Improved Recommendation Systems*

Extended Abstract

Baruch Awerbuch† Boaz Patt-Shamir‡ David Peleg§ Mark Tuttle¶

Abstract

We consider a model of competitive recommendation systems proposed by Drineas *et al.* [4]. In recommendation systems (e.g., for books or movies), the system tracks which product each user chose in the past, and tries to deduce which other products an asking user is likely to be satisfied with. Obviously, recommendation systems can be effective only for users who share preferences with many other users. Such users are said to belong to a "dominant type." Current approaches to on-line recommendation systems involve using Singular Value Decomposition (SVD), which is computationally intensive and, more important, often applicable only under additional strong conditions. Specifically, correctness is guaranteed in [4] only if users of different dominant types essentially do not share a product they like ("type separability"), and only if the number of users in non-dominant types is significantly smaller than the number of users in dominant types ("gap assumption"). The complexity of that algorithm is $O(mn)$, where m and n denote the number of users and products, respectively. Here, it is shown that in fact, very simple combinatorial algorithms can make good recommendations without using SVD. Our algorithms require neither the type separability nor the gap assumption, they are naturally amenable to distibuted computation, and their complexity is lower. In particular, the paper presents an $O(m+n)$ time centralized algorithm and a distributed algorithm that can be implemented in a peer-to-peer model even in the presence of adaptively colluding malicious players, with only logarithmic overhead.

*Research conducted at Hewlett-Packard Cambridge Research Laboratory, One Cambridge Center, Cambridge, MA 02142.

†Computer Science Department, Johns Hopkins University, 3400 N. Charles St., Baltimore, MD 21218. Email: baruch@acm.org.

‡Dept. of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel. Email: boaz@eng.tau.ac.il.

§Dept. of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel. Email: david.peleg@weizmann.ac.il.

¶Cambridge Research Laboratory, HP Labs, One Cambridge Center, Cambridge, MA 02142. Email: mark.tuttle@hp.com.

1 Introduction

We consider an abstraction of the *recommendation problem* proposed by [4, 6], that can be described informally as follows. There are m users and n products. Given a product, a user can tell whether he considers it to be good. A recommendation algorithm tracks the choices of all users; in a basic step, the algorithm recommends a product to a user and asks what the user thought about the product. The task of the algorithm is to recommend a good product to most users, and its performance measure is the number of recommendations made to a user until a good product is recommended, as well as the number of users that are eventually satisfied. We believe that most readers have been exposed to commercial recommendation systems in one form or another (say, book recommendations in Amazon).

Obviously, recommendation systems (a.k.a. *collaborative filtering* systems) may only be useful to a user who shares preferences with others: a user with esoteric preferences cannot rely on others to help him find a product he likes. In [4], this property is captured as follows. On the outset, each user is assumed to have a grade for each product, so that each user is represented by his vector of grades. (The grades are revealed in an on-line fashion, but are assumed to exist always.) To model popular preferences, it is assumed that there exists a set of k "canonical" vectors (where k is a small constant), such that most user vectors are the result of adding random noise to one of the canonical vectors. Such a user is said to belong to a *dominant type*. The assumption that most users belong to a small number of dominant types appears to be a reasonable simplification; the algorithm aims at satisfying these users.

The recommendation system proposed in [4] is a centralized algorithm based on *matrix reconstruction* [2, 12]. The algorithm of [4], denoted *MR* henceforth, can be roughly described as follows. *MR* first chooses a small set of users and asks them to try *all* products and grade them. Based on these responses, each remaining user is asked to try a carefully chosen small set of products; the latter responses are used to associate users with types, which in turn are used to recommend products to users. It is shown that *MR* guaran-

tees (with high probability) that each user from a dominant type gets a good recommendation. The heart of *MR* is a low-rank approximation of the full user-product grade matrix (the low rank is k, the number of dominant types). This approximation is based on Singular Value Decomposition (SVD). The algorithm works only when a few severe conditions are met. First is the *type separability* assumption, which requires that the canonical vectors be nearly orthogonal; this means that two users from different dominant types essentially do not share any product they both like. Second is the *gap* assumption, which says that any non-dominant type is far less popular than any dominant type. To complicate things further, *MR* must know the number of dominant types in order to produce correct results. Lastly, the computational complexity of the algorithm is $O(mn)$.

In this paper we show, surprisingly, that the algebraic approach can be abandoned for this particular problem without any loss in the quality of the results, and while dramatically simplifying the algorithms. Moreover, both type separability and gap assumptions turn out to be unnecessary. Most important, we show that a distributed peer-to-peer solution is possible, even in the presence of a very powerful adversary.

Intuitively, the basic idea behind our algorithms is the usage of "best-seller lists" via random sampling. Our distributed algorithm is based on techniques developed in the context of auction systems like eBay [1].

We present the following results. First, we give a centralized algorithm that runs in time $O(m + n)$ assuming that most users fall within one of $k = O(1)$ large types (but neither type separability nor the existence of a gap are assumed). We note that $\Omega(n + m)$ is an obvious lower bound on the number of recommendations made by any recommendation algorithm even when all users are identical (i.e., $k = 1$): if only one of the products is good, then $\Omega(n)$ products must be recommended by the algorithm until that good product is discovered; and in addition, the algorithm must recommend to each of the m users at least one good product.

Our main result is a distributed solution to the recommendation problem. We show that under any asynchronous schedule, the total number of recommendations required to satisfy any set of users with shared interest is $O(n + m \log m)$ with high probability, coming within a logarithmic factor of the optimal. Our algorithm can tolerate malicious users in the form of an omniscient, adaptive, Byzantine adversary. (This is an important consideration for distributed peer-to-peer systems.) Another important feature of the algorithm is that it does not require users to be rigidly classified into types: users may have any preference vector whatsoever without changing the algorithm; only the performance will be affected, in the sense that users with esoteric preferences will have to work more.

We note that our algorithms imply that type separation is equivalent to full matrix reconstruction, but it is not required for competitive recommendations.

Related work. Most prior research on recommendation systems focused on a centralized, off-line version of the problem, where the algorithm is presented with a lot of historical preference data, and the task is to generate a single recommendation that maximizes the utility to the user. This is usually done by heuristically identifying clusters of users [10] (or products [11]) in the data set, and using past grades by users in a cluster to predict future grades by other users in the same cluster. SVD was shown also to be effective for the off-line problem [12]. Some of these systems enjoy industrial success, but they are known to perform poorly when prior data is less than plentiful [13], and they are extremely vulnerable even to mild attacks [7, 8]. Canny [3] gives a distributed secure and private SVD computation for the off-line version of the problem.

Theoretical studies of recommendation systems usually take the latent variable model approach: a stochastic process is assumed to generate noisy observations, and the goal of an algorithm is to approximate some unknown parameters of the model. Kumar *et al.* [6] study the off-line problem for a model where preferences are identified with past choices (purchases). In this model there are clusters of products. Each user has a probability distribution over clusters; a user first chooses a cluster by his distribution, and then chooses a product uniformly at random from that cluster. The goal is to recommended a product from the user's most preferred cluster. Kleinberg and Sandler [5] generalized this model to the case where the choice within a cluster is governed by an arbitrary probability distribution, and also consider the mixture model, in which each cluster is a probability distribution over all products. Azar *et al.* [2] consider a model where there exists an unknown user-product preference matrix which can be approximated by a low-rank matrix. The system observes this matrix only after its entries were subjected to random additive noise and then to random omissions. They use SVD to reconstruct the original preferences.

In this work we use the on-line model of recommendation systems, as proposed in [4], which is discussed above, and in further detail in Section 5. We note that measures and techniques used for off-line algorithms do not readily apply to the on-line scenario, since the off-line case ignores the crucial interaction between the algorithm and the user; this feedback necessarily skews the statistical properties of the observed data.

Paper organization. We define the recommendation

problem and our model in Section 2. In Sections 3 and 4 we give our centralized and distributed algorithms, respectively. In Section 5 we present a detailed discussion of our model in the context of previous models.

2 Formal Model

The recommendation problem. Let us begin by defining the recommendation problem. Assume the existence of a set $\mathcal{U}$ of *users* and a set $\mathcal{P}$ of *products*. Denote $|\mathcal{U}| = m$ and $|\mathcal{P}| = n$. Each user has a numeric grade for each product. In most of this paper, we will assume for the sake of simplicity that the grades are binary, i.e., each user views some products as *good* and the rest as *bad* (more on that in Section 5.1). We use $good(u)$ to denote a function mapping each user $u \in \mathcal{U}$ to the set of products u views as good.

A *recommendation algorithm* has a single primitive operation, in which it atomically outputs a product name to a user ("making a recommendation"), asks the user for his rating of that product (good or bad), and gets the user's response as input. When we say "u probes p" we mean that the algorithm recommends the product p to u and thus learns whether $p \in good(u)$. We define the set of products recommended to a user to be the algorithm's recommendation to that user. We want algorithms that recommend a small set of products to each user, and so that, with high probability, a large fraction λ of the users will find a good product in the set recommended to them.

The performance of an algorithm for the recommendation problem is evaluated in two ways. The traditional *time complexity* measures the worst-case number of RAM-model steps taken until the algorithm terminates. In addition, we measure the *recommendation complexity*, which is the total number of times users test recommended products. We further define the *individual recommendation complexity*, which is the worst-case number of products a single user tests. The recommendation complexity is obviously a lower bound on the time complexity, and our algorithms have essentially equal recommendation and time complexities.

The distributed model. We now turn to our model of a distributed system. Each user maintains a log of his probe results. An execution of the system consists of a sequence of user steps. In each atomic step, a user may send messages to other users asking them about their probe histories, get results back, and optionally probe a product (and keep the result in his local history record).[1] In contrast to the centralized model, where the algorithm can choose which user takes the next step, in the distributed model we assume that the order of user probes is under the control of an external *schedule*; the role of a distributed algorithm is limited to directing the user, as to which user to consult with and what product to probe during a step, when the user's turn comes up according to the schedule.

We assume that some users are *honest* and some are *dishonest*. Honest users are required to follow the protocol, including, in particular, answering questions truthfully. Dishonest users are allowed to behave in an arbitrary (Byzantine) fashion, including giving false reports on their probes and colluding.

Formally, an execution of a recommendation algorithm is uniquely determined by the algorithm, the sequence of coins flips, and by three external entities:

1. The *user schedule* that determines the order in which users take steps.
2. The sets of *dishonest users* and of *good products*.
3. The *adversary* that determines the behavior of the dishonest users.

The adversary we consider is the extremely powerful adaptive Byzantine adversary. Formally, an adversary is a function that takes the set of dishonest users, the set of goodness predicates, the schedule and a random binary sequence, and maps them to a sequence of actions for the dishonest users, telling them exactly what to do in each step. The random binary sequence represents the entire sequence of coin flip outcomes during the execution, including even future coin flips. Note that with this information, the adversary can reconstruct the entire state of the system at any point in time, and use any of this information to choose the next move for each dishonest user.

An *operating environment* consists of a user schedule, a set of dishonest users, a set of goodness predicates, and an adversary. The purpose of the operating environment is to factor out all of the nondeterministic choices made during an execution, leaving only the probabilistic choices to consider. When we deal with probabilities or compute expected values, we fix an operating environment and consider the distribution of executions with this operating environment induced by the coin flips. We note that the user schedule may schedule steps by a user long after the user has found a good product and halted, in which case the user just skips its step.

Each execution of the system yields a *probe sequence* consisting of the sequence of products probed during the execution. When we compute the recommendation complexity of a probe sequence, we count only the number of probes by the honest users, and ignore the probes by the dishonest users. Our goal is to minimize the recommendation complexity for the honest users.

[1] We remark that in our distributed algorithm, only a single user is consulted at a step, and a valid response consists of at most one product, so communication complexity is kept at minimum.

Algorithm $\mathcal{A}_{\text{cntr}}(k, \gamma)$
Let $K = k \ln(k/\gamma)$.
Select K users a random. Call these users the *committee*.
Let each committee member u probe all n products, and let p_u be the product u views as best (break ties arbitrarily).
Output $\{p_1, \ldots, p_K\}$ as the set of recommended products.

Figure 1: The centralized algorithm $\mathcal{A}_{\text{cntr}}(k, \gamma)$ for integer $k \geq 0$ and error $\gamma > 0$.

3 Centralized algorithm

This section presents a centralized solution to the recommendation problem. The algorithm is a simplification of the algorithm in [4], but it does not make the "gap" and "type separability" assumptions (see below). In this section we assume that all users are honest; dishonest users are considered in the next section.

Figure 1 gives the centralized algorithm that takes as parameters an integer $k \geq 0$ and a real number $0 < \gamma \leq 1$ and outputs recommendations for each user. The algorithm first selects a random sample of $k \ln(k/\gamma)$ users called the "committee," and recommends all n products to each committee member. Each committee member chooses the product he liked best, and these $k \ln(k/\gamma)$ are recommended to all remaining users.[2]

To analyze the algorithm we assume that users can be partitioned into equivalence classes called *types*, where each equivalence class represents a set of users with similar preferences. The success of the algorithm depends on the abundance of large types. Formally, we have the following definitions.

DEFINITION 3.1. *A* type T *is a set of users* $U(T)$ *and a nonempty set of products* $P(T)$ *satisfying the condition that the set of good products for a user in* $U(T)$ *is exactly* $P(T)$*, namely,*

$$U(T) \subseteq \{u \in \mathcal{U} \mid P(T) = \mathit{good}(u)\}.$$

Given a collection $\mathcal{T}$ *of types, we write* $U(\mathcal{T}) \stackrel{\text{def}}{=} \bigcup_{T \in \mathcal{T}} U(T)$.

DEFINITION 3.2. *Let* $0 < \lambda \leq 1$ *be a real number, and let* $k > 0$ *be an integer. A collection* $\mathcal{T}$ *of types is a* (λ, k)-type-cover *if* $|U(\mathcal{T})| \geq \lambda m$ *and* $|U(T)| \geq m/k$ *for each* $T \in \mathcal{T}$.

[2]The algorithm works as stated for arbitrary numeric grades, as well as for binary values.

Let us highlight a few properties of our definition of type cover. First, the product sets of different types in a type cover may have arbitrary intersection. This is in contrast with the "type separability" assumption [4], which requires the product sets to be nearly disjoint. This is quite a severe restriction: for example, if preference vectors include a product that all users like (say, motherhood and apple pie), then type separation would allow only one vector to be included in a type cover. Second, types in a cover must have a minimum number of users, but nothing is required of types not in the cover. This in contrast to the "gap" property required by [4], that stipulates that the number of users of each type in the cover is much larger than the number of users in each type not in the cover. Finally, we note that the definition of a type implies that if T and T' are types with different product sets, then $U(T) \cap U(T') = \emptyset$, i.e., they have disjoint user sets. This restriction will be lifted in the next section.

We now characterize the performance of $\mathcal{A}_{\text{cntr}}(k, \gamma)$ in the context of a (λ, k)-type-cover:

THEOREM 3.1. *Let* $\gamma > 0$. *If there is a* (λ, k)*-type-cover for some* $\lambda > 0$ *and* $k > 0$*, then with probability at least* $1 - \gamma$ *the algorithm* $\mathcal{A}_{cntr}(k, \gamma)$ *generates a set of recommendations that satisfies* λm *users with recommendation and time complexity* $O(k(m+n) \log \frac{k}{\gamma})$.

Proof: Let $\mathcal{T}$ be a (λ, k)-type-cover. For a type $T \in \mathcal{T}$, the users of $U(T)$ will find a good product in the set recommended by Algorithm $\mathcal{A}_{\text{cntr}}$ if some user of $U(T)$ has been chosen to the committee. Each user of $U(T)$ is chosen to the committee with probability $|U(T)|/m \geq 1/k$, since $T \in \mathcal{T}$ and $\mathcal{T}$ is a (λ, k)-cover. Hence the probability that no user of $U(T)$ has been chosen to the committee is at most

$$\left(1 - \frac{1}{k}\right)^K = \left(1 - \frac{1}{k}\right)^{k \ln(k/\gamma)} \leq e^{-\ln k/\gamma} = \gamma/k \, .$$

Hence, the probability that the users of *some* type $T \in \mathcal{T}$ fail to find a good product in the set recommended by $\mathcal{A}_{\text{cntr}}$ is at most $k(\gamma/k) = \gamma$. The theorem now follows from the fact that $|U(\mathcal{T})| \geq \lambda m$. ∎

Intuitively, each user in the type cover has, with high probability, a "representative" user from his type in the committee, whose job is to discover the product his "constituent" users like; this representative reports his findings to the benefit of all others. The reader may note that in some sense, committee members play a role similar to that of critics in human society.

Theorem 3.1 bounds the probability that a user finds a good product after testing all $K = k \ln(k/\gamma)$ recommendations. It may be interesting also to understand

how many products the user should test *on average* until it finds that good product. We assume that the order in which non-committee users try recommendations is a random permutation.

THEOREM 3.2. *Let $\gamma > 0$. If there is a (λ, k)-type-cover $\mathcal{T}$ for some $\lambda > 0$ and $k > 0$, then the expected individual recommendation complexity for users in $\mathcal{T}$ is $O(k(1 + \frac{n \log(k/\gamma)}{m}))$.*

Proof: A random user u is chosen for the committee with probability $\frac{K}{m} = O(\frac{k \log(k/\gamma)}{m})$, in which case it tests all n products, accounting for the second term in the bound. Now suppose that $u \in U(\mathcal{T})$ is a non-committee member. In this case u only tests products until it finds a product it likes, or until all K recommendations are exhausted. Consider the ith recommendation it tests. Since committee members are picked at random, and since u picked a random recommendation as its ith product, we have that u follows the recommendation of a random user. Let T_u be the type of u. Since $T_u \in \mathcal{T}$, the probability that a random user is of type T_u is $|U(T_u)|/m \geq 1/k$. Since the ith recommendation is good for u if the ith committee member is of type T_u, we get that the probability that the first good recommendation is the ith one is at least $(1 - 1/k)^{i-1} \cdot (1/k)$. The probability that u does not find a good recommendation after trying all K of them is $(1 - 1/k)^K$. Therefore, the expected number φ_u of products u tests is bounded by

$$\begin{aligned}
\mathbf{E}[\varphi_u] &\leq \sum_{i=1}^{K} i \frac{1}{k} \left(1 - \frac{1}{k}\right)^{i-1} + K \left(1 - \frac{1}{k}\right)^{K} \\
&\leq k + K \left(1 - \frac{1}{k}\right)^{k \ln(k/\gamma)} \\
&\leq k + K e^{-\ln(k/\gamma)} \\
&= k + \frac{k \ln(k/\gamma)}{k/\gamma} = O(k) . \qquad \blacksquare
\end{aligned}$$

A detailed comparison of these results with those of [4] appears in Section 5.

Critique of Committee Algorithms. The idea of committee, used in Algorithm $\mathcal{A}_{\text{cntr}}$, has some serious disadvantages. First, the individual recommendation complexity of $\mathcal{A}_{\text{cntr}}$ is $\Omega(n)$, since a committee member must test all products to make a recommendation. This is not merely a formal objection, but rather a symptom of the real difficulty in implementing a committee-based algorithm (in [4] it was proposed to compensate committee members for their efforts).

The second point is manifest in a distributed implementation with malicious users. In this case, even choosing a committee is a non-trivial task: how will membership be determined? If users decide on their own whether they are committee members, many dishonest users might "volunteer" into the committee and effectively hide the honest members. One possible workaround for this problem is a random beacon [9] that generates public coin flips. But even if we had a random beacon at our disposal, any committee-based algorithm is vulnerable to *adaptive* Byzantine attacks that target committee members. This is a very practical threat: one real-world adaptive Byzantine adversary is the one who thwarts the algorithm by bribing committee members.

Subsequently, in the following section we present a distributed algorithm that does not use a committee and does not suffer from these problems.

Algorithm $\mathcal{A}_{\text{dist}}$ followed by each user
repeat
Flip a coin.
If the result is "heads," select a product uniformly at random and probe it.
If the result is "tails," select a user uniformly at random and probe the product that user recommends.
until a good product is found.
Recommend the good product and halt.

Figure 2: The distributed algorithm $\mathcal{A}_{\text{dist}}$.

4 Distributed Algorithm

This section presents a distributed solution to the recommendation problem that is resilient to arbitrarily malicious behavior from any fraction of the users. This algorithm shows that it is possible to do away with the committee altogether in a solution to the recommendation problem. Even more interestingly, the correctness of the algorithm does not depend on the existence of any kind of cover. The analysis of the algorithm's recommendation complexity does depend on covers, but it uses a significantly more relaxed notion of a cover by "special interest groups," and not type covers.

4.1 Algorithm. The algorithm $\mathcal{A}_{\text{dist}}$ is very simple (see Figure 2): each honest user repeatedly either chooses a random product and probes it, or chooses a random user and probes the product that user recommends. This is done by sending a message to that user, to which the consulted user responds with the identity of the best product he's probed so far (ties broken arbitrarily). When a user finds a good product, the user is said to be *satisfied* and he stops running the algorithm.

4.2 Analysis of recommendation complexity. The basic property of $\mathcal{A}_{\text{dist}}$ is stated in the following theorem, which bounds the total recommendation complexity of any set of users with a shared good product under any asynchronous schedule and any adversary.

THEOREM 4.1. *Let U be a nonempty set of honest users and P be a nonempty set of products such that $P \subseteq good(u)$ for every $u \in U$. For every operating environment, the expected recommendation complexity for users in U in an execution of Algorithm $\mathcal{A}_{dist}$ is at most $2\,(n/|P| + m \ln |U|)$.*

Proof: To prove that the bound on expected recommendation complexity holds for every operating environment, let us fix an arbitrary environment $\mathcal{E}$. Let σ be the random variable whose value is the sequence of probes by users in U during an execution of $\mathcal{A}_{\text{dist}}$ in the context of $\mathcal{E}$. Let $\sigma = \sigma_0\sigma_1 \cdots \sigma_{|U|}$ where σ_ℓ is the random variable whose value is the subsequence of probes preceded by *exactly* ℓ probes by users in U of products in P. In other words, exactly ℓ users in U have found a product in P at the start of σ_ℓ, and the final probe of σ_ℓ is by the $(\ell+1)$st user in U finding a product in P. Notice that some users in U may be satisfied by products outside of P, meaning that only a subset of users in U actually probe a product in P, in which case some suffix $\sigma_j\sigma_{j+1} \cdots \sigma_{|U|}$ of σ may actually be empty. Since users halt once they find a product in P, we can bound the recommendation complexity by counting the number of probes by users in U to products not in P. Clearly $count(\sigma) = \sum_{\ell=0}^{|U|} count(\sigma_\ell)$ is an upper bound on the total number of products tested by users in U as they search for a good product.

Consider the cost of σ_0. With probability $1/2$ the algorithm chooses a random product to sample. Since P is nonempty and there are n products, each probe in σ_0 probes a product in P with probability at least $|P|/2n$, and hence $\mathbf{E}\,[count(\sigma_0)] \leq 2n/|P|$. Now consider the cost of σ_ℓ for $\ell > 0$. With probability $1/2$ the algorithm chooses a user and samples its recommendation. Since exactly ℓ users in U have found a product in P when σ_ℓ begins, each probe in σ_ℓ finds a product in P with probability at least $\ell/2m$, so the expected number of probes in σ_ℓ is $2m/\ell$. Thus,

$$\begin{aligned}
\mathbf{E}\,[count(\sigma)] &= \mathbf{E}\,[count(\sigma_0)] + \sum_{\ell=1}^{|U|} \mathbf{E}\,[count(\sigma_\ell)] \\
&\leq \frac{2n}{|P|} + \sum_{\ell=1}^{|U|} \frac{2m}{\ell} \\
&= 2\left(\frac{n}{|P|} + m \ln |U|\right). \qquad \blacksquare
\end{aligned}$$

Intuitively, the analysis divides the execution into two stages: in the first stage, the users search for a good object until one of them finds it due to the "random product" probes; in the second stage, the identity of that product is spread among the users by the epidemic-style "random recommendation" probes. It may be interesting to note that in a "steady state," when many users have halted, the algorithm gives a concrete distributed implementation of best-seller lists by randomly sampling the users.

Theorem 4.1 holds unconditionally, but gives a good bound only for a set of users with a common interest. A good bound for most users can be obtained under an assumption similar to that made in Theorem 3.2, namely, the existence of (λ, k)-type-cover. However, Theorem 4.1 indicates that we can use a much weaker notion, referred to as a *special interest group* or *SIG*. Intuitively, while users of the same type must have exactly the same preferences, users in the same SIG need only have a non-empty intersection of preferences. Formally, we have the following definition.

DEFINITION 4.1. *A* special interest group (SIG) *S is a set of honest users $U(S)$ and a nonempty set of products $P(S)$ satisfying the condition that every product in $P(S)$ is good for every user in $U(S)$, namely,*

$$U(S) \subseteq \{u \in \mathcal{U} \mid P(S) \subseteq good(u)\}.$$

Given a collection $\mathcal{S}$ of special interest groups we write $U(\mathcal{S}) \stackrel{\text{def}}{=} \bigcup_{S\in\mathcal{S}} U(S)$.

A special interest group S, therefore, is a set of users $U(S)$ that would be satisfied by any product in $P(S)$ (and possibly by other products as well). The set of products $P(S)$ represents a common or special interest of the users in $U(S)$.

Using the concept of SIGs, we generalize Theorem 4.1 as follows.

COROLLARY 4.1. *Let $\mathcal{S} = \{S_1, \ldots, S_\ell\}$ be any collection of SIGs, and denote*

$$\begin{aligned}
\tilde{m}(\mathcal{S}) &= \max\{|U(S_1)|, \ldots, |U(S_\ell)|\}\ , \\
\tilde{n}(\mathcal{S}) &= \min\{|P(S_1)|, \ldots, |P(S_\ell)|\}\ .
\end{aligned}$$

The expected recommendation complexity for the users in $U(\mathcal{S})$ is at most

$$2\ell\left(\frac{n}{\tilde{n}(\mathcal{S})} + m \ln \tilde{m}(\mathcal{S})\right) \leq 2\ell(n + m \ln m).$$

Now we can compare $\mathcal{A}_{\text{dist}}$ to $\mathcal{A}_{\text{cntr}}$: suppose, in the spirit of Theorem 3.2, that there exists a small collection of SIGs that cover most users (i.e., $\ell = O(1)$ in

Corollary 4.1, and $\frac{|U(S)|}{m} \geq 1-\lambda$ for a small λ). Then the expected total recommendation complexity over most users is just $O(n+m\ln m)$ by Corollary 4.1, as opposed to $O(n+m)$ in Theorem 3.2. However, $\mathcal{A}_{\text{dist}}$ improves on $\mathcal{A}_{\text{cntr}}$ in that it tolerates malicious users, and in that SIGs are used, and only to bound performance (rather than requiring types, and to guarantee correctness).

4.3 Individual recommendation complexity in the synchronous model. Corollary 4.1 gives the expected recommendation complexity for the algorithm $\mathcal{A}_{\text{dist}}$, but what can we say about the individual recommendation complexity? Clearly the number of products a user must probe to find a good one depends how many other users from the same SIG are probing. If the schedule is such that a user is running alone, it will have to carry out the entire search on its own. If a user is running concurrently with many others from its SIG, it can expect the search to be distributed over all of these users, reducing the number of products it has to probe itself. To get a handle on individual costs, we study individual recommendation complexity in a *synchronous model* where an execution proceeds in a sequence of rounds. In each round, users first send messages to other users, then get replies, and finally probe a product. This model is different from the one obtained by restricting the asynchronous model to round robin schedules: round robin schedules consist of a sequence of atomic send-receive-probe steps, whereas the synchronous rounds entail concurrent actions, and therefore require a little more effort to analyze.

Our main result for this model is stated below.

THEOREM 4.2. *Let S be any special interest group. If $|U(S)| \geq \Omega(\log m)$, then with probability at least $1-m^{-\Omega(1)}$, at least half of the users in $U(S)$ have found a good product by time $O\left(\frac{\log m}{|U(S)|}\cdot\left(\frac{n}{|P(S)|}+m\right)\right)$.*

Proof: Define $\alpha_S \stackrel{\text{def}}{=} \frac{|U(S)|}{m}$ and $\beta_S \stackrel{\text{def}}{=} \frac{|P(S)|}{n}$, i.e., α_S and β_S are the fractions of users and products, respectively. Fix a constant $c \geq 2$, and assume that $\alpha_S > 32c\log m/m$. Denote by $\rho(t)$ the number of recommendations by users in $U(S)$ for products in $P(S)$ after t rounds. Any execution of the algorithm can be viewed as having $q = \lceil \log \frac{|U(S)|}{4c\log m} \rceil$ phases, defined as follows. Let $f_1 = 1$ and $f_k = 2^k c\log m$ for $2 \leq k \leq q-1$.

1. For $1 \leq k \leq q-1$, Phase i ends as soon as either half the users in $U(S)$ are satisfied or $\rho(t) \geq f_k$.
2. Phase q ends when at least half the users in $U(S)$ are satisfied.

(Note that the two events defining the end of a phase are not necessarily ordered, since generally, users in $U(S)$ can be satisfied by products other than those in $P(S)$ and similarly, products in $P(S)$ can be recommended by users other than those in $U(S)$.)

For $1 \leq k \leq q$, let t_k be a random variable counting the number of rounds of Phase k. Let $r_1 = \left\lceil 4c\frac{\ln m}{|U(S)|\beta_S}\right\rceil$, $r_2 = \left\lceil \frac{64c\log m}{\alpha_S}\right\rceil$, $r_k = \left\lceil \frac{16}{\alpha_S}\right\rceil$ for $3 \leq k \leq q-1$ and $r_q = \left\lceil \frac{16(c+1)\log m}{\alpha_S}\right\rceil$. Let $\mathcal{E}_k$ denote the event that $t_k > r_k$.

LEMMA 4.1. $\mathbf{P}[\mathcal{E}_k] \leq m^{-c}$ *for every* $1 \leq k \leq q$.

Proof: We prove the lemma for each phase separately.

Phase 1: In every round of Phase 1, each unsatisfied user probes a product in $P(S)$ with probability at least $\frac{1}{2}\frac{|P(S)|}{n} = \frac{\beta_S}{2}$. By definition of Phase 1, at least half of the users in $U(S)$ are unsatisfied throughout the phase. Hence $\mathbf{P}[\mathcal{E}_1]$ is bounded from above by the probability that none of these users probes a product in $P(S)$ in any of the first r_1 rounds, i.e.,

$$\begin{aligned}\mathbf{P}[\mathcal{E}_1] &\leq \left(1-\frac{\beta_S}{2}\right)^{|U(S)|r_1/2} \\ &\leq \exp(-|U(S)|\beta_S r_1/4) \\ &= \exp\left(-\frac{|U(S)|\beta_S}{4}\cdot\frac{4c\ln m}{|U(S)|\beta_S}\right) = m^{-c},\end{aligned}$$

implying the lemma for Phase 1.

Phase 2: If Phase 1 ends when half the users in $U(S)$ are satisfied then the ends of Phase 1 and 2 coincide and there is nothing to prove. So suppose Phase 2 begins with at least one recommendation for a product in $P(S)$, but fewer than half the users in $U(S)$ are satisfied. In this case, each unsatisfied user u samples a product of $P(S)$ with probability at least $\frac{1}{2m}$ in every round of the phase. Hence u finds a product of $P(S)$ in the first r_k rounds of Phase 2 with probability at least $1-p$ for $p = (1-\frac{1}{2m})^{r_2}$. Note that

$$\begin{aligned}p &\leq \left(1-\frac{1}{2m}\right)^{\frac{64c\log m}{\alpha_S}} \\ &\leq \exp\left(-\frac{32c\log m}{|U(S)|}\right) \leq 1-\frac{16c\log m}{|U(S)|},\end{aligned}$$

where the last inequality relies on the fact that $\exp(-x) \leq 1-x/2$ for $0 \leq x \leq 1$ and $\alpha_S > 32c\log m/m$. By definition of the phase, at least half of the users in $U(S)$ are unsatisfied throughout the phase. Let H_k denote the random variable counting the number of users of $U(S)$ that find a product of $P(S)$ during the first r_k rounds of Phase k. It follows that the expected value

of H_2 satisfies

$$\begin{aligned}\mathbf{E}[H_2] &\geq \frac{|U(S)|}{2}(1-p) \\ &\geq \frac{|U(S)|}{2}\cdot\frac{16c\log m}{|U(S)|} = 8c\log m\ .\end{aligned}$$

By Chernoff's bound,

$$\begin{aligned}\mathbf{P}[\mathcal{E}_2] &\leq \mathbf{P}[H_2 < 4c\log m] \\ &\leq \exp\left(-\frac{1}{2}\cdot\frac{1}{4}\cdot 8c\log m\right) = m^{-c}\ ,\end{aligned}$$

implying the lemma for Phase 2.

Phase k for $3 \leq k \leq q-1$: Again, if Phase $k-1$ ends when half the users in $U(S)$ are satisfied then there is nothing to prove, so suppose Phase k begins with at least f_{k-1} recommendation for a product in $P(S)$, but fewer than half the users in $U(S)$ are satisfied.

Assuming Phase k begins with at least f_{k-1} recommendations for products in $P(S)$, each unsatisfied user u samples a product of $P(S)$ with probability at least $\frac{f_{k-1}}{2m}$ in every round, hence u finds a product of $P(S)$ in the first r_k rounds of Phase k with probability at least $1-p$ for $p = \left(1-\frac{f_{k-1}}{2m}\right)^{r_k}$. Note that

$$p \leq \left(1-\frac{f_{k-1}}{2m}\right)^{16/\alpha_S} \leq \exp\left(-\frac{8f_{k-1}}{|U(S)|}\right) \leq 1-\frac{4f_{k-1}}{|U(S)|}\ ,$$

where the last inequality relies on the fact that $\exp(-x) \leq 1-x/2$ for $0 \leq x \leq 1$ and $f_{k-1} \leq |U(S)|/8$ for $k \leq q-1$. By definition of the phase, there are at least $|U(S)|/2$ unsatisfied users throughout it. It follows that the expected value of H_k satisfies

$$\mathbf{E}[H_k] \geq \frac{|U(S)|}{2}(1-p) \geq \frac{|U(S)|}{2}\cdot\frac{4f_{k-1}}{|U(S)|} = 2f_{k-1}\ .$$

By the Chernoff bound,

$$\begin{aligned}\mathbf{P}[H_k < f_{k-1}] &\leq \exp\left(-\frac{1}{2}\cdot\frac{1}{4}\cdot 2f_{k-1}\right) \\ &\leq \exp\left(-\frac{1}{8}\cdot 8c\log m\right) = m^{-c}\ .\end{aligned}$$

As Phase k begins with at least f_{k-1} recommendations for products in $P(S)$, we have that if $H_k \geq f_{k-1}$ then after r_k rounds of Phase k, the number of such recommendations is at least $f_{k-1}+H_k \geq 2f_{k-1} = f_k$, implying $\neg\mathcal{E}_k$. Hence $\mathbf{P}[\mathcal{E}_k] \leq m^{-c}$.

Phase q: Again, if Phase $q-1$ ends when half the users in $U(S)$ are satisfied then we are done, so consider the case when Phase q begins with at least f_{q-1} recommendation for a product in $P(S)$, but fewer than half the users in $U(S)$ are satisfied.

Assuming Phase q begins with at least

$$f_{q-1} = 2^{q-1}c\log m \geq c\log m\cdot\frac{|U(S)|}{8c\log m} = \frac{|U(S)|}{8}$$

recommendations for products in $P(S)$, each unsatisfied user u samples a product of $P(S)$ with probability at least $\frac{f_{q-1}}{8m}$ in every round, hence u fails to find a product of $P(S)$ in the first r_q rounds of Phase q with probability at most

$$\begin{aligned}\left(1-\frac{f_{q-1}}{8m}\right)^{r_q} &\leq \exp\left(-\frac{f_{q-1}\cdot r_q}{8m}\right) \\ &\leq \exp\left(-\frac{\frac{|U(S)|}{8}\cdot\frac{16(c+1)\log m}{\alpha_S}}{2m}\right) \\ &= \exp(-(c+1)\log m) = m^{-c-1}\ .\end{aligned}$$

Thus the probability that not all users of $U(S)$ will be satisfied after r_q rounds is at most m^{-c}. ∎

The theorem now follows by combining the claims of the lemma for the separate phases and concluding that the probability that any of the events $\mathcal{E}_i$ has occurred, for $1 \leq i \leq q+1$, is at most $(q+1)m^{-c} \leq m^{1-c}$. Hence with probability at least $1-m^{1-c}$, the total number of rounds until at least half the users of $U(S)$ were satisfied was bounded by

$$\begin{aligned}\sum_{k=1}^{q} r_k &\leq \left\lceil 4c\frac{\ln m}{|U(S)|\beta_S}\right\rceil + \left\lceil\frac{64c\log m}{\alpha_S}\right\rceil \\ &\quad + (q-3)\left(\frac{16}{\alpha_S}+1\right) + \left\lceil\frac{16(c+1)\log m}{\alpha_S}\right\rceil \\ &= O\left(\log m\left(\frac{1}{|U(S)|\beta_S}+\frac{1}{\alpha_S}\right)\right)\ ,\end{aligned}$$

as $q = O(\log m)$. ∎

Note that the theorem does not guarantee that all users in the SIG will find a good product quickly: this is because some of the users may be satisfied with products not in $P(S)$. The following lemma says that if sufficiently many users of $U(S)$ actually recommend products from $P(S)$, then *all* users in $U(S)$ will be done quickly.

LEMMA 4.2. *Let S be any special interest group, and suppose that at some time t, there are at least $|U(S)|/2$ recommendations for products in $P(S)$. Then with probability at least $1-m^{-\Omega(1)}$, all users in $U(S)$ will find a good product by time $t+O(\frac{m\log m}{|U(S)|})$.*

Proof: Let $c > 4$. We show that with probability at least $1-m^{1-c/4}$, all users in $U(S)$ will find a good

product by time $t + \frac{c \ln m}{\alpha_S}$. Consider any user $u \in U(S)$ that has not found a good product by time t. Let p_u denote the probability that u does not find a good product within $\frac{c \ln m}{\alpha_S}$ rounds after time t. In each round after time t, u follows the recommendation of a random user with probability 1/2. By assumption, this user recommends a product from $P(S)$ with probability at least $\frac{|U(S)|/2}{m} = \alpha_S/2$. Hence in each round after time t, u will try a product from $P(S)$ with probability at least $\frac{1}{2} \cdot \frac{\alpha_S}{2} = \frac{\alpha_S}{4}$. It follows that

$$p_u \leq \left(1 - \frac{\alpha_S}{4}\right)^{(c \ln m)/\alpha_S} \leq e^{-(c \ln m)/4} = m^{-c/4} .$$

The result follows, since the probability that not all users in $U(S)$ have finished by time $t + \frac{c \ln m}{\alpha_S}$ is at most $p_u |U(S)| \leq p_u m$. ∎

Theorem 4.2 and Lemma 4.2 yield the following implication for the special case of types.

COROLLARY 4.2. *Let T be any type. If $|U(T)| \geq \Omega(\log m)$, then with probability at least $1 - m^{-\Omega(1)}$, all users in $U(T)$ have found a good product by time $O\left(\frac{\log m}{|U(S)|} \cdot \left(\frac{n}{|P(S)|} + m\right)\right)$.*

Proof: By Theorem 4.2, with probability at least $1 - m^{-\Omega(1)}$ there exists some time $t_1 = O(\log m(\frac{1}{|U(T)|\beta_T} + \frac{1}{\alpha_T}))$ by which at least half of the users of $U(T)$ have found a good product. Since T is a type, the good products all these users found are in $P(T)$. The result hence follows from Lemma 4.2. ∎

We note that one can use either Theorem 4.2 or Corollary 4.2 to bound the individual recommendation complexity of users. It is clear that we can always partition the set $\mathcal{U}$ of users into equivalence classes where two users u and v are in the same equivalence class iff $good(u) = good(v)$. Each equivalence class naturally gives rise to a type T where $U(T)$ is the set of users in the class and $P(T)$ is the set of products they consider good. In this sense, we can always find a type T containing u, and use T and Corollary 4.2 to bound the recommendation complexity for u. However, $|U(T)|$ might be small. If there is a large SIG containing u, then Theorem 4.2 allows us to get a tighter bound on the individual recommendation complexity for at least half of the members of that SIG.

5 Discussion of results

In this section we compare the models used by the *MR* algorithm in [4] and our centralized model as describes in Section 2. Since our model is a streamlined version of the *MR* model, some (straightforward) analysis is required to show that up to constant factors, our model is no weaker than the *MR* model.

5.1 User preferences. The *MR* model uses real values to model user preferences for products, whereas we use binary values. We argue that using binary values is simpler and does not restrict generality.

Specifically, the *MR* model assumes that there is a *preference vector* A_u for each user u, a vector of nonnegative real values such that the pth entry $A_{u,p}$ in the vector is the *preference value* of product p in the eyes of u. For each user u, the model defines a threshold θ_u in terms of A_u, and defines a product p to be *good* for u if $A_{u,p} > \theta_u$. The model defines a *recommendation* to be a small set of products, and a recommendation is *good* for u if it contains a product that is good for u.

In contrast, we use binary preference values to products, saying that u has preference 1 for p if $p \in good(u)$ and 0 otherwise. This is justified simply because our algorithms work for arbitrary numeric values just as well. (As an aside, it appears that binary data is often seems easier to collect, say by interpreting a clicking on a product as endorsing it.) It may be interesting to consider translating the *MR* model to the binary model by replacing real preferences with binary preferences by defining $good(u)$ to be the products with real preferences above θ_u. This approach gives rise to the more important question: can a user u decide on-line if $p \in good(u)$? The implication of the existence of an on-line test (say, if the threshold value θ_u is known to u), is that users can stop as soon as they find a good product. If no on-line test is available, then the best one can hope for from an algorithm is to prove that if the users test enough of the recommendations made by the algorithm, they will be satisfied. Our distributed algorithm is written assuming that an on-line test is available (thus allowing early stopping), but can easily be adapted to the case where no such test is available.

5.2 Covers and effectiveness. The analysis of the *MR* algorithm uses a concept called (λ, k)-effectiveness. A set of users is said to be (λ, k)-*effective* for some real $0 \leq \lambda \leq 1$ and integer $k \geq 0$ if there exists a subset of at least λm users that belong to at most k types. In the analysis of Algorithm $\mathcal{A}_{\text{cntr}}$, we use a dual concept of (λ, k)-type-covers. The following lemma shows that the concepts of user effectiveness and covers are equivalent up to constant factors.

LEMMA 5.1. *Let $\mathcal{U}$ be a set of users and $good(u)$ be the good products for a each user $u \in \mathcal{U}$.*

(1) If there exists a (λ, k)-type-cover for $\mathcal{U}$, then $\mathcal{U}$ is (λ', k')-effective for $\lambda' = \lambda$ and $k' = \lceil \lambda k \rceil$.

(2) If $\mathcal{U}$ is (λ, k)-effective, then for any $0 < \epsilon \leq 1$, $\mathcal{U}$ has a (λ', k')-type-cover for $\lambda' = (1 - \epsilon)\lambda$ and $k' = k/(\lambda\epsilon)$.

Proof: Suppose $\mathcal{U}$ has a (λ, k)-type-cover $\mathcal{T}'$. To show that $\mathcal{U}$ is $(\lambda, \lceil \lambda k \rceil)$-effective, consider any subset $\mathcal{T}$ of $\mathcal{T}'$ such that $|\mathcal{T}| = \lceil \lambda k \rceil$. It is sufficient to show that $|U(\mathcal{T})| \geq \lambda m$. This is true because by definition, $|U(T)| \geq \frac{m}{k}$ for each $T \in \mathcal{T}'$, and $U(T) \cap U(T') = \emptyset$ for any two distinct $T, T' \in \mathcal{T}'$. Therefore $|U(\mathcal{T})| \geq \lceil \lambda k \rceil \frac{m}{k} \geq \lambda m$.

Conversely, suppose $\mathcal{U}$ is (λ, k)-effective, and fix $0 \leq \epsilon < 1$. Let $\mathcal{T} = \{T_1, \ldots, T_k\}$ be the type collection indicated by the (λ, k)-effectiveness assumption. Then $|U(\mathcal{T})| \geq \lambda m$. Define $\mathcal{T}_\epsilon = \{T \in \mathcal{T} \mid |U(T)| \geq \epsilon \lambda m / k\}$. We claim that $\mathcal{T}_\epsilon$ is a $((1-\epsilon)\lambda, k/(\lambda\epsilon))$-type-cover. By definition, each type in $\mathcal{T}_\epsilon$ has at least $\lambda \epsilon m / k$ users, hence the second cover requirement follows. To see that also $|U(\mathcal{T}_\epsilon)|$ is as required, define $\mathcal{T}'_\epsilon = \mathcal{T} \setminus \mathcal{T}_\epsilon$. By definition, $|U(T)| < \epsilon \lambda m / k$ for each $T \in \mathcal{T}'_\epsilon$. This yields the required size bound, since

$$|U(\mathcal{T}_\epsilon)| \;=\; |U(\mathcal{T} \setminus \mathcal{T}'_\epsilon)| \;\geq\; \lambda m - k \cdot \frac{\epsilon \lambda m}{k} \;=\; (1-\epsilon)\lambda m \,. \quad \blacksquare$$

Casting the results of Theorems 3.1 and 3.2 in the terminology of [4] and applying Lemma 5.1, we get the following:

COROLLARY 5.1. *If the users are (λ', k')-effective for some $\lambda' > 0$ and $k' > 0$, then for all $\epsilon > 0$ Theorems 3.1 and 3.2 hold with $\lambda = (1-\epsilon)\lambda'$ and $k = k'/(\lambda'\epsilon)$.*

5.3 Type deviation. An important aspect of the *MR* model that we abstract is type deviation. Specifically, in the *MR* model, it is assumed that the preference vector of each user is a random variable, obtained by adding to the user's type vector an "error vector." The error vector is assumed to consist of n independent 0-mean random variables with variance $\sigma^2 = \frac{\epsilon^2}{m+n}$ for some small $0 < \epsilon < 1$.

Assuming such a small magnitude of variance of the errors, it is easy to justify our abstraction: This assumption implies that for each user and any $p > 0$, with probability $1 - p$, all preferences are at most $\delta \stackrel{\text{def}}{=} \epsilon / \sqrt{p}$ away from the type vector. This is because Chebychev's Inequality says that the probability that a user's preference for any specific product differs from the canonical preference by more than δ is less than

$$\frac{\sigma^2}{\delta^2} = \frac{\epsilon^2}{(m+n)\delta^2} \leq \frac{\epsilon^2}{n\delta^2} = \frac{\epsilon^2 p}{n\epsilon^2} = \frac{p}{n}.$$

It follows that the probability that *all* the preferences of a user differ from the type vector by *at most* δ is at least $\left(1 - \frac{p}{n}\right)^n \geq 1 - p$. Hence, for any given $p > 0$ we can define a goodness predicate using a threshold θ so that all but a fraction of p of the users have preferences that agree with all other users of their type.

As an aside, we note that SIGs may be viewed as another (deterministic) abstraction of type deviation, that allows for much stronger noise models. We defer discussion of this aspect to the full paper.

Acknowledgment. The authors wish to thank Prabhakar Raghavan for his helpful comments.

References

[1] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. Tuttle. Collaboration of untrusting peers with changing interests. In *Proc. 5th ACM Conf. on Electronic Commerce (EC)*, pages 112–119, May 2004.

[2] Y. Azar, A. Fiat, A. Karlin, F. McSherry, and J. Saia. Spectral analysis of data. In *Proc. 33rd ACM Symp. on Theory of Computing (STOC)*, pages 619–626, 2001.

[3] J. F. Canny. Collaborative filtering with privacy. In *Proc. IEEE Symp. on Security and Privacy*, pages 45–57, 2002.

[4] P. Drineas, I. Kerenidis, and P. Raghavan. Competitive recommendation systems. In *Proc. 34th ACM Symp. on Theory of Computing* , pages 82–90, 2002.

[5] J. Kleinberg and M. Sandler. Convergent algorithms for collaborative filtering. In *Proc. 4th ACM Conf. on Electronic Commerce (EC)*, pages 1–10, 2003.

[6] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Recommendation systems: A probabilistic analysis. In *Proc. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 664–673, 1998.

[7] S. K. Lam and J. Riedl. Shilling recommender systems for fun and profit. In *Proc. 13th Conf. on World Wide Web (WWW)*, pages 393–402. ACM Press, 2004.

[8] M. P. O'Mahony, N. J. Hurley, and G. C. M. Silvestre. Utility-based neighbourhood formation for efficient and robust collaborative filtering. In *Proc. 5th ACM Conf. on Electronic Commerce (EC)*, pages 260–261, 2004.

[9] M. O. Rabin. Transaction protection by beacons. *J. of Comp. and Syst. Sc.*, 27(2):256–267, 1983.

[10] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proc. 1994 ACM conf. on Computer Supported Cooperative Work*, pages 175–186, 1994.

[11] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proc. 10th Int. Conf. on World Wide Web (WWW)*, pages 285–295. ACM Press, 2001.

[12] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *Proc. 2nd ACM Conf. on Electronic Commerce (EC)*, pages 158–167. ACM Press, 2000.

[13] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proc. 25th Ann. Int. ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '02)*, pages 253–260, 2002.