supports both collaborative and content-based filtering, and has previously been used in the EFOL project [9] recommending food recipes, and in GeoNotes, a position-based system for annotation of physical places [2].

At a conceptual level, the architecture of the recommender engine is very simple. Each user's interests are represented in a profile. In the case of collaborative filtering, the profile is usually a feature vector of the user's explicit or implicit votes. In other cases, e.g. content-based filtering, a classifier or regression machine may be trained to learn the user's interest model.

In RIND, each completed and purchased product configuration is represented and stored as a feature vector. The component choices are represented as the feature values. Each choice is interpreted as an implicit vote for the corresponding component value. The current user's configuration is used as a query to the recommender, but is not stored until the final purchase.

For this specific project we have developed three basic algorithms for filtering, which are all based on neighborhood formation and prediction from the neighborhood. The distance between two profiles is the number of features with equal values. We write  $v_a$  to denote the profile of user  $a$ , and  $v_{a,j}$  for the value of feature  $j$  in that profile. Predictions are denoted  $p$  and use the same subscript notation as profiles. Furthermore, we define  $X(a, k)$  as the  $k$  nearest neighbors to user  $a$ .

#### 3.1 Weighted Majority Voter

The simplest algorithm is the Weighted Majority Voter, which predicts the value of a component on the basis of a weighted majority vote of the  $k$  nearest neighbors. The weight is set equal to the distance between two user profiles, i.e. the number of equal component values in the two configurations. The prediction  $p$  for user  $a$  on item  $j$  is then

$$p_{a,j} = \operatorname{argmax}_{s \in S_j} \sum_i w(a, i) [v_{i,j} = s]$$

where  $i \in X(a, k)$ ,  $S_j = \{v_{i,j}\}$  and  $w(a, i) = \sum_j [v_{a,j} = v_{i,j}]$ .

#### 3.2 Most Popular Choice

The second algorithm, Most Popular Choice predicts the most popular entire configuration of the  $k$  nearest neighbors. The algorithm is an extended Naïve Bayes classifier using  $m$ -estimate for estimating the probabilities [5]. Because the

prediction regards entire configurations, the Naïve Bayes classifier is extended to handle multiple components. The sought components (components not already chosen by the user) are assumed to be independent.

The  $m$ -estimate gives weight to the probability according to the user's current choices even when there is no configuration with all of the user's choices. The extension to many components gives weight to the probability according to the popularity (= number of occurrences) of the sought component values. These two properties lead to the following (good) characteristic; if the user has not chosen any component-value pairs or if no stored configuration has the component-value pairs of the user's choice, then the configuration with the most popular component-value pairs will be recommended. Otherwise, the probability of a configuration is weighted according to whether it has any of the component-value pairs of the user. The prediction is

$$p_a = \operatorname{argmax}_i \Pr(\{v_{i,u} | u \in \bar{J}\}) \times \prod_{j \in J} \Pr(v_{a,j} | \{v_{i,u} | u \in \bar{J}\})$$

where  $J = \{j \mid v_{a,j} \neq \{\}\}$ , the set of features for which user  $a$  have selected values, and  $\bar{J}$  is the set of features for which  $a$  has not yet selected any values. We let  $\Pr(\{v_{i,u} | u \in \bar{J}\}) = \prod_{u \in \bar{J}} \Pr(v_{i,u})$  by assuming independency.

For the probabilities, we count the frequencies over the neighbors as follows. We let  $\Pr(v_{i,u}) = f_u/k$  where  $f_u$  is the number of neighbors with component  $u$  set to the value  $v_{i,u}$ . We define  $\Pr(v_{a,j} | \{v_{i,u} | u \in \bar{J}\})$  as  $(f_{i-a+j} + mp)/(f_{i-a} + m) = (f_{i-a+j} + 1)/(f_{i-a} + k)$  when as  $m$ -estimate we use  $m = k$  and  $p = 1/k$  (the latter is the smallest probability any component value can have except for the zero probability).  $f_{i-a}$  is the number of neighbors with the same component-value pairs as  $v_i$  except for all components set in  $v_a$ . Finally,  $f_{i-a+j}$  is the number of neighbors with the same component-value pairs as  $v_i$  except for all components in  $v_a$  but component  $j$  that has value  $v_{a,j}$  (that is  $v_{i,j} = v_{a,j}$ ). This means that  $f_{i-a+j} > 0$  only if  $v_{i,j} = v_{a,j}$ .

### 3.3 Naïve Bayes Voter

The third and last algorithm is the Naïve Bayes Voter, which is similar to the Weighted Majority Voter but uses Naïve Bayes with  $m$ -estimate as a basis for the predictions. Instead of using the sum of the distances between the user profiles as a vote, it uses the probability of a component value given the  $k$  nearest neighbors. Thus the most probable value for any component is recommended. As  $m$ -estimate we use again  $m = k$  and  $p = 1/k$ . The  $m$ -estimate makes it possible to recommend values for all components regardless of whether any other user profile contains all the component-value pairs of the active user.

$$p_{a,j} = \operatorname{argmax}_{s \in S_j} \Pr(v_{i,j} = s) \prod_{s' \in v_a} \Pr(v_{a,t} = s' | v_{i,j} = s)$$

where again  $S_j = \{v_{i,j}\}$ . For the frequencies, we define  $\Pr(v_{i,j} = s) = f_{j=s}/k$  where  $f_{j=s}$  is the number of neighbors with value  $s$  for feature  $j$ . Furthermore,

we let  $\Pr(v_{a,t} = s' | v_{i,j} = s) = (f_{j=s \wedge t=s'} + 1) / (f_{j=s} + k)$  where  $f_{j=s \wedge t=s'}$  is the number of neighbors with both value  $s$  for feature  $j$  and value  $s'$  for feature  $t$ .

## 4 Bootstrapping

When starting up any recommender system every developer will be faced with the early-rater problem. Without any users it is hard to make good recommendations. Even though there are many users, making good recommendations can be hard when each item has only a few user opinions and the overlap between the opinions is sparse.

The ordinary approach to tackle this problem is to combine entirely opinion based collaborative filtering with content-based methods, expecting the two approaches to level out each other's weaknesses [8]. In this project, the content is already used as the base for recommendations and hence another approach is required. We propose an approach that makes use of prior knowledge acquired from asking experts, analyzing the recommended items, and using prejudices and good guesses. The acquired knowledge is often uncertain and thus represented with a probabilistic model. Simulated user profiles can then be sampled from the probabilistic model and bootstrap any used learning algorithm including the  $k$  nearest neighbor.

### 4.1 The probabilistic model

The current RIND implementation cannot track individual users and thus only dependencies between attributes and not between users are modeled. The prior knowledge is encoded as a Bayesian belief net, which can be constructed with any schoolbook procedure for knowledge acquisition [4]. The prior knowledge modeled in the belief net is based on:

- Normally, users prefer cheap products.
- People who want to play games usually need graphics and music.
- An interest in graphics indicates a need for a good monitor and graphics card.
- An interest in music indicates a need for a good sound card and speakers.
- An interest in both graphics and music indicates a need for a larger RAM memory, a larger hard disk and good performance.
- For variables without good value distribution guesses, uniform probabilities are used. For example, each type of computer (standard/build your own/games/business) gets a uniform probability of 0.25.

User profiles used for bootstrapping the system are then sampled from the belief net by probabilistically selecting feature values in random order. Only values validated by the configurator are considered. If there is no selectable valid value for a feature, the user profile is discarded and instead a new profile is formed. Every time a value is selected for a feature the posterior probability of the belief

net is recomputed given the currently selected feature values. The procedure is then repeated until there are no more selections left for each user profile.

The proposed approach of course makes it harder to discern the meaning of a recommendation, at least when the recommendations are based on very few real users. A recommendation is no more only about whether other people liked or disliked an item but also what the system designer recommends. This makes the problem of visualizing a recommendation more complicated. We could partly get around this problem by distinguishing between recommendations based on real user profiles and sampled profiles in the user interface.

## 5 RIND Help System

The key requirement for the help system is to guide ephemeral users to those help documents that meet their information request, independent of previous knowledge of the PC domain. Requirements such as maintaining consistent help for newly released product information and integration with the configuration interface are also considered as central.

To support ephemeral users with varying domain knowledge, the help system needs a navigation tool that is suitable for novices as well as for experts. Our approach presents a hierarchic view of the domain with terms that are more general or more specific in relevance to each other based on the user query. A novice can browse the domain and refine the query by selecting a more specific or more general term in the hierarchic view. By entering a precise query an expert can search for help documents that fit the information request, but also refine the query using the hierarchic view.

### 5.1 The Help Web Interface

The help interface (fig. 2) is separated into three parts. The user query is displayed at the top as an editable input box. In addition to input from the user, the input box is also used to display the current hierarchic location in the help domain. In the bottom part, the hierarchic view can be found to the left, while the help text itself is displayed to the right. Some selectable terms are embedded in the text and have the same functionality as a selectable spread topic.

In the hierarchical view, more general terms are displayed under the Spread Topics header. If the user selects the spread topic "PC" in figure 2 the help interface is updated and "PC" becomes the new user query. In this case the user will see how the previous query term "motherboard" is related to other topics also related to PC.

Terms displayed under the Narrow Topics header are terms that are subtopics to the current user query. When the user selects the narrow topic "performance" the term is added to the user query and the interface is updated accordingly.

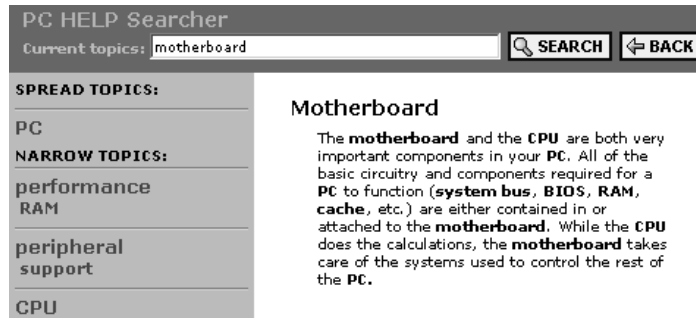


Fig. 2. Help interface

## 5.2 Clustering Method

Van Rijsbergen [11] defines two types of automated methods for associating documents with each other. In the polythetic clustering method the membership between a document and the cluster is based on how many of the document terms that are in cluster defined terms. The problem with polythetic methods is that not all the terms defining the clusters are guaranteed to be present in the documents.

In the monothetic clustering method, all terms defined by the cluster are guaranteed to exist in all documents. A common form of monothetic cluster hierarchy is taxonomic web directories such as Yahoo!. These directories manually place web pages into a hierarchical structure. One problem with this approach is that most texts are not taxonomically related, but discuss different topics simultaneously. A second problem is that the web pages have to be manually placed into the structure. Finally, users have to learn and remember the structure of the hierarchy.

Sanderson and Croft [7] attempt to overcome these problems by automatically constructing a concept hierarchy derived from a set of documents. They extracted terms for the hierarchy from the documents such that a parent term would refer to a more general concept than its child.

The RIND help system uses a similar approach but with predefined terms describing each help text, and a monothetic clustering method with additional polythetic clustered terms describing the monothetic term cluster. The predefined terms are based on a controlled dictionary without any synonyms, since the attempt is to solve the polysemy part of the vocabulary problem [3].

## 6 Concluding Remarks

We have identified a domain in which recommendations and cluster-based help systems seem well suited to assist ephemeral users in the task of selecting product

configurations. To this end, we have built a prototype on top of a configurator to demonstrate our approach.

We found several ways of performing recommendations by using the database of purchased products, and we have also developed a principled way of bootstrapping recommender systems with prior knowledge and good guesses.

A key requirement for the prototype was that the help system should be able to support both novice and expert users. We have therefore developed a cluster-based help system in which experts can find exact information and novice users can see a hierarchical structure of how components or parts of the configuration are related.

## 7 Acknowledgements

This work was performed in a collaborative project between Tacton AB ([www.tacton.se](http://www.tacton.se)) and the Swedish Institute of Computer Science, SICS ([www.sics.se](http://www.sics.se)), supported by funding from VINNOVA, the Swedish Agency for Innovation Systems.

## References

1. L. Ardissono, A. Felfernig, D. Jannach, G. Friedrich, R. Schäfer, and M. Zanker. Customer-adaptive and distributed online product configuration in the CAW-ICOMS project. In *Proceedings of the Configuration Workshop at IJCAI 01*, Seattle, USA, 2001.
2. F. Espinoza, P. Persson, A. Sandin, H. Nyström, E. Cacciatore, and M. Bylund. Geonotes: Social and navigational aspects of location-based information systems. In Abowd, Brumitt, and Shafer, editors, *UbiComp 2001: Ubiquitous Computing, International Conference*, pages 2–17, Atlanta, Georgia, 2001. Berlin: Springer.
3. G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, 1987.
4. A. Gonzalez and D. Dankel. *The engineering of knowledge-based systems: Theory and practice*, 1993.
5. T. Mitchell. *Machine Learning*. New York: McGraw Hill, 1997.
6. Martin Molina. An intelligent sales assistant for configurable products. In Ning Zhong, Yi Yu Yao, Jiming Liu, and Setsuo Ohsuga, editors, *Proceedings of the 1st Asia-Pacific conference on Web Intelligence: Research and Development*, volume 2198 of *Lecture Notes in Computer Science*. Springer, 2001.
7. M. Sanderson and W. B. Croft. Deriving concept hierarchies from text. In *Research and Development in Information Retrieval*, pages 206–213, 1999.
8. Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, Jonathan L. Herlocker, Bradley N. Miller, and John Riedl. Using filtering agents to improve prediction quality in the groupLens research collaborative filtering system. In *Computer Supported Cooperative Work*, pages 345–354, 1998.
9. M. Svensson, K. Höök, J. Laakso, and A. Waern. Social navigation of food recipes. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 341 – 348, Seattle, Washington, 2001.

10. Thomas Tran and Robin Cohen. Hybrid recommender systems for electronic commerce. In *Papers from the Seventeenth National Conference on Artificial Intelligence (AAAI-2000) Workshop on Knowledge-Based Electronic Markets*, pages 78–84. AAAI Press, 2000.
11. C. J. Van Rijsbergen. *Information Retrieval*. Dept. of Computer Science, University of Glasgow, 1979.



## **Recent licentiate theses from the Department of Information Technology**

- 2002-002** Anders Berglund: *On the Understanding of Computer Network Protocols*
- 2002-003** Emad Abd-Elrady: *Harmonic Signal Modeling Based on the Wiener Model Structure*
- 2002-004** Martin Nilsson: *Iterative Solution of Maxwell's Equations in Frequency Domain*
- 2002-005** Kaushik Mahata: *Identification of Dynamic Errors-in-Variables Models*
- 2002-006** Kalyani Munasinghe: *On Using Mobile Agents for Load Balancing in High Performance Computing*
- 2002-007** Samuel Sundberg: *Semi-Toeplitz Preconditioning for Linearized Boundary Layer Problems*
- 2002-008** Henrik Lundgren: *Implementation and Real-world Evaluation of Routing Protocols for Wireless Ad hoc Networks*
- 2003-001** Per Sundqvist: *Preconditioners and Fundamental Solutions*
- 2003-002** Jenny Persson: *Basic Values in Software Development and Organizational Change*
- 2003-003** Inger Boivie: *Usability and Users' Health Issues in Systems Development*
- 2003-004** Malin Ljungberg: *Handling of Curvilinear Coordinates in a PDE Solver Framework*
- 2003-005** Mats Ekman: *Urban Water Management - Modelling, Simulation and Control of the Activated Sludge Process*
- 2003-006** Tomas Olsson: *Bootstrapping and Decentralizing Recommender Systems*



UPPSALA  
UNIVERSITET