

PocketLens: Toward a Personal Recommender System

BRADLEY N. MILLER, JOSEPH A. KONSTAN, and JOHN RIEDL
University of Minnesota

Recommender systems using collaborative filtering are a popular technique for reducing information overload and finding products to purchase. One limitation of current recommenders is that they are not portable. They can only run on large computers connected to the Internet. A second limitation is that they require the user to trust the owner of the recommender with personal preference data. Personal recommenders hold the promise of delivering high quality recommendations on palmtop computers, even when disconnected from the Internet. Further, they can protect the user's privacy by storing personal information locally, or by sharing it in encrypted form. In this article we present the new PocketLens collaborative filtering algorithm along with five peer-to-peer architectures for finding neighbors. We evaluate the architectures and algorithms in a series of offline experiments. These experiments show that PocketLens can run on connected servers, on usually connected workstations, or on occasionally connected portable devices, and produce recommendations that are as good as the best published algorithms to date.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Storage and Retrieval—*Information filtering*

General Terms: Algorithms, Measurement, Security

Additional Key Words and Phrases: Collaborative Filtering, Peer-to-Peer Networking, Recommender Systems, Privacy

1. INTRODUCTION

Since the advent of the Web, nine short years ago, more than 7,500 terabytes of information have gone online. More than 513 million people around the world now have access to this global information resource [Lyman and Varian 2000]. But, how do all of these people find the information they are most interested in from among all of those terabytes? And, how do they find the other people they would most like to communicate with, work with, and play with?

We would like to acknowledge the financial support to this project provided by the National Science Foundation under grants: DGE 95-54517, IIS 96-13960, IIS 99-78717, and IIS 01-02229; and by Net Perceptions, Inc., a recommender systems company cofounded by us and two colleagues.

Authors' addresses: B. Miller, Computer Science Department, Luther College, 700 College Drive, Decorah, IA 52101; email: millbr02@luther.edu; J. A. Konstan and J. Riedl, Department of Computer Science and Engineering, University of Minnesota, 4-192 EE/CS Building, 200 Union Street SE, Minneapolis, MN 55455; email: {konstan,riedl}@cs.umn.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.

© 2004 ACM 1046-8188/04/0700-0437 \$5.00

Increasingly, people are turning to recommender systems to help them find the information that is most valuable to them. Recommender systems support a broad range of applications, including recommending movies, books, relevant search results, and even pets. One of the most successful technologies in recommender systems research and practice is collaborative filtering.

Collaborative filtering works by looking for patterns of agreement among the ratings of groups of people [Resnick et al. 1994; Shardanand and Maes 1995]. Ratings are expressions of interest by users for items. The ratings can either be explicit indications, typically on a 1–5 scale, or implicit indications, such as purchases or click-throughs. The collaborative filtering system predicts which items a user is likely to rate highly, purchase, or click on.

Two key problems in recommenders that remain to be solved revolve around the issues of portability and trust. *Portability* refers to the user's preference to be able to obtain good recommendations whenever and wherever he is. Portable recommendations are available on small devices, such as palmtop computers. *Trust* refers to the issues that arise from the centralized nature of most recommenders. The user must share his opinions with the E-commerce website running the recommender, and then trust that site to use his data appropriately. Once the user has shared opinions with the site, he may find it difficult to prevent abuses of his trust. We will now examine these two issues in more detail.

1.1 Portability

Outside of some high-end retail stores where the clerk remembers your name and favorite color, personal recommendations have traditionally not been available in brick and mortar stores. That is starting to change. For instance, Safeway Inc., the United States' third largest grocery store, has just started a pilot test of "Magellan" [AP 2002], a shopping cart with a computer, a small LCD screen, a bluetooth chip, and a slot for consumers to swipe their Safeway club cards. As the consumer wheels Magellan around the store recommendations and special offers appear on the LCD screen. The special deals and recommendations are dependent on what products the consumer is near as determined by the bluetooth receiver in the cart, and the customer's past shopping history. Magellan is an example of a portable recommender, offering recommendations that consumers can carry with them to the place where they make their decisions about what to buy. Portable recommenders will be even more useful when they are available for a wide range of products, and when they are small enough that they can fit on a palmtop computer.

Imagine that sometime in the future, you are standing in Best Buy. On the spur of the moment you have decided to add a new DVD to your collection, but aren't sure which one. You pull your palmtop out of your pocket, and tap on the movie recommender application. You are thinking of something in the science fiction genre and as you look over the list of science fiction DVDs you own, you decide that this week you would like something along the lines of *Contact* or *The Matrix*. You select these two movies and then tap the recommendation button. Your palmtop then provides you with a list of five new DVDs for you to choose

from. You decide to purchase *Artificial Intelligence: AI*. So you put it in your cart, tap on the check box next to the title and it is added to your collection.

A few days later you are in your home office at your desktop computer. Having decided to go to the movie theater that night you bring up your desktop recommender client. Because you have synchronized your palmtop, the client asks you what you thought of *AI*. Of course you loved it, so you rate it a 5, and go on to investigate your in-theater movie recommendations for tonight. The movie recommended is *Solaris*, which is unfamiliar to you, but the recommender system tells you that it has a similar cast and the same director as a number of movies you have really liked, so you buy some tickets online and prepare to head out.

One of our goals is to develop portable recommenders that can realize scenarios like these.

1.2 Trust

In order to get value from a collaborative filtering system the user must entrust the system with a certain amount of personal information. The information needed may be anything from the purchases a user has made, to the pattern of websites a user has visited to explicit ratings of movies, books, or news articles.

Many sites collect explicit preference information. For instance, Amazon.com offers consumers the chance to rate items they have purchased to improve the quality of their recommendations. Many sites also use implicit preference information. E-commerce sites often make use of their shoppers' purchasing history to recommend new products to their customers. In addition, some systems try to make sense of shoppers' browsing behavior by analyzing the web log and looking at the pattern of pages each user visits [Mobasher et al. 2000]. Our early research in Usenet news showed that we could use the amount of time a user spent reading a news article as a surrogate for the user's rating [Miller et al. 2002; Morita and Shinoda 1994]. Other researchers have also looked into the problem of using implicit information in a collaborative filtering system to supplement or replace explicit preference information [Claypool et al. 2001; Terveen et al. 1997; Goecks and Shavlik 2002].

When a site collects personal information, the user must trust the site to protect the information appropriately. Foner [1999] identifies five ways that this trust might be misplaced. In **deception by the recipient**, a site could set out to deliberately trick users into revealing personal information, for identity theft [Whelan 2002] or other malicious purposes [Clymer 2003]. **Mission creep** refers to a situation in which an organization begins with a clearly defined use for personal data, but expands the original purpose over time. **Accidental disclosure** happens when a website accidentally makes private information available. For instance, the U.S. Air Force once inadvertently sold 'surplus' tapes that had secret data on them [Neumann 1995]. **Disclosure by malicious intent** happens when personal information that you entrust to an organization is stolen. Finally, information is sometimes released because of **subpoenas**. Even though an organization may take great care to protect personal information, they are obliged under the law to disclose that information when they are

subpoenaed. For example, Federal Express receives several thousand subpoenas each day for their shipping records [Foner 1999].

The issue of trust is especially important in e-commerce applications, which is where most applications of collaborative filtering today are found. The merchant wants to know his customer better in order to sell her the products that maximize profits. The customer, although willing to share some personal information, does not want that information to be used inappropriately. This tension is highlighted by the findings of Ackerman, Cranor and Reagle's 1999 survey of Internet users [Ackerman et al. 1999]. They found that the following four factors were key determinants in a user's decision to share data online:

- (1) Whether or not the site shares the information with another company or organization.
- (2) Whether the information is used in an identifiable way.
- (3) The kind of information collected.
- (4) The purpose for which the information is collected.

Most websites try to alleviate their customers' fears by posting a privacy policy on their site that describes how the merchant will use information provided by the customer. However, the recent economic downturn has caused some merchants to rethink their policies, or abandon them altogether. This was the case with the defunct Toysmart.com. When Toysmart ceased operations in May 2000, the company was forced to solicit bids for its assets, including its customer list and customer profiles. In addition, some of the largest websites, like Yahoo.com have altered their privacy policy to allow them to sell their customers' email addresses in order to add sources of revenue [Hansell 2002]. In another case, 800.com sold its web address and customer profile database to Circuit City. Customers of 800.com were told: "As a result of the transfer, any customer information transferred to Circuit City will be governed by Circuit City's privacy policy." Customers did not get the option to keep their profiles out of Circuit City's possession.

The key issue highlighted in these scenarios is that once merchants have control of users' personal information, the users can no longer control the uses to which that information is put. These are exactly the kind of control issues that the users in Ackerman, Cranor, and Reagle's study were concerned about [Ackerman et al. 1999]. User control of personal information is a key trust issue for recommenders.

A second key trust issue is the reliability of the recommendations. Very recently, the credibility of Amazon.com's recommendation services was called into question when it admitted to using 'faux recommendations' in order to drive business to its new clothing store partners [Wingfield and Pereira 2002]. The 'faux recommendations' were presented right next to the traditional recommendations for other products, but were not based on a customer's past shopping behavior or preferences. The 'recommendations' were merely links designed to drive the user to new parts of the Amazon store. This raises the general question: How can a user know what the recommender system on a site is doing to produce a recommendation? Almost all recommender systems manipulate

the recommendations in some ways. Many such manipulations are benign: a recommender might remove an item from the list if it is not in stock. Others are less benign: a recommender might reorder the recommendations so higher profit items are at the top of the list. How can users know whether to trust the recommendations that are made?

One possibility is that users might be able to trust recommendations from third-party sites that do not directly sell products more than recommendations from sites that do directly sell products. Three examples of such services are: Jungle, Jango, and Deja. Amazon purchased Jungle in 1998 with the promise that the Jungle technology would simply make Amazon the anchor tenant in the largest mall on earth. Currently, Amazon only provides the Jungle search capability within the family of Amazon associates. Likewise, Jango and Deja were purchased by third-party companies and no longer provide independent recommendations.

We have seen that users have two trust concerns with recommenders: first, that a centralized recommender might share personal data in inappropriate ways, and second, that a recommender owned by a commerce site might make recommendations that serve the site, not the user. Both concerns can be met by a *personal recommender*. The personal recommender holds the user's personal data locally, and only shares data the user explicitly identifies as sharable. The personal recommender is software owned by the user and running on the user's local machine. It answers to no one except the user. Our long-term vision is to develop such a personal recommender. This paper describes our first steps.

1.3 Research Goals

We have created a set of four goals for a personal recommender system:

Portability. Users should be able to receive recommendations wherever they are, on whatever client they are using, even when the client is disconnected from the Internet.

Palmtop. Users should be able to run the recommender on palmtop size machines. Specifically, we are interested in allowing users to maintain their profiles on a palmtop such as a Palm Pilot or PocketPC. To run on a palmtop, recommender algorithms must be able to run efficiently on architectures with slow processors and limited memory. We benchmarked several different processors, and studied the effectiveness of different algorithms at a range of memory sizes, from 1 megabyte to 256 megabytes. Though PDAs in the future will have more than 256 megabytes of memory, we did not study larger memory sizes because our preliminary studies showed that adding memory beyond 256 megabytes yielded no additional benefit in recommendation quality.

Studies of web usability have shown that users have sensitivity to latency. For instance, users in one study found latency of 5–10 seconds to be about “average” [Bouch et al. 2002]. Above 8 seconds, satisfaction with the performance of the website dropped significantly. We analyze the processor performance required to deliver accurate recommendations within 8 seconds for each algorithm, and at each model size.

The *Palmtop* and *Portability* goals go hand in hand. Ideally we would like to see a recommender system that runs solely on a palmtop; however we will see that we can achieve additional flexibility for a solution by allowing desktop hardware to perform much of the online computation and using the palmtop as an offline device.

User Control. Users should control their profile and ratings information. We define control as the ability for a user to choose which of their ratings they would like to share with the community, and perhaps even which individuals they are willing to share their ratings with. We have investigated a range of available algorithms, with a corresponding range of options for user control, from choosing to share their profile anonymously, to sharing a version of their profile that can be used but not read by other community members.

User control involves twin challenges: develop recommender algorithms that provide control choices, and develop interfaces that help users exercise those choices in a live recommender system. The present article focuses entirely on the former challenge. Developing suitable interfaces will be crucial, once algorithms are known that make choices available.

Accuracy. Our goal is to make portable recommendations that are highly accurate. Concretely, the accuracy goal is to provide recommendations that are as good as those provided by the best known algorithms.

1.4 Contributions

The contributions of this paper are as follows:

- We introduce the PocketLens collaborative filtering algorithm. PocketLens is a recommender algorithm that is designed to run in a peer-to-peer environment. Specifically, PocketLens is designed to run on client devices as small as palmtop computers, and to enable users to choose to only share some of their ratings with other users.
- We provide a comparison of five architectures for distributing ratings among recommender clients. Each architecture provides a way for PocketLens clients to discover a set of ratings they can use to make accurate recommendations.
- We evaluate the architectures in two ways. First, we quantitatively evaluate their ability to meet the accuracy and performance goals. Second, we qualitatively evaluate the tradeoffs the architectures offer among the information security guarantees they make, implementation complexity, and performance.

1.5 Roadmap

The rest of this article proceeds as follows. In Section 2 we review several different algorithms currently in use for collaborative filtering. We also review several different kinds of peer-to-peer and intelligent agent systems to place peer-to-peer collaborative filtering in context.

In Section 3 we provide an overview of the PocketLens algorithm. Everyone should read Section 3.1 for a summary of the benefits of the algorithm.

Readers interested in the details of the algorithm should also read Section 3.2. In Section 4 we describe five different peer-to-peer architectures that incorporate the PocketLens algorithm. In Section 5 we compare the performance of the PocketLens algorithm on each of the different architectures. We summarize the tradeoffs in scalability, privacy, and security of each architecture in Section 6. Finally we conclude and discuss some future directions for research in Section 7.

2. RELATED WORK

Most people are intimately familiar with the most basic form of collaborative filtering: “word of mouth.” After all, it is a form of collaborative information filtering when someone consults with their gastronomically inclined friends to gather opinions about a new restaurant before reserving a table. Or, when someone consults with their literary colleagues before plunking down twenty five dollars for a new hardcover. Or finally when someone pays attention to the opinions of their musically gifted friends before ordering a new compact disc.

In the information overload age collaborative filtering is taken to the next level by allowing computers to help each of us be filters for someone else, even for people that we do not know. *Automated collaborative filtering* requires that each of us be willing to provide the collaborative filtering system with our expression of value for a particular information item. The expression of value by a user for an item is called a rating. The computer’s role is to predict the rating a user would give for an item that he or she has not yet seen. The first step in calculating a prediction is to find a group of individuals who have rated information items similarly. This group of individuals is called a neighborhood. Once a neighborhood is formed a prediction is calculated for an item by looking at how the user’s neighbors have rated that item. The intuition behind collaborative filtering is that if the user has agreed with their neighbors in the past, they will continue to do so for future items.

Collaborative filtering allows both humans and computers to do what they do best. Humans are good at reading and reacting to information items. Humans can make judgments about the value of information items. Humans can evaluate the quality of information. Computers excel at crunching data. Computers can store and retrieve data quickly. Computers can find patterns in data. Collaborative filtering combines the qualitative judgments of humans with the data crunching capabilities of computers to make predictions about things humans will value.

In this section we will look at the evolution of collaborative filtering algorithms, with a special emphasis on the item-item algorithm that we build on throughout the rest of the article. We close the section with an outline of some types of peer-to-peer systems, and with a brief overview of intelligent agent systems.

2.1 User-Item Collaborative Filtering

The basic user-item collaborative filtering algorithm described in Resnick et al. [1994], can be divided into roughly three main phases: neighborhood formation, pairwise prediction, and prediction aggregation. In Figure 1 the

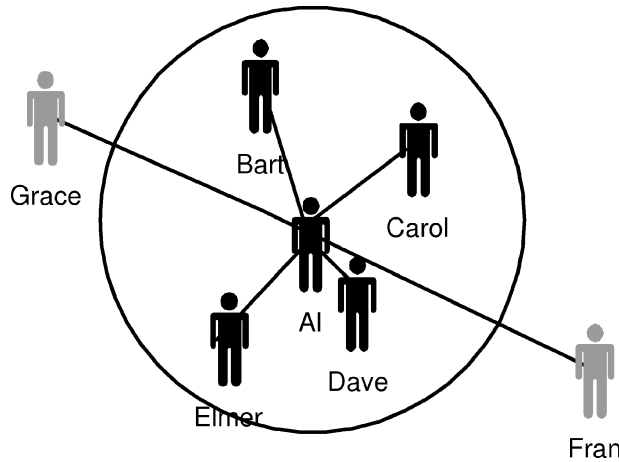


Fig. 1. Basic collaborative filtering algorithm.

Table I. An Example User-Item Matrix

	X-men	Star Wars	Clockstoppers	Minority Report	Two Towers
Len	5	4	?	4	4
Jane	1	2	3	1	1
Josh	5	4	2	?	5
Betty	2	?	3	2	1

shadows represent users. In particular we are interested in calculating predictions for the user Al. The distance between each shadow indicates how similar each user is to Al. The closer the shadows the more similar the users. In neighborhood formation the trick is to select the right subset of users who are most similar to Al. Once the algorithm has selected a neighborhood, represented in Figure 1 by the users in the circle, it can make an estimate of how much Al will value a particular item. After the algorithm has learned how much each user in the neighborhood rated a particular item, the final step is to do a weighted average of all the ratings to come up with a final prediction.

Internally, we represent the collaborative filtering problem as predicting missing values for cells in a matrix like the one shown in Table I.

Suppose we are trying to predict how much Josh will like *Minority Report*. We start by finding a neighborhood for Josh. Just by looking at the matrix you can see that Josh and Len tend to agree strongly on the movies they have both seen. There are several ways to formalize this idea of agreement, for example as cosine similarity, or Pearson correlation [Devore 1995]. In the user-item algorithm we use Pearson. The formula for calculating a Pearson correlation coefficient is shown in Equation 1. $w_{a,u}$ represents the similarity between the active user a and a potential neighbor u . $r_{a,i}$ represents the rating for one item i of the active user, and \bar{r}_u is the average of all of user u 's ratings. σ_a is the standard deviation of the active user's ratings. σ_u is the standard deviation of the potential neighbor's ratings.

$$w_{a,u} = \frac{\sum_{i=1}^m (r_{a,i} - \bar{r}_a) * (r_{u,i} - \bar{r}_u)}{\sigma_a * \sigma_u} \quad (1)$$

Once we have selected a set of neighbors for Josh, we can Calculate the prediction p for the *Minority Report* according to Equation 2.

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u=1}^n (r_{u,i} - \bar{r}_u) * w_{a,u}}{\sum_{u=1}^n w_{a,u}} \quad (2)$$

Breese et al. [1998] and Herlocker et al. [1999] have performed extensive empirical analyses of neighborhood based collaborative filtering algorithms. Herlocker investigated two variations on the basic algorithm we have just described. Herlocker tried two different methods of normalizing the rows of the matrix:

- Normalize each row vector to be of unit length.
- Normalize each row vector by subtracting the row average from each rating. We call this the zscore method.

Herlocker found that the zscore method worked the best and dramatically improved the performance of the basic collaborative filtering algorithm.

2.2 Model Based Collaborative Filtering

The nearest neighbor algorithm described in the previous section has two main limitations:

Scalability. As the number of users and items grows, the process of finding neighbors becomes very time consuming. In fact the computation is approximately linear with the number of users. This is especially problematic for large, high volume websites that want to do a lot of personalization.

Sparsity. Another problem with the nearest neighbor problems is that as the number of items grows, users rate a smaller percentage of the item population. Nearest neighbor algorithms require that users have at least two items they have both rated in order to correlate them. In a large data set many users may have no correlation at all. Finally, the sparsity problem also means that accuracy may suffer because predictions for items must be based on only a few ratings.

Many researchers have looked at the problem of increasing the density of the matrix using a variety of techniques including:

- **Filterbots:** Filterbots use simple content analysis to produce automatic ratings of every item in the dataset [Sarwar et al. 1998; Good et al. 1999]. Filterbots directly reduce sparsity, and produce subsets of the ratings matrix with very high density, which helps in correlation.
- **Clustering users:** Some researchers have looked at the model based problem as finding clusters of users and then using the nearest neighbor algorithm on the clusters [Ungar and Foster 1998; Mobasher et al. 2002].
- **Classifiers:** Other researchers have recast collaborative filtering as a classification problem in which a neural network or other machine learning algorithm classifies each item according to how it believes a user will rate the item.

	beer	diapers	shampoo	conditioner	milk	cookies
beer		10	1	2	5	8
diapers	10		3	2	4	6
shampoo	1	3		9	1	2
conditioner	2	2	9		2	3
milk	5	4	1	2		15
cookies	8	6	2	3	15	
Total	1	13	1	11	6	10

Fig. 2. Counts of copurchased items form a very simple item-item model. The counts can be used to produce recommendations.

- **Singular Value Decomposition (SVD):** SVD is a technique that leads to a clustering of similar items in a matrix called Singular Value Decomposition. The SVD is a compact representation of the entire user-item matrix with estimates for the missing cell values.
- **Item-Item:** Item-Item is a technique for capturing the similarity relationships between pairs of items. We will look at the details of the item-item algorithm in Section 2.2.1

All of the above mentioned approaches have one thing in common. Each approach separates the collaborative filtering computation into two parts. In the first part, which can be done offline, we build a *model* that captures the relationships between users and items. The second part of the computation uses the model to compute a recommendation. If we are clever we can do most of the work in building the model and make the recommendation computation very fast. In this section we will look at two methods for model-based collaborative filtering that help alleviate the sparsity and scalability problems without using the content of the items.

2.2.1 Item-Item. The model-based approach that we will use throughout this paper is based on the observation that the purchase of one item will often lead to the purchase of another item [Aggarwal et al. 1999; Billsus and Paz-zani 1998; Breese et al. 1998]. To capture this phenomenon we build a model that captures the relationships between items. We call this approach *item-item* [Karypis 2001; Sarwar et al. 2001].

A very simple item-item model can be built by simply counting the number of times that a pair of products is purchased by the same user. Figure 2 illustrates the model resulting from a small set of products.

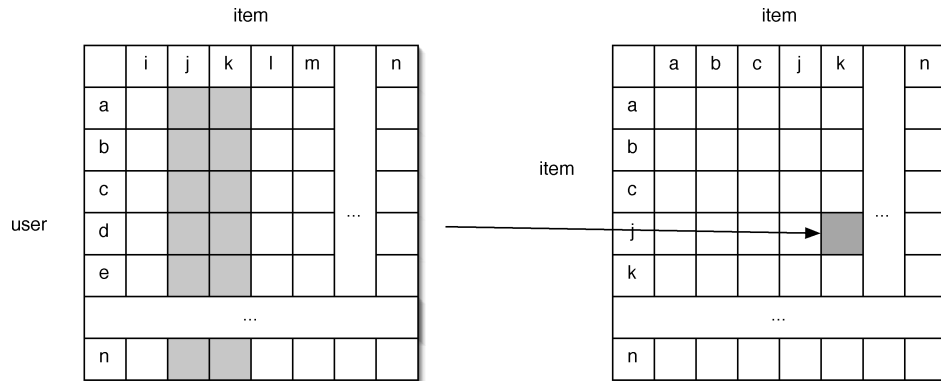


Fig. 3. Calculating a similarity score for a pair of items from a user-item matrix.

Notice that the model represents some common sense associations like milk and cookies, or shampoo and conditioner. We can use this simple model to make recommendations for a user about other products she might be interested in buying. For example, if a user has already purchased cookies and shampoo we can find out what other items are most likely to go with those two items by adding up the columns for those two rows, giving us the vector shown in Figure 2.

If we sort the vector we can generate a top n list of recommendations for our user and recommend the N items with the largest values. In this case a top 3 list would consist of diapers, conditioner, and cookies. In practice this simple model works quite well for recommending items. There are two disadvantages to this simple model. The first is that it tends to recommend the most popular items. The second is that the model cannot be used to predict what the user would have rated an item. So, it is good for predicting consumption but not good for predicting a user’s perception of quality. Even with these disadvantages this type of model has been used successfully in e-commerce applications.

We can however; make a small refinement to the model building process to enhance the model sufficiently to allow us to make quality predictions. The enhancement is to store a similarity score in each cell rather than the co-purchase count. The method we use to calculate the similarity score is vector *cosine similarity*. Cosine similarity for two vectors is defined in Equation 3.

$$\cos \vec{u}, \vec{w} = \frac{\vec{u} \cdot \vec{w}}{\|\vec{u}\| \|\vec{w}\|} \tag{3}$$

The vectors we will use for our calculation are the column vectors from a user-item matrix as shown in Figure 3.

We can fill in an entire item-item matrix by repeating the similarity calculation for each pair of items. Now, we can still calculate a recommendation list for a user using the procedure we described for co-purchase matrix. In addition, if the user asks for a prediction for a particular item we can directly compute it

by using the following formula:

$$P_{u,i} = \frac{\sum_{i,N} (S_{i,N} * R_{u,N})}{\sum_{i,N} (|S_{i,N}|)} \quad (4)$$

Equation 4 simply states that we can calculate a prediction for an item $P_{u,i}$ by calculating a weighted average of the user's ratings $R_{u,N}$ using the similarity score $S_{i,N}$ as the weight. The idea is that the average rating of items that are similar to the selected item is a good estimate of the rating for the selected item. We weight the ratings for the other items by their distance from the selected item.

Sarwar [Sarwar et al. 2001] found that the item-item algorithm can be improved by using the same normalization techniques on the row vectors of the matrix as reported by Herlocker [Herlocker et al. 1999].

The item-item algorithm has a number of advantages. It also provides a solution to the sparsity problem. Item-item allows us to create a model that a relatively new user can make use of right away, and it relieves us from the computation of neighborhoods at run time. As we will see in Section 3, the item-item model lends itself to an incremental approach that is critical for distributing the algorithm.

2.3 Peer-to-Peer

Peer-to-peer computing systems have been around since networked computing was invented. However, the success of Napster made the term ubiquitous. Although much of what people think of as peer-to-peer networking today is oriented towards file sharing, peer-to-peer computing includes a wide range of applications. A working definition that has proved useful in identifying the new generation of peer-to-peer systems is as follows:

peer-to-peer nodes must operate (mostly) outside the Domain Name Service (DNS) and have significant or total autonomy from central servers [Shirkey 2001].

In addition, a peer-to-peer network allows for a significant amount of resource sharing among the clients on the network. What this means is that we can harness the storage, CPU power, and bandwidth of computers that are only occasionally connected to the network, and have no persistent IP address. Some examples of peer-to-peer systems that fit this definition include: Gnutella, Napster, ICQ, United Devices, and SETI@home.

In this section we will look at examples of peer-to-peer systems in several application categories. Table II gives an overview some of the categories where there are peer-to-peer applications.

Large Scale Computation. SETI@home is a good example of a peer-to-peer application that utilizes the computing power of millions of personal computers [Anderson et al. 2002]. The goal of the project is to analyze radio signals from the Aricebo radio observatory in search of messages from extraterrestrial beings. The amount of data to be processed is beyond the abilities of today's central supercomputers. Therefore, the processing is distributed to individual

Table II. Peer-to-Peer Application Examples

Category	Example Applications
Large Scale Computation	SETI@home, United Devices, Folding@home
One-to-One Communication	AIM, ICQ, MSN, Yahoo Messenger
File Sharing	Gnutella, Kazaa, Freenet, Oceanstore, Limewire
Content Distribution	Akamai, Tangler, Publius
Collaborative Filtering	Yenta, PocketLens

clients running on servers and desktop computers. As of August 2002, more than 3.91 million people had downloaded the SETI@home client. Each client is responsible for processing one work unit at a time, and then reporting the results of the processing back to the central control system. After one work unit is completed another work unit is assigned. Overall, the joint computations performed by SETI@home clients amount to 1.873×10^{21} floating point operations! This is the largest computation on record [Anderson et al. 2002]. SETI@home fits our definition of a peer-to-peer system because each client has a high degree of autonomy over when and how much processing it will do for a given work unit.

Since the success of SETI@home several other organizations have started their own peer-to-peer computational projects. Some examples are: United Devices (www.uniteddevices.com), distributed.net, and Folding@home. Folding@home focuses on the protein folding problem, which involves predicting the three dimensional structures of amino acid sequences. Understanding the protein folding problem is beneficial to the fields of biomedicine and nanotechnology. [Distributed.net](http://distributed.net) is trying to win the RSA Data Security challenge by decrypting a secret message encoded using the RC5 encryption algorithm with a 72 bit key. The method they are using is simply the brute force technique of trying every possible key. The effort currently has enough participants to process 127.2 billion keys per second (www.distributed.net). Clearly, a peer-to-peer network can address some very large computational problems.

One-to-one communication. Another popular set of peer-to-peer applications is the Instant Messaging (IM) applications. Examples of IM clients include AOL Instant Messenger (AIM), ICQ, Yahoo Messenger, and Microsoft Network. Each of these systems have a similar structure. Each person runs a client on their desktop machine or PDA. The client communicates with a central server to maintain its status, and to monitor the status of your “buddies” (people with whom you may want to communicate while online). If you choose to send a message to one of your buddies the communication takes place directly between your computer and your buddy’s. IM clients fit our definition of peer-to-peer computing because the communication that takes place is directly between two clients, even though there is a strong reliance on a central server to maintain the directory of active clients.

File Sharing. There are several active file sharing networks including Gnutella [Gnutella.wego.com], OceanStore [Kubiatowicz et al. 2000], and Freenet [Clarke et al. 2002]. There are also a number of research prototype networks such as Chord, CAN, CFS, and Pastry [Stoica et al. 2001; Ratnasamy et al. 2001; Rowstron and Druschel 2001a, 2001b; Dabek et al. 2001]. All of

these systems are organized around allowing a user to share files with other users. The specific protocols differ but each system provides the following general functionality:

- Join the network: usually by contacting a well known host to learn about at least one peer.
- Search the network for a file using some key.
- Transfer a file: Often the file transfer protocol uses HTTP rather than specifying file transfer as part of the protocol.
- Leave the network.

All of the file sharing networks fit our definition of peer-to-peer computing because they share a significant amount of storage, and because once a client is bootstrapped into the network, the communication does not require DNS services and the communication takes place directly between clients on the network.

The differences in the various file sharing peer-to-peer networks center around the efficiency of the network with respect to finding resources. We will discuss these differences in more detail in Section 4 where we discuss how we extend the ideas in these file sharing networks to achieve our goal of a peer-to-peer recommender system.

Content Distribution. Another application of peer-to-peer networking is for content distribution. One of the earliest peer-to-peer networks was Usenet. More recently researchers have been looking at replacing centralized web servers with a network of content providers. There are two main motivations for peer-to-peer content distribution: moving cached content closer to the end users, and preventing censorship. The most commercially famous of these system is Akamai [Karger et al. 1997], which focuses on caching content, on behalf of centralized websites, to reduce network bandwidth utilization.

There are several content distribution research prototypes that add additional layers of security and tamper detection such as Publius [Waldman et al. 2000], Tangler [Waldman and Mazi 2001], Free Haven [Dingledine et al. 2001], Intermemory [Chen et al. 1999], and Mojonation (www.mojonation.net). As an example of how one of these systems works we will look at Tangler. Tangler has two key components, a publishing component that transforms a document into a disjoint set of blocks, and a reconstruction component that reassembles the blocks into the original document. The publishing component breaks up the original document and combines, or *tangles*, the blocks of the original with blocks of other published documents. The individual blocks are encrypted, signed with a hash key, and stored on a network of computers. The reconstruction program must find the blocks, check their hash signatures, and then put back together the original document. The idea is that it would be extremely difficult for someone to change or remove a document published in a Tangler network.

Collaborative Filtering. Although most collaborative filtering algorithms are centralized, there has been some research on distributed collaborative

filtering algorithms [Sarwar et al. 2001] and moving collaborative filtering toward a peer-to-peer approach.

Canny [2002a, 2002c] builds on three key elements of secure distributed computing: ElGamal encryption [Pedersen 1991], homomorphic encryption, and Zero Knowledge Proofs [Cramer et al. 1997] to provide a peer-to-peer collaborative filtering system with strong privacy guarantees. Canny's approach uses a mechanism for building an SVD model based only on vector addition. ElGamal encryption, combined with the homomorphic mathematics allows vectors to be summed by multiplying the encrypted addends, and then decrypting the final result. Individual addends can be verified as valid data by using zero knowledge proofs. In Section 4.5 we show how these mechanisms can also be used to build a model using the PocketLens algorithm. The PocketLens algorithm has two advantages over SVD in this application. First, the PocketLens algorithm would only require a single iteration of the security protocol to build a complete model. Second, the security protocol can be used to incrementally update the item-item model without starting over from scratch.

2.4 Intelligent Agents

Another area of research that has much in common with peer-to-peer applications is the mobile and intelligent agents community. Much work has been done in creating groups of agents who work together to solve a common problem [Chavez and Maes 1996; Minar 1998; Minar et al. 1999]. Some early work in using agents for information filtering includes Amalthea and other projects from the MIT media lab [Moukas and Zacharia 1997; Maes 1994; Sheth and Maes 1993; Maes and Kozierok 1993].

The first multi-agent system for collaborative filtering systems was Yenta [Foner 1996]. Yenta is a multi-agent matchmaking system that learns about a user's interests and matches the user with other people who have similar interests. Yenta learns about people by finding sets of keywords that describe the various interests of a user. A Yenta client can then compare these sets of keywords with other clients, without giving away the identity of the user, to determine if one user is a good match for another. If two users are a match, the clients can then discretely negotiate to see if the users would like to reveal their identities to each other. Yenta blazed the trail by showing that collaborative filtering could be done in a distributed environment. Our work extends the Yenta system by characterizing users by their ratings, and then using the ratings to make recommendations about movies for users.

3. POCKETLENS

3.1 Benefits of PocketLens

In this section we present a peer-to-peer collaborative filtering algorithm, called PocketLens, that is suitable for use in a personal, recommender system. In Section 1.3 we introduced four goals for our personal recommender system. The four goals are *portability*, *palmtop compatibility*, *user control*, and *accuracy*.

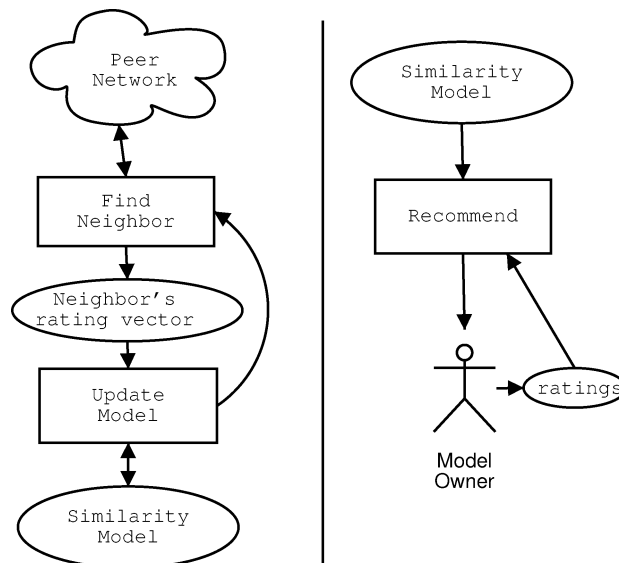


Fig. 4. Overview of the PocketLens architecture for building a similarity model and making recommendations.

The PocketLens algorithm, along with the architectures we will discuss in the next section must support these goals. The goal of *portability* is met by creating a model while the user is online, that can be used to make recommendations while the user is offline. In addition, because the computation of the model is distributed across many processors, and the resulting model is small, the PocketLens algorithm supports the *palmtop compatibility* goal. The PocketLens algorithm supports the goal of *user control* by allowing the user to choose which part of her profile she wants to share with other members of the peer-to-peer community. In some cases users may even choose to share none of their ratings and simply create a model from the ratings shared by others. Finally, we experimentally evaluate the extent to which PocketLens supports the *accuracy* goal.

3.2 Algorithm

Recommender algorithms can be characterized by the neighbors they choose for each user, the model they build based on those neighbors, and the way they use the model to form recommendations. The key to peer-to-peer collaborative filtering lies in separating the incremental construction of the similarity model from later use of that model to make recommendations.

Figure 4 illustrates the processing required to create a model and produce a recommendation list. The first step is to create a similarity model. The **Find Neighbor** module searches the peer-to-peer network to find neighbors. When a neighbor is found, the ratings from that neighbor are passed to the **Update Model** module, which incorporates the ratings from that neighbor into the similarity model. Once the model is updated, the current neighbor's ratings are discarded.

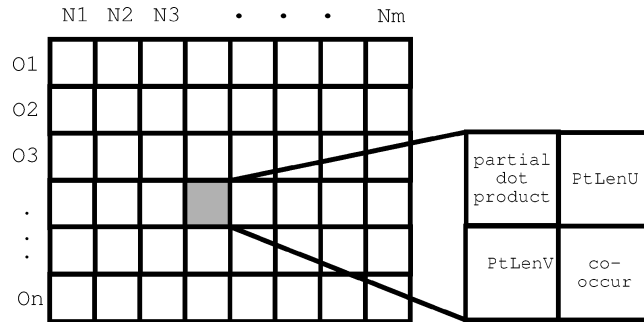


Fig. 5. Constructing the partial item–item matrix. Each cell in the matrix comprises the similarity between item O_i and N_j .

To make recommendations, the **Recommend** module examines the similarity model to find the n items that are the most similar to some or all of those that the model owner has already rated. This list of n items is then presented to the user through a user interface.

The similarity model at the heart of the PocketLens algorithm comprises a set of similarity relationships between pairs of items. Each similarity relationship is computed based on the way users have rated the items. We examined several ways to compute the similarity, including correlation, cosine and conditional probability. The cosine measure was the most accurate in preliminary experiments, which is consistent with past research [Karypis 2001; Herlocker et al. 1999], so the PocketLens algorithm is based on cosine similarity. Consider two items \vec{u} and \vec{w} . We represent these items as a vector of ratings in user space, where u_i is the rating given to item \vec{u} by user i . We can define the cosine similarity between two items as in:

$$sim(u, w) = \cos \vec{u}, \vec{w} = \frac{\vec{u} \cdot \vec{w}}{\|\vec{u}\| \|\vec{w}\|}$$

The similarity model can be represented as a matrix, M , in which rows and columns are both items. An x, y entry represents the similarity between the item on row x with the item on column y . Sarwar and Karypis describe how to compute an $n \times n$ item–item similarity matrix given an $n \times m$ user–item rating matrix in Sarwar et al. [2000b] and Karypis [2001]. We have designed the PocketLens algorithm so that we only need access to the ratings of the model owner and one other user at a time.

The PocketLens algorithm is designed so that we limit the rows of M to the set of items, O , that the model owner has rated. Since the similarity model will only be used by the owner, the rows for items the owner has not rated are not needed. The columns of the matrix are the items that other users have rated; we call this set of items N . N is also the set of items that are available to be recommended to the model owner. Storing a limited set of rows and columns keeps the model size relatively small. In addition, because M is quite sparse in practice [Sarwar et al. 2000a] we use sparse matrix storage techniques to further reduce the model size. Figure 5 illustrates the contents of the model for a user.

For a particular neighbor we define the set of items rated by that neighbor as C , and the vector of ratings as \vec{c} . The first thing we do is normalize the ratings so that $\|c\| = 1$: all neighbors have approximately equal influence on the model. Otherwise, neighbors that have rated a large number of items would have more influence than neighbors who have only rated a few items [Herlocker et al. 1999].

In Section 2.2.1 we computed the similarity score by simply applying Equation 5 directly to the column vectors of the full matrix. In the PocketLens algorithm we must incrementally compute the dot product and vector lengths of the column vectors (PtLenU, and PtLenW). For each neighbor, we want to update some of the cells in M . The items we are interested in updating for a particular neighbor are in the set C , such that $C \cap O$ is non-empty. For each cell, (O_i, N_j) , in M where $O_i \in O \cap C$ and $N_j \in C - O$ we need to update the four values shown in Figure 5. Let v_k be our neighbor's rating for item O_i and let u_k be our neighbor's rating for item N_j . We update the quantities as follows:

$$\begin{aligned} \text{PartialDot}(O_i, N_j) &= \text{PartialDot}(O_i, N_j) + u_k w_k \\ \text{PtLenU}(O_i, N_j) &= \text{PtLenU}(O_i, N_j) + u_k u_k \\ \text{PtLenW}(O_i, N_j) &= \text{PtLenW}(O_i, N_j) + w_k w_k \\ \text{Cooccur}(O_i, N_j) &= \text{Cooccur}(O_i, N_j) + 1 \end{aligned}$$

At any point in the process of constructing the similarity model, we can compute a recommendation list by computing the similarity scores for each item as shown:

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

k is defined by Herlocker as the significance weighting factor [Herlocker et al. 1999] and serves to devalue neighbors with whom we agree strongly, but only have a few items in common.

$$k = \begin{cases} 1 & \text{Cooccur}(O_i, N_j) \geq 50 \\ \text{Cooccur}(O_i, N_j)/50 & \text{otherwise} \end{cases}$$

Once we have completed the calculations for each cell we sum the similarity scores for each column N_j . The items N_j that have the highest total score are the items to recommend. This list of items becomes the recommendation list to be presented to the user in a recommendation application. If, instead, the user is asking for a prediction for a particular item or set of items, we can directly compute the predicted rating for an arbitrary item N_j by calculating the weighted average of the similarity scores in the column. The weight factors are simply the model owner's ratings for the rows O_i that intersect the column.

In summary, the PocketLens algorithm is a variant of the accurate and efficient item-item algorithm [Karypis 2001; Sarwar et al. 2001], with three key features introduced for the peer-to-peer environment. First, we have modified the algorithm to construct the matrix incrementally. Second, the PocketLens model is small, since it is only for one user, which allows us to use the algorithm

on desktop computers and palmtops. Third, to preserve privacy, the PocketLens algorithm has the property that none of ratings in \vec{u} or \vec{w} are saved in conjunction with anything that would identify the user they came from.

4. POCKETLENS ARCHITECTURES

The PocketLens algorithm describes how to construct an item-item model given some way of finding neighbors. The method used to find neighbors has an impact on the accuracy of the results, as well as the privacy afforded to the participants. In this section we examine five different architectures for locating neighbors for the model. In Section 6 we will see that the different architectures lead to different tradeoffs among our goals for peer-to-peer recommender systems.

The five reference architectures range from centralized to distributed, and from less secure to more secure. Figures 6(a) – 6(e) show the five architectures and illustrate their connectivity and processing. Each of our reference architectures uses the PocketLens algorithm.

4.1 Central Server

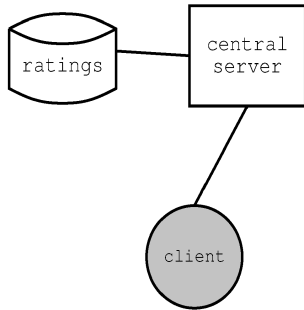
The central server architecture, shown in Figure 6(a), is based around a trusted central ratings storage server. In this architecture each user has a unique persistent identifier, which is used to associate her ratings with items stored in the central database. This system is similar in flavor to the SETI@Home project where the key data is stored on a central server, but the computing is done by individual nodes. In this case, ratings are stored at the central server, but each client builds its own model and computes its own recommendations.

The central server can help the client optimize the neighborhood by creating an ordered list of the most similar neighbors for every user in the system. As the PocketLens algorithm creates the similarity model the **find neighbor** module will query the server for the ratings of one neighbor at a time. In this way we distribute the computational load for creating a similarity model for each user, but retain the benefits of centralized collaborative filtering in that we are able to use the optimal set of neighbors. Because only the best neighbors are used, the accuracy of the recommendations in central server are as good as traditional collaborative filtering.

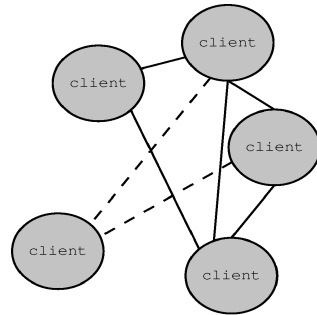
4.2 Random Discovery

The Random Discovery architecture, shown in Figure 6(b), allows each user to remain anonymous and uses the underlying protocol of Gnutella (www.gnutella.com) to find neighbors. The key element of the protocol used is the ping/pong mechanism for locating other nodes on the network. Figure 7 illustrates a small part of the Gnutella network, and shows how the ping and pong protocol elements work.

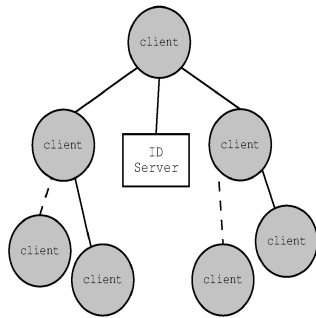
In Figure 7, node j has just joined the network. Node j learned about the existence of two additional hosts, d and e from the bootstrap server. Node j now issues a ping request to both d and e . When they receive the ping requests, both d and e respond to j with the neighbors that they know about. In this example, d responds that it knows about nodes b and e . Node e responds that



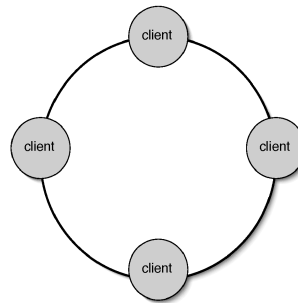
(a) Central Server—storage central, computation distributed.



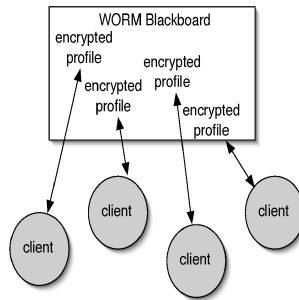
(b) Random—storage and computation distributed.



(c) Transitive Traversal—storage and computation distributed.



(d) Content Addressable—storage and ratings distributed, community model.



(e) Secure Blackboard—encrypted partial results written to WORM blackboard, community model.

Fig. 6. Five architectures for collaborative filtering. Computation is done in the shaded figures. Dashed lines indicate some clients may be off-line.

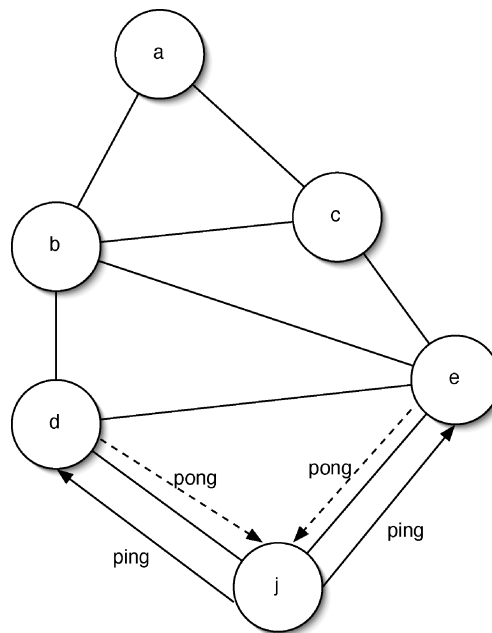


Fig. 7. The Gnutella PING / PONG protocol.

it knows about nodes d , b , and c . Thus, after one round of the protocol, node j now knows about four additional nodes.

The PocketLens algorithm can use this ping/pong mechanism for finding neighbors to create a model using only the neighbors that happen to be available on the network. We call this the random discovery architecture because the topology of the client connections has been shown to be similar to a random graph [Ripeanu et al. 2002; Jovanovic 2001], and the **find neighbor** module incorporates neighbors into the model without evaluating them first.

Jovanovic [2001] shows that a network constructed as we have described exhibits the property that every node in the network is only an average of 3.5 links away from any other node, and the graph is highly clustered. For example, Ripeanu et al. show that 95% of the nodes connected at any one time were in a single cluster. For purposes of neighborhood construction this means that any user currently connected to the network is as likely to be available to share ratings as any other.

To build a new model, a client continues to expand the set of neighbors by using the PING/PONG protocol and uses the HTTP GET mechanism, typical in the Gnutella network, to request a set of ratings from each new client. In this way, one client can quickly incorporate ratings from many other clients into its own model.

4.3 Transitive Traversal

The transitive traversal reference architecture, shown in Figure 6(c), improves upon the random discovery architecture by incrementally learning who the

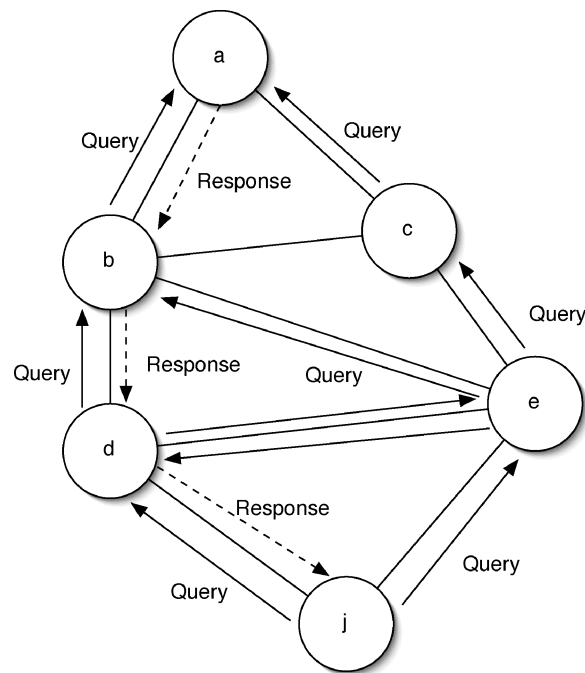


Fig. 8. The gnutella query protocol.

most similar neighbors are each time it encounters a new user. In order to remember who the best neighbors are we must re-introduce a persistent identifier (pseudonym) for each set of ratings. The identifier could be anything as long as it is unique and is used consistently for a user's set of ratings.

PocketLens can be smarter about how it enlarges the neighborhood by allowing clients to share their neighborhood lists and thus build neighborhoods using a form of transitivity. The intuition behind this approach is that if you are one of my best neighbors, then it is likely that your best neighbors will be good neighbors for me as well. The transitive traversal architecture maintains a queue of “neighbors' neighbors” as it scans potential neighbors. If some of the best neighbors are offline while building the model, the queue provides a pool of quality neighbors as substitutes.

We can build transitivity on top of the Gnutella network by employing Gnutella's search protocol. In Figure 8 we show an example of a query in which node *j* starts a search for the content on node *a*. The search begins with node *j* sending a query to each of its neighbors *d* and *e*. Since neither *d* nor *e* can satisfy the query request both nodes pass on the query to all of their neighbors, *b, c, d* and *e*. This method of passing on the query is called query flooding. Since *d* and *e* have already seen the query they do not propagate it again, but both *b* and *c* would propagate the query to *a*. Since *a* contains the file of ratings node *j* was looking for it sends a response back to node *b*. Node *b* propagates the response to node *d*, which propagates the response to *j*. Once again, *j* can use the HTTP get protocol to get the ratings file from *a*.

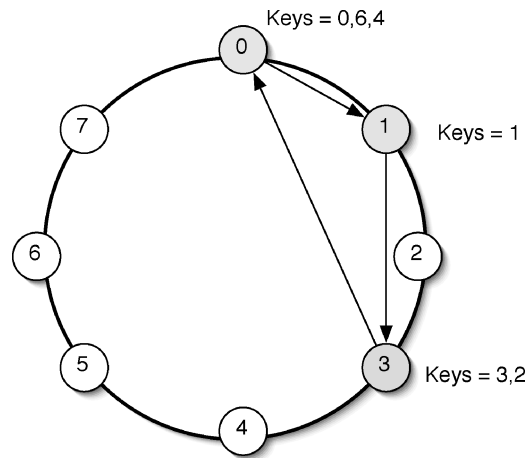


Fig. 9. Chord lookup.

4.4 Content Addressable

Our next architecture is based on recent advances in architectures for peer-to-peer file sharing networks such as Chord, CAN, and Pastry [Stoica et al. 2001; Ratnasamy et al. 2001; Rowstron and Druschel 2001a; Druschel and Rowstron 2001]. These systems have advantages over Gnutella because they impose a deterministic overlay routing system on the network. For example, CAN maps each node into the space defined by a d -torus, and Chord maps each node onto a ring. The new routing algorithm allows the network to make guarantees about bandwidth utilization, search times, and the availability of data. We will use the Chord architecture as the basis for our discussion, although any of the systems cited could serve as the storage layer for parts of the model and would lead to the same final design.

The basis for Chord is to provide scalable, distributed *lookup* operation. Given a *key*, *lookup* will return a *value* for that key in $O(\log(n))$ time. For example, in a peer-to-peer file sharing network, the key would be the file name, and the value would be a URL representing the location of the file, or perhaps even the file itself.

In order to understand how Chord can provide an $O(\log(n))$ lookup function in a peer-to-peer network we need to look at the architecture. A Chord network is organized as a virtual ring, as shown in Figure 6(d). Nodes are mapped onto the virtual ring using a persistent hash function [Karger et al. 1997]. Nodes on the ring are numbered from 0 to n , where n is chosen to be large enough to avoid collisions when the hash function, modulo n is applied to the node identifier. Similarly, *keys* are mapped onto the ring using the same hash function. A node on the network is responsible for all the keys equal to or less than its own position, but greater than its predecessor on the network. In addition, every node on the network has a pointer to its successor on the ring.

Figure 9 illustrates how a *key* is located on the Chord network. Suppose that node 0 wants to locate key 2. Since node 0 doesn't have key 2, it passes on the

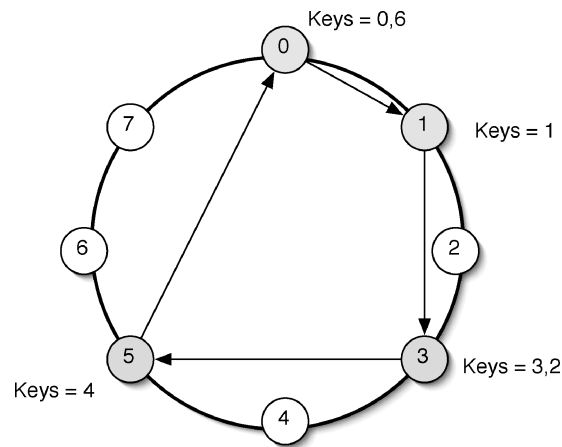


Fig. 10. Chord join.

request to its successor, node 1. Node 1 does not have the key so it passes on the request to its successor node 3. Since the 2 is less than three and greater than 1, 3 has the key. The location of the key is passed back to node 0.

As described, the lookup function could easily be $O(n)$ for a key that was far away from a node on the ring. So, the designers of Chord added a ‘finger table’ to each node. The finger table for each node stores a set of key intervals and node IDs for nodes at a repeatedly doubling distance around the ring. In this way each node has more local knowledge of key storage, but can pass on requests for keys that are far away by halving the distance to the key at each step. Figure 10 shows an example of a new node, 5, joining the network. When the new node joined the network it was assigned to position five on the ring by the bootstrap host. The bootstrap host also informed 5 that its immediate predecessor was node 3. Node 5 contacts node 3 to let 3 know that 5 is now its successor. Node three tells node 5 that its successor is node 0. Node 5 contacts node 0 to let 0 know that 5 is now its predecessor, and to take control of any (key, value) pairs on the interval [5, 3). In this example key 4 migrates to node 5.

There are two ways that a node may leave the network: A planned exit, or an unplanned exit. In the case of a planned exit, the node simply contacts its predecessor and successor and transfers its keys to its successor. To handle the case of an unplanned exit, the protocol is modified in two ways: First, rather than a single successor, each node keeps a list of successors. Second, each key, value pair is replicated k times along the successor path starting with the first node that should be responsible. Each node in the network periodically runs a stabilization routine in which it verifies that its predecessors and successors are still intact. The stabilization routine allows each node to keep its list of successors up to date, and to update its predecessor should the predecessor drop out.

There are two ways we could implement the PocketLens algorithm on top of the Chord architecture. The first option, which we call IU-Chord, allows each user to simply publish their pseudonym and ratings as a (key, value) pair on the

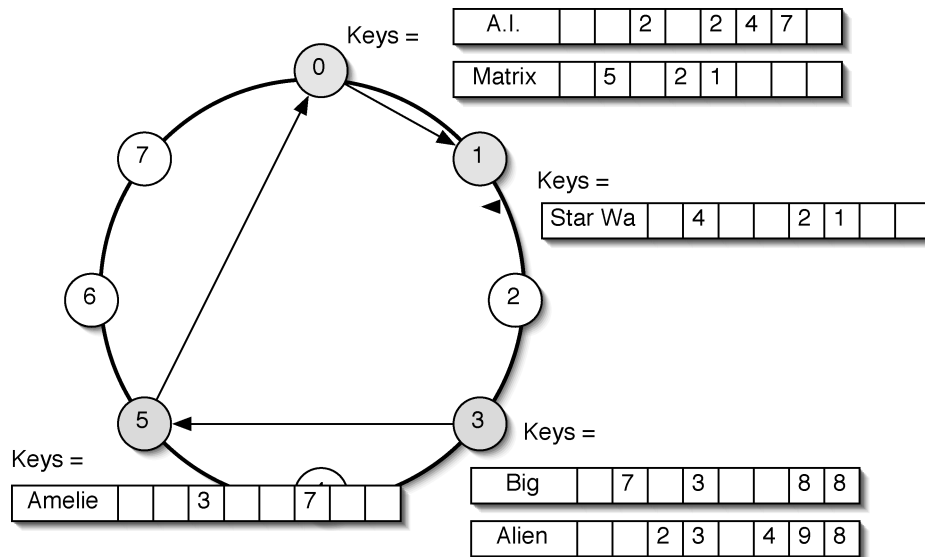


Fig. 11. II-Chord stores each row of the item-item matrix at a node.

network. We call this option IU-Chord since both the Items and the Users are explicit. The second option, called II-Chord (since we are constructing an item-item matrix) uses the network as a distributed storage mechanism to maintain a shared copy of the whole item-item matrix.

The IU-Chord implementation of PocketLens is equivalent to the centralized approach described in Section 4.1, except that a client is responsible for discovering its own best set of neighbors through trial and error. A client simply traverses the ring, finding sets of ratings. Over time, the client learns the best set of neighbors. Then, whenever the client wants to build a new model it gets the latest copy of the ratings from its best neighbors. The downside to this approach is that a user must be willing to publish some set of ratings that everyone on the network is entitled to use.

The II-Chord implementation of PocketLens is illustrated in Figure 11. The (key, value) pairs stored at the nodes of II-Chord are item-identifiers and their corresponding rows of the item-item matrix. Using this approach, a user can simply make a local copy of all rows corresponding to items he has rated and use that as his private model for making recommendations.

The key to II-Chord is that the item-item matrix is built by the entire community. When a new user joins the community she updates the row vectors for the items she has rated as specified in Section 3. The next time any other user requests the set of row vectors, he retrieves the new row vector in addition to the previously existing set. After retrieving the row vectors, he can use the usual item-item computations to compute recommendations or predictions.

As an optimization, if each user sorts their items in increasing order and only updates the matrix entries in sorted order, the matrix is symmetric, and the upper triangle is sufficient for the calculations.

4.5 Secure Blackboard

The algorithms we have examined so far have relied on anonymity or pseudonymity to protect each user's privacy. Suppose that it was possible for a user to first encrypt her profile before she shared it with other users in the community. Furthermore, suppose that the community of users had a function that would correctly add the profiles together without needing to decrypt them in order to do the addition. Finally, suppose that after all of the pieces of the profile were added together, the community could decrypt the final results and use the model. The steps we have just described are possible through the use of ElGamal encryption algorithms and homomorphic mathematics [Pedersen 1991].

The set of operations described in the previous paragraph has been applied to a secure online voting protocol in Cramer et al. [1997] and in collaborative filtering in Canny [2002a]. As discussed in Section 2.3 Canny [2002a] uses an SVD approach to collaborative filtering, we propose that these same techniques can be applied to our PocektLens algorithm to achieve a high degree of security. However, in this paper we do not prove that all of the security properties still hold when using the fundamental techniques in a new way.

Since the ElGamal scheme is a homomorphic encryption we can use the same property of homomorphic encryption that secure election schemes use. The important property is that we can obtain a tally of the votes without decrypting the individual votes. More precisely, we say that E is a homomorphic encryption scheme if, given $c_1 = E_{r_1}(m_1)$ and $c_2 = E_{r_2}(m_2)$ there exists an r such that:

$$c_1 * c_2 = E_r(m_1 + m_2) \quad (5)$$

In essence we multiply the encrypted messages to get the sum of the original messages [Cramer et al. 1997].

Any of the algorithms using a secure blackboard can be implemented in a peer-to-peer network making use of a write once, read many (WORM) blackboard, and a secure source of random bits. The WORM capability is available in both the Oceanstore [Kubiatowicz et al. 2000] and Groove peer-to-peer systems. The secure source of random bits must be shared by all participants in the secure blackboard.

As we did in Section 2.2.1 we will examine a very simple item-item model where we count the number of times that items have been copurchased. We can think of the model construction as an on-line election, where each pair of copurchased items represents a vote.

Each client in the network prepares a set of votes for each of the copurchased items in its local profile. Each vote is encrypted using an ElGamal encryption algorithm [Pedersen 1991].

The Cramer protocol ensures that users are not trying to supply illegal data to ruin the results of the vote by using zero knowledge proofs for each of the encrypted vote messages. The Cramer protocol also guarantees the privacy of individual votes and prevents vote stuffing in the presence of persistent pseudonyms [Cramer et al. 1997]. Not even the Cramer protocol is sufficient to prevent vote stuffing if pseudonyms are cheap enough so that users are willing to make many pseudonyms. In that case, no current collaborative filtering

system can prevent vote stuffing. The many pseudonym problem has been studied extensively by Friedman and Resnick [1999].

We can generalize the secure voting scheme to building an item-item model using cosine similarity. The PocketLens algorithm for building the model amounts to simply adding individual user components to each row vector of the model. These additions can be done as homomorphic operations on the encrypted data.

In the description that follows, we intentionally treat the details of the security aspects of the algorithm at a high level. Details of the security components, including proofs, can be found in Pedersen [1991] and Cramer et al. [1997].

The outline of the process for building a model using the PocketLens algorithm is as follows:

- (1) Run the Pedersen [1991] protocol to generate an ElGamal public key and shared private key.
- (2) Each client computes the partial dot products for their ratings vectors.
- (3) Each client computes a zero knowledge proof that their data is valid.
- (4) Each client encrypts their partial dot products with the ElGamal public key, and posts them to the blackboard.
- (5) Each client acts as a tallier and uses the source of random bits to choose a subset of the partial dot products for validity checking of the zero knowledge proof.
- (6) All talliers vote on the validity of each proof.
- (7) Each tallier totals results for all the partial sums that are valid. Validity is determined by majority vote.
- (8) Use the Pedersen protocol to decrypt the final totals.

The result of the secure model-building process is that each client creates its own decrypted version of the full item-item model. We will call this architecture SBB+PocketLens. The SBB+PocketLens model can then be used locally by the client to make predictions.

5. SIMULATION RESULTS

To test the different architectural ideas we have just described, we wrote a simulation framework that allowed us to try the PocketLens algorithm, and compare the quality of the recommendations provided by each architecture.

The data-set we used for our investigation of the Central, Random and Transitive architectures was constructed from a random sample of user movie ratings collected on our MovieLens research website. To be included in our random sample, a user had to have rated at least 20 movies. Our data included 96,000 ratings for 3,775 movie titles and a population of 1,000 users. We do not report results for IU-Chord as they are equivalent to the transitive architecture.

The sample data-set was used to randomly create 10 trial data-sets. Each trial data-set consisted of a training file and a test file. The training file contained all of each user's ratings except one. The test file contained one rating for each user.

The data set for the II-Chord and Secure-Blackboard (SBB) architecture was again taken from our MovieLens research website. However, because II-Chord and SBB are models built by the entire community we wanted to look at the impact of the community size rather than an individual user's neighborhood. So that we could randomly sample from a large data set, we started with the total MovieLens database. The full database consists of six million ratings for 5700 movies and 68,000 users. To simulate different community sizes we selected groups of 500, 1000, and 2000 users at random to construct the community model.

5.1 Experimental Methods

To test the Central, Random, and Transitive Architectures we ran experiments where we alternated between training and testing while adding neighbors to the neighborhood. During each training cycle five new neighbors were added to the model according to the method defined by the architecture. When the training cycle was complete, a test cycle was performed where we generated a set of recommendations for the model owner. After each test cycle, data was collected to compute the metrics outlined below. In each case, the model was started from scratch and continually enlarged until it had incorporated ratings from 100 neighbors.

To test the II-Chord and SBB architectures, a community model was built using community sizes of 100, 500, 1,000, and 2,000 users. We randomly reserved one or more ratings from each user to be a part of the test set. The reserved ratings were not used in building the model. After the model was built, test recommendations were generated for each user's reserved item(s).

5.2 Metrics

Over the years many metrics have been proposed for comparing collaborative filtering algorithms. In Herlocker et al. [1999], Herlocker compares a wide range of metrics and recommends coverage and mean absolute error for comparing various algorithms. Where possible we have adopted the metrics recommended by Herlocker.

Average Similarity. Because our architectures primarily differ in the way that the neighbors are chosen it is important to incorporate one metric that directly measures the quality of the neighborhood. The average similarity is a measure of how close a group of neighbors is to the model owner. The average similarity is simply calculated by summing the cosine similarity for the individual neighbors, and dividing by the number of neighbors. Rather than computing the average similarity as a running average over the entire neighborhood, we compute it for each new group of neighbors to sharply highlight how well each architecture performs as the neighborhood size grows.

Mean Absolute Error. (MAE) is a measure of how accurately we can predict a user's rating for an item. Define R_i to be the rating for item i , and P_i to be the prediction for item i . The MAE is then defined as $\sum_n \frac{|P_i - R_i|}{n}$. Mean absolute error (MAE) has been used to measure prediction engine performance for many different algorithms [Shardanand and Maes 1995; Sarwar et al. 2000a;

Herlocker et al. 1999]. Other metrics that have been used are root mean squared error and the correlation between ratings and predictions.

Recall is a measure of how often a list of recommendations contains an item that the user has actually rated. We calculate the recall by generating a list of the top 10 recommended items for each user and then testing whether or not an item left out of the user's training data is present on the top 10 list. We should note that recommender systems research generally cannot use the traditional information retrieval recall metric because there is no ground truth about which items are good, and it is infeasible to ask a set of users to, for example, watch and evaluate each of 4000 movies. The intuition behind this simpler recall measure is that the withheld item, being one the user has chosen to see, is one that has a high probability of being one that the system should recommend.

Coverage is a measure of the percentage of items for which a recommendation system can provide predictions. There are two possible ways to compute the coverage metric. One way is to look globally at the universe of items and compute the percentage of those items for which the system could make a prediction (averaged across users). This approach distorts the actual user experience in our usage scenarios, as it penalizes a system that cannot readily make predictions for items the user is not expected to like; accordingly, it penalizes compact models. Instead, we compute coverage across the set of withheld test ratings: items that we know the user has chosen to see. We report the percentage of these withheld items for which the system could generate a prediction.

Memory Usage is a measure of how large the model grows as more items and neighbors are incorporated. At each step in the training cycle we recorded how many cells in the matrices that comprise the model were used. The number of bytes can be computed from the cells by multiplying by the size of the internal representation.

5.3 Results

We now compare each of the reference architectures with respect to the metrics defined in Section 5.2.

Neighborhood Similarity. Since the architectures primarily differ in the way the neighborhood is created we will start with a direct measure of the quality of the neighborhood. Figure 12 shows the average similarity score of each successive group of five new neighbors added to the model. These figures are based on a "steady-state" rebuild of the model; we assume that the central server is initialized with all user models and that the transitive traversal starts from a set of historic best neighbors. The *central server* architecture starts with the best neighbors and works its way down. Hence, it illustrates a gradual decline in the quality of successive neighbors. By contrast, the *random* series shows constant, but much worse, neighbor quality. The *transitive traversal* architecture starts by creating a neighborhood that is the same quality as the central server. After about 25 users, however, transitive traversal starts finding less similar users, though ones that are still much closer than the random ones.

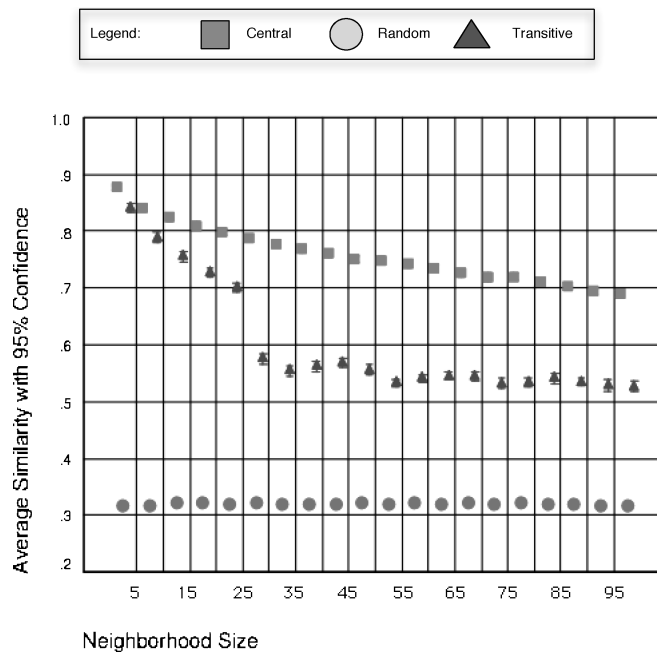


Fig. 12. Average similarity of users in the model as neighborhood size grows. Shown with 95% confidence intervals.

Because the set of users chosen to build the community model in *II-Chord* and *SBB* is selected at random, the neighborhood similarity for those architectures is the same as for *random*.

Coverage. Although not as important for our application scenario, coverage is another indicator of the quality of the neighborhood. Ideally, the users should not only be similar to you, but should also provide you with predictions for a wide range of desirable items. Random neighborhood construction allows us to make predictions for 83% of the items with a neighborhood size of 25 users. This is 10% less than the central server and transitive traversal reference architectures at the same neighborhood size. When the neighborhood grows to 100 users the coverage grows to 92% for random neighborhood construction. Figure 13 shows a graph of how coverage improves as the neighborhood size grows.

The coverage for models built using *II-Chord* or *SBB* is very good. With a community of only 100 people coverage is 71%, but with a community of 500 people coverage climbs to 95% and continues upward to 99% coverage with a community of 2000. This shows that even a relatively small group of people can build a community model and receive benefit.

Mean Absolute Error. Next we looked at the MAE scores generated for predictions. Figure 14 shows that all three architectures provide nearly the same value for MAE; in fact the 95% confidence intervals overlap all three means. In Sarwar et al. [2001] Sarwar reports an MAE of 0.72 for a full item-item matrix which is also inside the error bars for these three architectures.

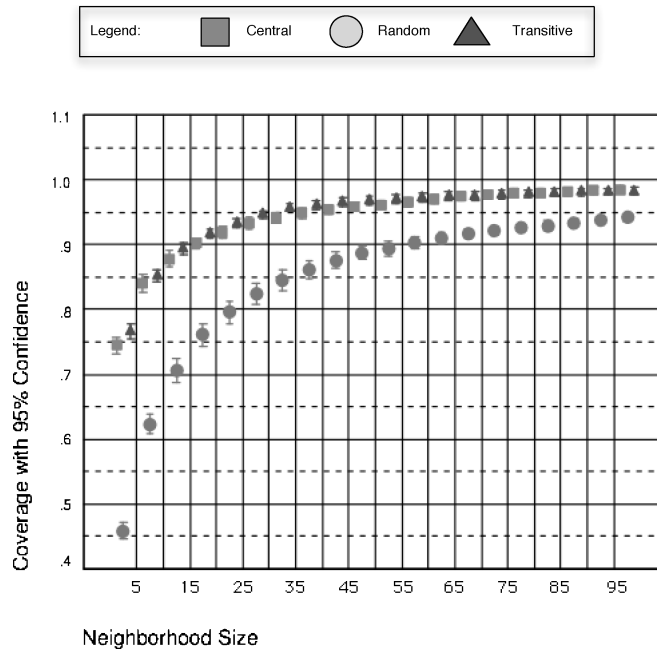


Fig. 13. Coverage as neighborhood size grows. Shown with 95% confidence intervals.

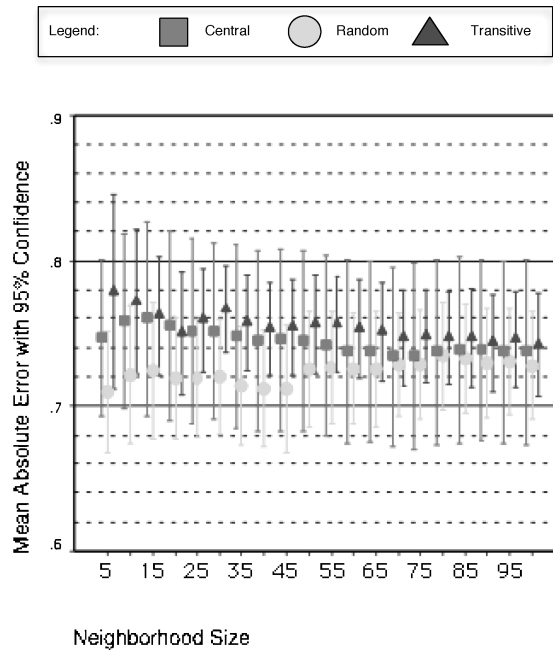


Fig. 14. MAE as neighborhood size grows. Shown with 95% confidence intervals.

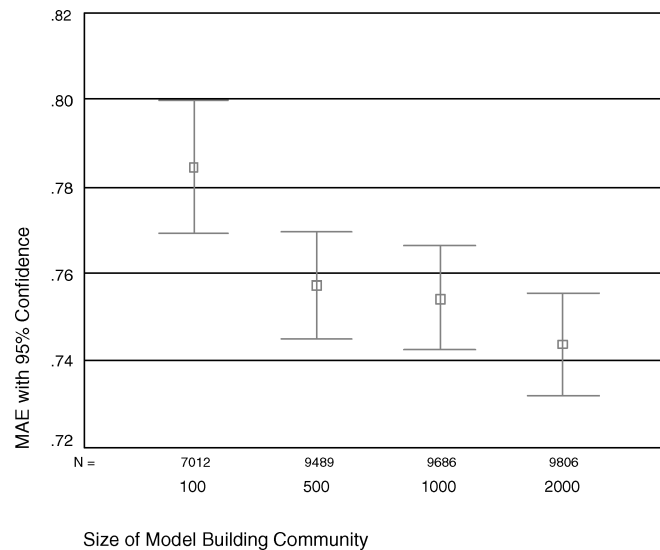


Fig. 15. Mean Absolute Error for community built models, II-Chord and SBB. Shown with 95% confidence intervals.

The MAE reported by Sarwar et al. [2001] is compared against other current algorithms and is as good as any reported to date.

Figure 15 illustrates the MAE for the II-Chord and SBB architectures at community sizes of 500, 1000, and 2000 users. Even for a very small community of 500 users we still get an MAE that is within the error bars around the MAE measured for all the previous architectures. At large community sizes we get an MAE value within two percent of the published results for other item-item based architectures [Sarwar et al. 2001]. The slight differences between our MAE and those reported previously could be explained by differences in the data-set. The data-set used in Sarwar et al. [2001] was a dense subset of the MovieLens ratings database, whereas we constructed a ratings database to more accurately reflect the character of the current ratings database.

Recall. Measuring the quality of the neighborhood helps us compare architectures, but it is the quality of the recommendations themselves that the users will experience. Recall helps us see whether we are making recommendations for items that our users will recognize and value. The *central server* series of Figure 16 shows how the recall changes as the neighborhood size grows. The graph shows that the recall levels off at 0.28 for a neighborhood size of 25 users.

Based on the average similarity, one might expect that the random discovery architecture will give very poor recommendations. However, random neighborhood selection still gives us a recall of 0.20 at the 25 user mark, and continues to improve to 0.25 at the 75 user mark. The *random* series in Figure 16 illustrates both the difference between the central server and the random discovery architecture as well as the improvement in recall as the neighborhood grows.

The *transitive* series shows that with the transitive traversal architecture, the recall measure is nearly equal to the central server architecture for a

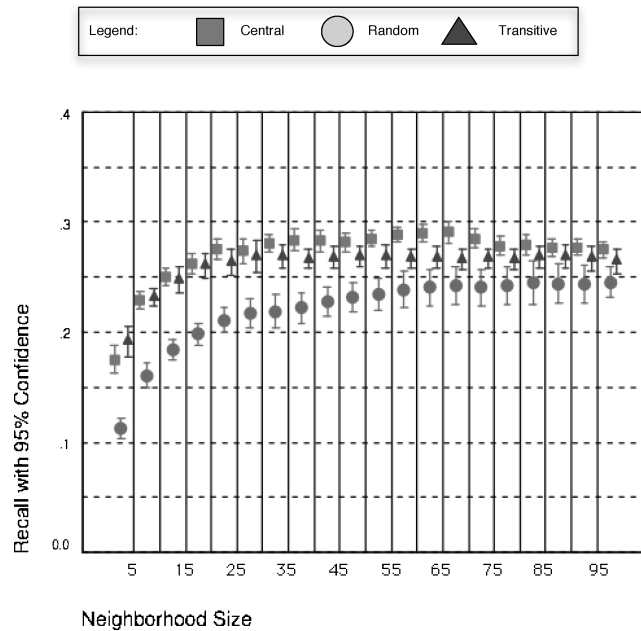


Fig. 16. Recall as neighborhood size grows. Shown with 95% confidence intervals.

neighborhood size of 25 users at 0.27, and is substantially better than random discovery at 0.21. In our experiments we evaluated the transitive traversal architecture under many different training conditions. The results shown in Figure 16 reflect the recall after the algorithm has determined its best 25 neighbors from a sample of 250 random neighbors.

Both the central server and transitive architectures exhibit a gradual decay in the recall after about 25 users. We investigated this decay from two perspectives. First, we compared the recall using the top 50 items to the recall using the top 10 items. The graph of the top 50 recommendations peaks at a recall of 0.5 but does not decline. Next, we looked at the average similarity score for the items in the recommendation lists. We found that the average similarity was only 15% better using top 10 than using top 50.

The recall values for the II-Chord and SBB architectures are constant and equal to the Central server architecture at 0.25. Since the SBB algorithm is based on an SVD computation, we also implemented and measured the recall for a model built using the previously-studied SVD recommender algorithm [Sarwar et al. 2000b]. We measured the recall for the SVD models to be 0.014. The poor performance is because this implementation of the SVD model requires a complete matrix, and fills in the missing matrix entries with the item averages. The resulting model returns recommendations that are very similar to the most popular overall items.

6. DISCUSSION

The genesis of our research into *personal recommenders* was the twin problems of *portability* and *trust* in recommender systems. To increase portability, we

Table III. Effect of Model Truncation in the PocketLens Algorithm on Size, Speed, Coverage, and MAE

Row Trunc	Model Size	MAE	Coverage	Pred Time (sec/pred)	TopN Time (sec/topn)
10	378k	0.76	.49	.013	.048
50	1.3m	0.74	.84	.014	.069
100	2.0m	0.72	.94	.014	.096
500	2.0m	0.70	.998	.016	.263
1000	2.3m	0.70	.999	.017	.444

sought recommender systems that could be available to users wherever and whenever they wanted, even when they were not near a desktop computer—or even an Internet connection. To increase trust, we sought recommender systems that would enable users to decide how much of their personal information to share, and that would allow users to share their information while preserving their anonymity.

In Section 5 we studied the accuracy of the item-item algorithm in the peer-to-peer context. We found that there is a minimum number of neighbors required by the system to achieve an acceptable quality level. The acceptable quality level is defined in terms of both mean absolute error (MAE) and recall.

In Table III we summarize the scalability aspects of the PocketLens algorithm. One way to improve scalability is to truncate each row of the matrix, keeping the n most similar items for that row. Since most users have a limited number of items they will be interested in, the truncated model may give acceptable quality and coverage while improving scalability. Table III shows the impact on MAE, coverage, prediction and topn times for different amounts of model truncation.

The data in Table III represents an estimate of the performance on a Sharp Zaurus SL-5500 PDA. The data was obtained experimentally by measuring performance and model sizes on a desktop computer. We then used a set of benchmark Java programs to compare the performance characteristics of the desktop hardware with the J2ME Personal Profile (<http://java.sun.com/j2me>) on the Zaurus. The benchmark times were used to scale the desktop prediction and topn times to estimate the performance on the Zaurus.

Table III illustrates the tradeoffs between quality, measured by coverage and MAE; and performance, measured by prediction and topn times. For example, in comparing the first and last rows of the table you can see that coverage doubles and MAE is improved by 8%, however topn performance is reduced by a factor of ten and memory utilization increases by a factor of six. However, the table shows that on average-sized PDAs a user can achieve the maximum coverage and best MAE while computing recommendations in less than a second.

Figure 17 compares the architectures with respect to data security and the complexity of cooperating with the community to update a model. The central, random, and transitive architectures allow each user to build their own small model, on their own time, with very little interdependence on the availability of other peers. The disadvantage of these architectures is that they are open to an

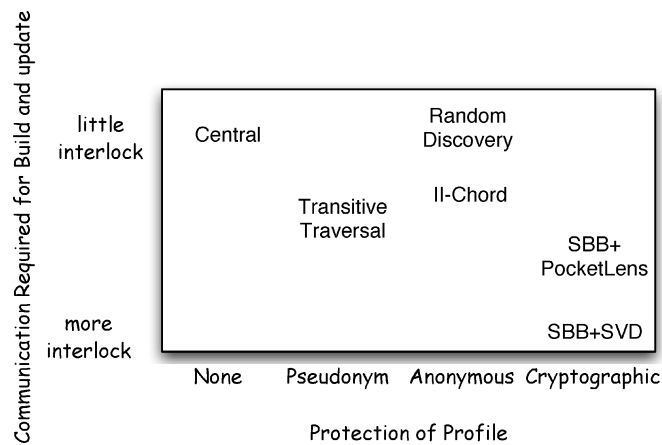


Fig. 17. Comparison of different architectures with respect to data security and complexity of model building.

attack where an attacker can collect data from anonymous individuals and use it for their own purposes. For instance, the attacker might be able to prepare more successful shilling¹ attacks based on a completely anonymous dataset, because the attacker could create fake users who are similar to the real users in the system.

The Random and Transitive architectures offer some privacy by building on top of the Gnutella architecture. Gnutella clients can record IP addresses of other clients, which may provide insufficient privacy for some applications. Additional layers of anonymity may be created by using onion routing [Goldschlag et al. 1999], or the Tarzan peer-to-peer system for IP routing [Freedman and Morris 2002]. These two systems allow the content of messages to be encrypted, while hiding the source and destination IP addresses from a traffic analysis attack.

The II-Chord architecture solves the problem of a malicious user collecting data by ensuring that any individual user's profile data is only added to a public aggregate by the owner of the profile. Although the user need never trust anyone with their individual profile, the public nature of the aggregate data makes possible an attack where a malicious user manipulates the aggregate data to achieve certain recommendations. For instance, the attacker might write to a client that ensures that the movie they are promoting is very similar to all other popular movies by simply overwriting the calculated similarity scores in the model. Since the II-Chord model is under continuous construction, it is the only architecture that makes it difficult for a user to take back a rating. (It is easy to delete a rating when you start the model over from scratch.) The II-Chord architecture adds additional complexity to the model building process

¹*Shilling* is a type of attack on a recommender system in which the attacker creates many fake ratings of items to influence the predictions shown to users. For instance, the author of a book might create thousands of fake users who all rate his book highly. Collaborative filtering has a partial defense against shilling because only users who have agreed historically with a target user are used in computing recommendations.

in the underlying network, as well as an assumption that there will be enough hosts continually online to ensure the continuity of the model.

The secure blackboard architectures can reliably prevent malicious users from keeping the profiles of their peers by only exposing encrypted copies of profile data. In addition, the family of secure blackboard architectures avoids the problem of exposing the aggregate to manipulation because each user stores a full copy of the model locally. However this security comes at a price that is paid in additional complexity, both in implementation and in the communication interlock that is required of the users in building the model. Interlock refers to the interdependence between individual users in building a community model. In any of the secure blackboard architectures, interlock comes from the security protocols that have certain steps where all the users must be in sync before they can move forward. For example, all individual results must be posted to the blackboard before they can be verified and summed. SBB+PocketLens is somewhat less susceptible to these scalability problems because building the item-item model does not require multiple iterations to achieve model convergence like SBB+SVD.

The SBB+PocketLens implementation extends Canny [2002a, 2002c] by allowing each user to incrementally update their own copy of the model with their own profile data at any time. This approach will allow users to benefit from their own new ratings, while allowing the community to run the model-building process less frequently. Finally, the SBB+PocketLens approach adds additional efficiency by allowing a group of users to work together to update the community with new ratings, rather than building an entirely new model. Whereas SBB+SVD requires a new model to be built from scratch each time, the SBB+PocketLens model can continue to expand without starting over.

Of the five architectures, the II-Chord architecture is the only one to meet all four goals simultaneously. This architecture incorporates a community model-building process that allows users the highest degree of control over which ratings they wish to contribute to the community. Even when users do contribute ratings to the community, the contribution is in the form of a change to an aggregate model; thus the user never needs to explicitly share a rating with anyone. Secondly, because PocketLens separates the creation of the similarity model from the use of the model to generate recommendations, users are able to have recommendations even when they are off-line, satisfying our goal for *portability*. The memory requirements for the community based model are such that even a low end palm pilot with a small amount of memory could store the model for an average user. Finally we experimentally verified that it is possible to attain highly accurate recommendations within this architecture. The limitation of II-Chord is that it is easily susceptible to attack by malicious users who want to subvert the recommendations.

7. CONCLUSION AND FUTURE WORK

Recommender systems are providing value to users in many different content and commerce environments. Two limitations of most recommender systems

are *portability* and *trust*. In this paper we have described a *personal recommender* based on the PocketLens algorithm for collaborative filtering in a peer-to-peer environment. The PocketLens algorithm is portable enough to run on disconnected palmtop computers, and can protect the user's privacy and provide trusted recommendations.

To demonstrate the tradeoffs in performing collaborative filtering in a peer-to-peer environment we looked at five architectures that use the PocketLens algorithm. We found that the quality of the recommendations is as good as the best previously reported results [Sarwar et al. 2001]. Finally, we qualitatively examined the trade-offs in privacy and security that each architecture provides. No one architecture is perfect, but the best of the architectures provides fast, portable recommendations with privacy protection, and good quality.

There are several interesting research issues to address in order to successfully take our architectural design and turn it into a working system. One future project would be to implement the clients within a security framework that ensures the integrity of the host systems, and protects the network against a rogue client that collects every user's ratings. Another project would be to implement the SBB+PocketLens architecture directly and find out how important the coordination challenges are in a real world setting. A third project would be to investigate ways for content addressable architectures to more efficiently search for neighbors based on similarity in their rating vectors. Finally, the security of SBB+PocketLens architecture needs to be carefully validated to ensure that we have not introduced any security holes by using the techniques of Cramer et al.[1997] and Canny [2002b] in a new way.

Another project is to investigate interface issues for helping users manage and control their profile information. The personal recommenders described in this paper can work in an environment in which users choose which of their ratings to share, and which others to share them with. How will users specify their preferences? How will the system show them the effect of their choices?

Finally, all of the architectures are susceptible to the same shilling attacks that may threaten centralized recommenders. An attacker may make many profiles and enter biased ratings data under those profiles to influence the recommendations. Shilling is an interesting and open problem for recommender systems in general. Once solutions are known for centralized recommenders, they may need to be adapted to solve the shilling problem for distributed recommenders.

Beyond these clear next steps, portability and trust issues in recommenders open a broad range of important questions. Over the long-term, will recommenders primarily serve the interests of the businesses who deploy them, the interests of the consumers who seek recommendations, or a combination of the two? Exactly what sort of privacy guarantees are possible from a technology that fundamentally requires a deep understanding of its users? What does it mean for a recommender to be portable in an increasingly connected world? Is it sufficient to have a PDA-based interface to a centralized recommender? The present research has taken a few steps forward in this rich space, but there are many questions as yet unexplored.

ACKNOWLEDGMENTS

We gratefully acknowledge the careful reviews from the anonymous reviewers, and the shepherding by the editor. We especially benefited by feedback from the broad range of reviewers, helping to ensure the presentation is accurate despite the wide range of issues it discusses. We would like to express our appreciation to the present and past members of the GroupLens Research Group.

REFERENCES

- ACKERMAN, M. S., CRANOR, L. F., AND REAGLE, J. 1999. Privacy in e-commerce: Examining user scenarios and privacy preferences. In *ACM Conference on Electronic Commerce*. 1–8.
- AGGARWAL, C., WOLF, J., WU, K., AND YU, P. 1999. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 201–212.
- ANDERSON, D. P., COBB, J., KORPELA, E., LEBOFKY, M., AND WERTHIMER, D. 2002. Seti@home: An experiment in public-resource computing. *Comm. ACM* 45, 11 (November), 56–61.
- AP. 2002. New shopping technology could breed supermarket class system. *San Jose Mercury News* (November 10).
- BILLSUS, D. AND PAZZANI, M. J. 1998. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 46–54.
- BOUCH, A., KUCHINSKY, A., AND BHATTI, N. 2002. Quality is in the eye of the beholder: Meeting users' requirements for internet quality of service. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, 297–304.
- BRESE, J. S., HECKERMAN, D., AND KADIE, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*. 43–52.
- CANNY, J. 2002a. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy*. 45–57.
- CANNY, J. 2002b. Collaborative filtering with privacy: To appear. In *IEEE Conference on Security and Privacy*.
- CANNY, J. 2002c. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.
- CHAVEZ, A. AND MAES, P. 1996. Kasbah: An agent marketplace for buying and selling goods. In the *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*. Practical Application Company, London, UK, 75–90.
- CHEN, Y., EDLER, J., GOLDBERG, A., GOTTLIEB, A., SOBTI, S., AND YIANILOS, P. 1999. A prototype implementation of archival intermemory. In *Proceedings of the Fourth ACM International Conference on Digital Libraries*.
- CLARKE, I., HONG, T. W., MILLER, S. G., SANDBERG, O., AND WILEY, B. 2002. Protecting free expression online with freenet. *IEEE Internet Computing*. 6, 1 (January), 40–49.
- CLAYPOOL, M., LE, P., WASEDA, M., AND BROWN, D. 2001. Implicit interest indicators. In *Proceedings of the ACM Intelligent User Interfaces Conference (IUI)*.
- CLYMER, A. 2003. Troops risk identity theft after burglary. *New York Times* (January 12).
- CRAMER, R., GENNARO, R., AND SCHOENMAKERS, B. 1997. A secure and optimally efficient multi-authority election scheme. *Lecture Notes in Computer Science* 1233, 103–118.
- DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. 2001. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*. Chateau Lake Louise, Banff, Canada.
- DEVORE, J. L. 1995. *Probability and Statistics for Engineering and the Sciences*, fourth edition Duxbury Press.
- DINGLEDINE, R., FREEDMAN, M. J., AND MOLNAR, D. 2001. The free haven project: Distributed anonymous storage service. *Lecture Notes in Computer Science* 2009.

- DRUSCHEL, P. AND ROWSTRON, A. 2001. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of HOTOS*. 75–80.
- FONER, L. 1999. Political artifacts and personal privacy: The yenta multi-agent distributed matchmaking system. Ph.D. thesis, Massachusetts Institute of Technology.
- FONER, L. N. 1996. A multi-agent referral system for matchmaking. *Proceedings of the Second International Conference on Multi-Agent System (ICMAS-96)*.
- FREEDMAN, M. J. AND MORRIS, R. 2002. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*. Washington, D.C.
- FRIEDMAN, E. AND RESNICK, P. 1999. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10, 2 (August), 173–199.
- GNUTELLA. Gnutella website <http://gnutella.wego.com>.
- GOECKS, J. AND SHAVLIK, J. 2002. Learning users' interests by unobtrusively observing their normal behavior. In *Proceedings of the ACM Intelligent User Interfaces Conference (IUI)*.
- GOLDSCHLAG, D., REED, M., AND SYVERSON, P. 1999. Onion routing for anonymous and private internet connections. *Comm. ACM* 42, 2 (February), 39–41.
- GOOD, N., SCHAFER, B., 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 1999 Conference of the American Association of Artificial Intelligence (AAAI-99)*.
- HANSELL, S. 2002. Privacy policy on web shifts as profits ebb. *New York Times* (April 11).
- 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 (SIGIR-99)*.
- JOVANOVIĆ, M. A. 2001. Modelling large peer-to-peer networks and a case study of gnutella. M.S. thesis, University of Cincinnati.
- KARGER, D., LEHMAN, E., LEIGHTON, T., LEVINE, M., LEWIN, D., AND PANIGRAHY, R. 1997. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*. 654–663.
- KARYPIS, G. 2001. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the 10th Conference of Information and Knowledge Management*.
- KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. 2000. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM.
- LYMAN, P. AND VARIAN, H. 2000. How much information? <http://www.sims.berkeley.edu/how-much-info>.
- MAES, P. 1994. Agents that reduce work and information overload. *Comm. ACM* 37, 7 (July), 30–40.
- MAES, P. AND KOZIEROK, R. 1993. Learning interface agents. In *Proceedings of AAAI-93 and IAAI-93*. AAAI Press; Menlo Park, CA, USA.
- MILLER, B., RIEDL, J., AND KONSTAN, J. 2002. *From Usenet to CoWebs: Interacting with Social Information Systems*. Springer Verlag, Chapter Experiences in Applying Collaborative Filtering to a Social Information System.
- MINAR, N. 1998. Designing an ecology of distributed agents. M.S. thesis, MIT.
- MINAR, N., GRAY, M., ROUP, O., KRİKORIAN, R., AND MAES, P. 1999. Hive: Distributed agents for networking things. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*. Palm Springs, CA, USA.
- MOBASHER, B., COOLEY, R., AND SRIVASTAVA, J. 2000. Automatic personalization based on Web usage mining. *Comm. ACM* 43, 8, 142–151.
- MOBASHER, B., DAI, H., AND TAO, M. 2002. Discovery and evaluation of aggregate usage profiles for web personalization. *Data Mining and Knowledge Discovery* 6, 61–82.
- MORITA, M. AND SHINODA, Y. 1994. Information filtering based on user behavior analysis and best match text retrieval. In *Proceedings of the 17th International Conference on Research and Development in Information Retrieval. SIGIR 94*, W. Croft and C. van Rijsbergen, Eds. Springer-Verlag; Berlin, Germany, 48.
- MOUKAS, A. AND ZACHARIA, G. 1997. Evolving a multi-agent information filtering solution in amalthaea. In *Proceedings of Autonomous Agents 97*.

- NEUMANN, P. G. 1995. *Computer Related Risks*. Addison Wesley.
- PEDERSEN, T. 1991. A threshold cryptosystem without a trusted party. In *Advances in Cryptology - EUROCRYPT'91 Lecture Notes in Computer Science*, D. W. Davies, Ed. Vol. 547. Springer-Verlag, 522–526.
- RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. 2001. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*.
- RESNICK, P., IACOVOU, N., SUSHAK, M., BERGSTROM, P., AND RIEDL, J. 1994. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of CSCW 1994*. ACM SIG Computer Supported Cooperative Work.
- RIPEANU, M., FOSTER, I., AND IAMNITCHI, A. 2002. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Comput. J.* 6, 1.
- ROWSTRON, A. AND DRUSCHEL, P. 2001a. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science* 2218.
- ROWSTRON, A. I. T. AND DRUSCHEL, P. 2001b. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*. 188–201.
- SARWAR, B., KONSTAN, J., 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 1998 Conference on Computer Supported Cooperative Work*.
- SARWAR, B., KONSTAN, J., AND RIEDL, J. 2001. *Internet Commerce and Software Agents: Cases, Technologies, and Opportunities*. Idea Group, Chapter Distributed Recommender Systems: New Opportunities for Internet Commerce, Idea Group Publishing, Hershey, PA.
- SARWAR, B. M., KARYPIS, G., KONSTAN, J. A., AND RIEDL, J. 2000a. Analysis of recommender algorithms for e-commerce. In *ACM E-Commerce 2000*. 158–167.
- SARWAR, B. M., KARYPIS, G., KONSTAN, J. A., AND RIEDL, J. 2000b. Application of dimensionality reduction in recommender system—a case study. In *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*.
- SARWAR, B. M., KARYPIS, G., KONSTAN, J. A., AND RIEDL, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International World Wide Web Conference (WWW10)*. Hong Kong.
- SHARDANAND, U. AND MAES, P. 1995. Social information filtering: Algorithms for automating ‘word of mouth’. In *Human Factors in Computing Systems CHI '95 Conference Proceedings*. 210–217.
- SHETH, B. AND MAES, P. 1993. Evolving agents for personalized information filtering. In *Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications*. IEEE Computer Society Press; Los Alamitos, CA, USA.
- SHIRKEY, C. 2001. *Peer-to-Peer, Harnessing the Power of Disruptive Technologies*. O’Reilly and Associates, Chapter Listening to Napster.
- STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. 2001. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*. 149–160.
- TERVEEN, L., HILL, W., AMENTO, B., McDONALD, D., AND CRETER, J. 1997. Phoaks: A system for sharing recommendations. *Comm. ACM* 40, 3 (Mar.), 59–62.
- UNGAR, L. AND FOSTER, D. 1998. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*. AAAI Press, Menlo Park California.
- WALDMAN, M. AND MAZI, D. 2001. Tangler: a censorship-resistant publishing system based on document entanglements. In *ACM Conference on Computer and Communications Security*. 126–135.
- WALDMAN, M., RUBIN, A. D., AND CRANOR, L. F. 2000. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proceedings of the 9th USENIX Security Symposium*. 59–72.
- WHELAN, C. 2002. How consumers can strike back if their identity has been stolen. *Wall Street Journal* (August 21).
- WINGFIELD, N. AND PEREIRA, J. December, 2002. Amazon uses faux suggestions to promote new clothing store. *Wall Street Journal* (December 4).

Received January 2003; revised June 2003, October 2003; accepted January 2004