Collaborate With Strangers To Find Own Preferences[*]

Extended Abstract

Baruch Awerbuch[†]   Yossi Azar[‡]   Zvi Lotker[§]   Boaz Patt-Shamir[¶]   Mark R. Tuttle[‖]

April 29, 2005

Abstract

We consider a model with n players and m objects. Each player has an unknown grade for each object, modeled by a "preference vector" of length m. A player can learn his grade for an object by probing that object, but performing a probe incurs cost. The goal of the players is to learn their own evaluations of objects with minimal cost, by adopting the results of probes performed by other players. To facilitate communication, we assume that players collaborate by posting their grades for objects on a shared billboard: reading from the billboard is free. We consider players whose preference vectors are popular, i.e., players whose preferences are common to many other players. We present distributed and sequential algorithms to solve the problem with logarithmic cost overhead.

1   Introduction

Most people encounter recommendation systems on a daily basis in many situations, such as buying a book, renting a movie, choosing a restaurant, looking for a service provider etc. Recommendation systems are supposed to help us make choices among the different alternatives that are available to us. The introduction of the Internet had made recommendation systems even more critical than ever, since entities (such as users and products) on the Internet are typically virtual, and may be located on the other side of the globe. As a result, a user may have very little side-information to work with, and must rely on computerized recommendation systems. The idea in these systems is to take advantage of the existence of many users with similar opinions: a set of users with similar preferences (called *type*) can collaborate by sharing the load of searching the object space and identifying objects they deem as good. However, players belonging to the same type do not know of each other a priori, and therefore, implicitly or explicitly, significant effort must be directed into identifying potential collaborators (while avoiding possible crooks).

Much of the extant research has concentrated on trying to find a good object for each user; in this paper we address a more general question, where the goal of the users is to try to find what are their opinions on *all* objects. The advantage of having such an algorithm is that it allows users to identify others who share their opinions, which may be useful in the long range: When a new batch of objects arrives, much effort can be saved by using the opinions of users who have already been identified as belonging to my own type. Moreover, as we show in this paper, the cost of finding the complete set of grades (and hence identifying users with similar preferences) is comparable with the cost of finding just one good object for each user.

Roughly speaking, our model is as follows (formal definitions are provided in Section 2). We assume that there are n users and m objects, and that each user has an unknown grade for each object, i.e., each user can be represented by his preference vector. The goal is for each user to find his complete preference vector.[1] The problem is that players do not know what is their preference for an object unless they *probe* it, but probing is a costly operation (e.g., the user needs to buy the product). Nevertheless, players would like to predict what is their preference for items they did not probe. To facilitate this, the system provides a shared "billboard," on which players can post the results of their probes, thus allowing players with similar preferences to split the probes among them and share the results. The existence of a billboard does not immediately solve the problem: the question now is which grades should one adopt, since players may have arbitrarily different preference vectors.

Symmetric operation.   One important requirement imposed on any reasonable solution is that it must be symmetric, i.e., with high probability, all players execute nearly the same amount of work. If this restriction is lifted, it is easy to come up with simple asymmetric "committee-style" solutions to the problem (see Sec. 5.1). However, such algorithms, while possibly suitable for centralized settings, seems completely unrealistic in the distributed model, for the following reasons:

- Lack of fairness: the number of objects may be huge, and the tolerance of players for doing a lot of work for others may be in short supply (cf. [4, 1] for more discussion).

---

[1]We arbitrarily write "he" for users, but it should be read as "he or she."

- Distrust of committees: small-size committees are prone to easy manipulation by adversaries (bribery, intimidation, other forms of corruption) and thus cannot be trusted to be objective judges on important matters.

1.1 Lower bounds

Before we state our results precisely, let us sketch two trivial lower bounds on the cost. Consider first a nearly ideal scenario, where all players in a given type T know the identities of all other type-T players, but they don't know what is the type-T preference vector. Even if the players coordinate perfectly to minimize their overall cost to find the vector, at least m probes must be executed just so that each object has at least one grade from a type-T player. This gives a lower bound of $\Omega(m)$ on the cost. Now consider a second nearly ideal scenario, in which all type vectors are common knowledge, and the only question a player must answer is what type does he belong to. It is not hard to see that if there are k possible types, $\Omega(k)$ expected probes by each player are unavoidable. It follows that the total cost of all players of a single type of size $\Omega(n/k)$ is $\Omega(n)$. Combining the lower bounds from the two scenarios above we conclude that no algorithm can solve the problem is $o(m + n)$ cost over all players of a given type.

1.2 Our results

In this paper we take another step toward understanding recommendation systems. Informally, we present algorithms (in the simplified model sketched above), that demonstrate that if one's unknown preferences are known not to be esoteric (i.e., it is known that there are quite a few other players with the same unknown preferences), then one can find his own preferences without performing more than a logarithmic number of probes. More specifically, we present symmetric algorithms that approach the above lower bound to within a logarithmic factor. The algorithms are randomized, and they are guaranteed to work with probability $1 - n^{-\Omega(1)}$. The algorithms rely on knowing a lower bound on the popularity of the type in question. Formally, the algorithms use a parameter $0 < \alpha \leq 1$ such that there are at least αn players with the preference vector we would like to output.

Our main result is a distributed (in fact, parallel) algorithm called $\mathcal{D}$, whose running time is $O\left(\frac{\log n}{\alpha}(\frac{m}{n} + 1)\right)$. This means that the total number of probes done by users of a given type is $O\left((n + m)\log n\right)$. We stress that our algorithm works for any set of user preferences: only the running time depends on the type size.

We also present a sequential algorithm that slightly improves the total number of probes for large α, while maintaining the crucial fairness property that all users make approximately the same number of probes. The sequential algorithm (called S_1) ensures that the total number of probes done collectively by players in the type is at most $O\left(m \log n + \frac{n}{\alpha}\right)$. Unfortunately, the running time of the latter algorithm may be high.

Detailed comparison with previous results is postponed to Sec. 5. As a one line summary, let us emphasize at this point that unlike our algorithms, all previous solutions to this problem are either asymmetric (i.e., some players do linear work), or cannot deal with all inputs.

Paper organization. The remainder of this paper is organized as follows. In Section 2 we formalize the model. In Section 3 we present our sequential algorithm, S_1. In Section 4 we present our parallel algorithm, $\mathcal{D}$. Some concluding remarks and open problems are presented in Section 6. In Section 5 we describe existing work. In section 6 we conclude with some open problems.

2 Model

Our model is formally defined as follows. There are n *players* and m *objects*. For each object $i \in \{1, \ldots, m\}$ and player $j \in \{1, \ldots, n\}$, there exists a value $v_i^j \in \{0, 1\}$, called the *grade* or *value* of object i to player j. The vector $v^j \stackrel{\text{def}}{=} (v_1^j, v_2^j, \ldots, v_m^j)$ is the *preference vector* of player j. A *type* is a vector $v \in \{0, 1\}^m$. A player j is said to *belong to type* v if his preference vector v^j is equal to v. We say that the *popularity* of a type is α, for some $0 < \alpha \leq 1$, if there are at least αn players of that type.

An *execution* is a sequence of steps, where in each step some players move. A basic move of a player j is to reveal his value v_i^j for some object i. In this case we say that *player j probed object i*, and say that v_i^j is the vote of j on i. Each player may probe one object in a step, but many players may probe the same object simultaneously. The players communicate by means of a shared billboard. To model its existence, we assume that the results of all past probes are freely available to all players.

An *algorithm* in our model is a function that decides which player probes which object, based on past probe results, and possibly using a tape of random bits. Algorithms are parametrized by the number of players and objects (called n and m, respectively), a confidence parameter (called c), and a lower bound α on the popularity of the type whose vector the algorithm needs to output. The individual or aggregate *cost* of an algorithm is the number of probes incurred by either an individual player or by all players of a given type, respectively. The running time of an algorithm is the total number of parallel steps taken since the start of the algorithm until all players (possibly of a given type) have terminated.

An algorithm is said to be *correct with probability p* if with probability p, by the end of the execution of the algorithm, each player j outputs the complete vector $v^j = (v_1^j, v_2^j, \ldots, v_m^j)$. Probability is taken over the tape of random bits. The algorithms presented in this paper are correct with *high probability*, meaning that the probability of failure is $n^{-\Omega(c)}$.

We stress that the only way randomization is used by algorithms in this paper is in assuming that the identities of players form a random permutation of $\{1, \ldots, n\}$, and that the identities of objects form a random permutation of $\{1, \ldots, m\}$.

3 Sequential Algorithm

In this section we present a simple algorithm called S_1 with low cost, but high running time. We fix our attention on a single type of popularity $\alpha > 0$. Algorithm S_1 works for any value of $0 < \alpha \leq 1$. Its cost is better than the cost of Algorithm $\mathcal{D}$ (presented in Section 4) for $\alpha = \Omega(1/\log n)$.

Algorithm S_1:

For each object i in turn do:

    1. Let the number of votes of qualified players for object i be q_i.

    2. Let $\theta_i \leftarrow \frac{\alpha}{2}(q_i - 8c \ln n)$.   // θ_i is a threshold

    3. If there is only one value v with more than θ_i qualified votes, then execute CALL(i, v), and proceed to the next object.

    4. Otherwise (multiple values pass threshold):

        (a) If not all players have probed i, let a random player probe object i and go to Step 1.

        (b) Otherwise (all players have already probed i), let v be the value seen by the local player. Execute CALL(i, v) and proceed to the next object.

Subroutine CALL(i, v):

  (1) Output the value v for object i.

  (2) Mark all players whose vote on i is different than v as "unqualified."

Figure 1: *Pseudo-code for Algorithm S_1.*

The algorithm works as follows (see Figure 1). The algorithm proceeds by determining the value of objects one at a time. This is done by letting random players probe the object until the players can conclude, with high probability, what is the value of the object (Step 3). The high probability test is based on the lower bound on the number of players with the same type: if there are too few players who think that the value of the object is v, then, given the lower bound on the popularity of the type, the value of that object to that type is not v. The threshold is computed in Step 2. It may be the case, however, that the high probability test will never end with a single value for the object (say when $\alpha < 1/2$). In this case all players might end up testing the object. Following this strategy naively will lead to the trivial cost of nm probes. To avoid that, we use the following simple idea. Each player locally marks each other player as either "qualified" or "unqualified." The interpretation of the marks is that a player j' is marked "unqualified" by player j only if there is evidence that j' does not belong to the type of j. Initially, all players are marked "qualified" by everyone. The marks are updated by subroutine CALL: Once an object value is determined, all players who voted for another value are "disqualified" by the current player i.e., that player disregards them from that point and onward.

The important property of the algorithm is that all players of the same type maintain a consistent view of which players are qualified and which aren't.

Lemma 3.1 *For any object i, all players of type T execute CALL(i, v) with the same value v. Furthermore, they all have the same set of qualified players.*

Proof: By induction on the objects. For the basis of the induction we have that all players of type T have the same set of players as qualified when the algorithm starts, since all players are qualified. For the induction step, consider the point when an object is called. This happens either in Step 3 or in Step 4b. In the former case, all players of type T will call the same value because they do it based on the same billboard and, by induction, they use the same set of qualified players. In case an object is

4

called in Step 4b, all players of type T have the same local value for the object by definition of type. The agreement on the set of qulified players follows in both cases. ∎

Lemma 3.1, in conjunction with the following lemma, implies correctness of the output.

Lemma 3.2 *If a player j of a type whose popularity is at least α executes $\text{CALL}(i, v)$, then $\mathbf{P}\left[v_i^j \neq v\right] < n^{-c}$.*

Proof: If player j executes $\text{CALL}(i, v)$ at Step 4b, then j actually probed i, and $v_i^j = v$ with certainty. If j executes $\text{CALL}(i, v)$ at Step 3, then we have that only the value v received more than θ_i of the q_i qualified votes. In this case it is straightforward to bound the probability that $v_i^j \neq v$ using the Chernoff bound as follows (we use the versions of [9]). The voters are chosen at random (Step 4a), and therefore, by assumption on the popularity of the type, the probability that a random vote is v_i^j is at least α. Hence

$$\mathbf{P}\left[v_i^j \neq v\right] \ \leq \ \mathbf{P}\left[v_i^j \text{ received less than } \theta_i \text{ votes}\right] \ < \ e^{-\alpha q_i/8} \ < \ e^{-c\ln n} \ = \ n^{-c} \ .$$

The first inequality above follows from Step 3; the second inequality follows from Chernoff, since the expected number of votes for v_i^j is at least $\alpha q_i > 2\theta_i$; and the last inequality follows from the fact that when $\text{CALL}(i, v)$ is executed, θ_i is strictly positive, and hence, by definition of θ_i, we $\alpha q_i > 8c\ln n$ at this point. ∎

We now bound the cost of S_1. To do that, we count the number of qualified votes, i.e., votes of players that are considered qualified when Step 1 is executed (note that some of the players who cast such votes may later be disqualified).

Lemma 3.3 *The total number of qualified votes throughout the execution is at most $O\left(m\ln n + \frac{1-\alpha}{\alpha}n\right)$.*

Proof Sketch: For each object i, let q_i' denote the total number of qualified players who vote on i when $\text{CALL}(i, v)$ is executed. We bound $\sum_i q_i'$ by counting the number of *disqualified* players. On one hand, each player may be disqualified at most once, and hence the total number of disqualifications, throughout the execution, is at most $(1-\alpha)n$. On the other hand, note that whenever $\text{CALL}(i, v)$ is executed, at least $\frac{\alpha}{2}((q_i'-1) - 8c\ln n)$ players are disqualified: this is because i was not called in the previous round, when there were $q_i'-1$ qualified votes, and therefore there were at least $\frac{\alpha}{2}(q_i'-1-8c\ln n)$ players who voted for a value different than v. It follows that

$$(1-\alpha)n \ \geq \ \sum_{i=1}^{m} \frac{\alpha}{2}\left(q_i' - 1 - 8c\ln n\right) \ ,$$

i.e.,

$$\sum_{i=1}^{m} q_i' \ \leq \ O\left(m\ln n + \frac{1-\alpha}{\alpha}n\right) \ . \qquad ∎$$

We summarize with the following theorem.

Theorem 3.4 *Suppose that there are at least αn players from a certain type, and that they all run Algorithm S_1. Then with probability $1 - n^{-\Omega(1)}$ they will all output the correct vector, after executing $O(m\log n + \frac{n}{\alpha})$ probes.*

Since in Algorithm S_1 only one player probes in each round, the time complexity is equal to the number of probes. It is easy to speed up the algorithm somewhat. For example, we can have all

---

Algorithm $\mathcal{D}(P, O)$:   *// P is a set of players and O is a set of objects*

1. If $|P| < \frac{8c \ln n}{\alpha}$ then probe all objects in O, output their values, and return.

2. (Otherwise) Let P' be the half of P such that $j \in P'$, and let P'' be the other half. Let O' be the half corresponding to P' in O, let O'' be the other half of O.

   // The partition of P and O is a fixed function of the IDs

3. Execute $\mathcal{D}(P', O')$.   *// upon returning, values for all objects in O' were output by all players in P', and values for all objects in O'' were output by all players in P''*

4. Scan the billboard. Let V be a set of vectors for O'' such that the each vector in V is voted for by at least $\alpha/2$ of the players in P''.

5. Let $C \subseteq O''$ be the set of objects for which there are different votes in V.

6. While $C \neq \emptyset$ do

   (a) Choose an arbitrary object $i \in C$. Probe i, and let the probe result be denoted by v.

   (b) Remove from V vectors whose value on i is not v.

   (c) Update C to contain all objects on which surviving vectors in V differ.

7. Output the (unanimous) votes of surviving vectors in V for all objects in O'' and return.

Figure 2: *Pseudo-code for Algorithm $\mathcal{D}$ as executed by player j. When The algorithm returns, the players in P have output values for all objects in O.*

players probe all objects in parallel until there are $O(\log n)$ samples per object, and then apply the algorithm only to the undetermined objects. We do not elaborate any further, since in the following section we present an algorithm with nearly the best possible parallel time.

Fuzzy types. The case of types that are not very crisply defined can be handled by a simple extension of the algorithm, given that the types are sufficiently separated. Specifically, suppose that we define a type to consist of all players whose preference vectors differ from the type vector in at most r objects (i.e., the Hamming distance of a player's preference vector from its type vector is at most r). Suppose further that the Hamming distance between any two type vectors is at least $4r + 1$. Algorithm S_1 can handle this case if we modify the way players are disqualified in Subroutine CALL: in the fuzzy case, a player j' will be disqualified by player j (and vice versa) after $2r + 1$ discrepancies between their values are detected. The separation assumption guarantees that no player ever disqualifies another player of his type. However, the running time of the algorithm is increased by a factor of $O(r)$, and the output of a player may differ from his own preferences for $O(r)$ objects. Details are omitted.

4   Parallel Algorithm

In this section we present our main result, a recursive parallel algorithm called $\mathcal{D}$. This algorithm achieves near-optimal time complexity while maintaining the cost modest.

The idea in Algorithm $\mathcal{D}$ is as follows (see Figure 2). Given a set of players and a set of objects, we

$$\begin{pmatrix}
v_1^1 & v_2^1 & \cdots & v_{m'}^1 & & & & \\
v_1^2 & v_2^2 & \cdots & v_{m'}^2 & & & & \\
\vdots & \vdots & \ddots & \vdots & & & & \\
v_1^{n'} & v_2^{n'} & \cdots & v_{m'}^{n'} & & & & \\
& & & & v_{m'+1}^{n'+1} & v_{m'+2}^{n'+1} & \cdots & v_{2m'}^{n'+1} \\
& & & & v_{m'+1}^{n'+2} & v_{m'+2}^{n'+2} & \cdots & v_{2m'}^{n'+2} \\
& & & & \vdots & \vdots & \ddots & \vdots \\
& & & & v_{m'+1}^{2n'} & v_{m'+2}^{2n'} & \cdots & v_{2m'}^{2n'}
\end{pmatrix}$$

Figure 3: *Matrix representation of the results after Step 3 in Algorithm $\mathcal{D}$. Each row represents a player, and each column represents an object. P' is the top half of the rows, and O' is the left half of the columns. $|P'| = |P''| = n'$ and $|O'| = |O''| = m'$. The entries in two quadrants of the matrix are unknown.*

split the players and objects into two subsets each, let each half of the players recursively determine the values of half of the objects, and then merge the results. Figure 3 gives a visual representation of the situation after returning from the recursive call. Merging results is simply filling in the missing entries. Merging is done as follows. First, each player finds the preference vectors that are sufficiently popular in the other half (Step 4). This leaves only a few candidate vectors the players needs to choose from to complete his row. Testing objects with disputed values (Steps 5–6), the player eliminates at least one vector in each probe, and eventually, the player adopts the last surviving vector.

In the algorithm, we assume that the partition of sets of players and objects into halves is done by a fixed function. For example, if the current set of players is $P = \{j, j+1, \ldots, j+k-1\}$ and the current set of objects is $O = \{i, i+1, \ldots, i+k'-1\}$, then P' may be defined by $P' = \{j, j+1, \ldots, j+k/2-1\}$ and similarly $O' = \{i, i+1, \ldots, i+k'/2-1\}$. The sets P'' and O'' are defined to be the remainders $P \setminus P'$ and $O \setminus O''$, respectively. Since we assume that the identities of players and objects form a random permutation, these choices define random partitions of P and O into two halves each (we assume, for convenience, that n and m are powers of 2).

In the analysis, we focus our attention on a certain type T with popularity α.

Lemma 4.1 *Fix an invocation of $\mathcal{D}(P,O)$ with $|P| \geq \frac{8c \ln n}{\alpha}$. Then with probability at least $1 - n^{-c}$, there are at least $\alpha|P|/2$ type-T players in P in that invocation.*

Proof: The lemma follows from the fact that P is a random subset of the players, and hence the expected number of type-T players in P is $\alpha|P|$. Using Chernoff we get that

$$\mathbf{P}\left[\text{less than } \tfrac{\alpha|P|}{2} \text{ type-}T \text{ players in } P\right] < e^{\alpha|P|/8} \leq n^{-c} ,$$

since $|P| \geq \frac{8c \ln n}{\alpha}$. ∎

Lemma 4.2 *The total number of probes for a single player in the execution of the algorithm is $O\left(\frac{\log n}{\alpha}\left(\frac{m}{n}+1\right)\right)$.*

Proof Sketch: Step 1 is taken only once by a player. Since the player set and the object set are

halved in each recursive call, $|P| < \frac{8c \ln n}{\alpha}$ implies that in that call, $|O| < m \cdot \frac{8c \ln n}{\alpha n}$, i.e., there are $O\left(\frac{m \log n}{\alpha n}\right)$ are done in Step 1. In each other invocation of Algorithm P, the algorithm makes at most one probe for each member of V. Since by Step 4 each member of V represents at least an $\alpha/2$ fraction of the players, we have that $|V| \leq \frac{2}{\alpha}$. Since the depth of the recursion is $\log \frac{n}{8c \ln n/\alpha} = O(\log n)$, we have that the total number of probes done in Step 6a by each player, throughout the execution of the algorithm, is $O\left(\frac{\log n}{\alpha}\right)$. The result follows. ∎

We summarize in the following theorem.

Theorem 4.3 *Suppose that there are $\Theta(\alpha n)$ players from a certain type T, and that they all run Algorithm $\mathcal{D}$. Then with probability $1 - n^{-\Omega(1)}$ they will all output the correct vector, after executing $O((m + n) \log n)$ probes in $O\left(\frac{\log n}{\alpha}\left(\frac{m}{n} + 1\right)\right)$ rounds.*

We note that both the total cost and the individual running time of Algorithm $\mathcal{D}$ are the best possible, up to a factor of $O(\log n)$.

5 Related work

Most prior research on recommendation systems focused on a centralized, off-line version of the single recommendation 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 [11] (or products [12]) in the data set, and using past grades by users in a cluster to predict future grades by other users in the same cluster. Singular Value Decomposition (SVD) was also shown to be an effective algebraic technique for the off-line single recommendation problem [13]. Some of these systems enjoy industrial success, but they are known to perform poorly when prior data is less than plentiful [14], and they are extremely vulnerable even to mild attacks [8, 10]. Canny [3] gives a distributed secure and private SVD computation for the off-line single recommendation version of the problem.

Algorithmic results.   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.* [7] 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 [6] 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.

Singular value decomposition approachers.   Azar *et al.* introduced in [2] the idea of using SVD to reconstruct the unknown preference vectors. This idea was later used by Drineas *et al.* [4]. The attractive feature of the SVD method is its ability to deal with some noise. However, SVD

is inherently incapable of handling preference vectors that cannot be approximated by a low rank matrix. By contrast, our algorithm requires only a bound on the individual type size: what happens with other types is completely irrelevant. As a result, our algorithms can deal with *any* input, and the running time for a given type is nearly optimal, while the correctness of SVD-based algorithms is dependent on players from other types. This makes SVD an easy target even for weak adversaries (e.g., non-adaptive), against which our algorithms are immune.

Best value approachers. Much of the research on recommendation systems concentrated on a passive model, where the algorithm analyzes the data and is supposed to make recommendations; the influence of the algorithm on what data is collected, was ignored. An *active* model of recommendation systems was introduced in [4] (called "competitive recommendation systems" there). In this model the objective is to find, for each user, at least one object he rates as "good." If preference vectors of different types are orthogonal (i.e., do not share objects they both rate as "good"), then these algorithms can be used to reconstruct the complete preference vectors of the users. In fact, [4] uses SVD and works only if type orthogonality holds; it is vulnerable to attacks by dishonest users. In [1], it is shown that a simple committee-based algorithm (see below) suffices to find a good recommended object without restricting the user preferences. Additionally, [1] give a distributed peer-to-peer algorithm for picking a good object, that can withstand any number of dishonest users (the performance depends only logarithmically on the total number of users).

Again, our algorithms find the complete preference vectors without assumptions. We also note that finding complete vectors is useful in the long run: as a by-product, our algorithms generate a classification of users into types, which can be re-used when coping with a new batch of objects.

Learning relations. Another related problem is the learning model of Goldman *et al.* [5], where the algorithm works for all inputs. The setting in [5] is that a centralized algorithm needs to learn a binary relation: the relation value for all (i, j) pairs must be output. In each step, the algorithm *predicts* the value of the relation for some pair i, j, and then the true value is revealed. The measure of performance in this case is the number of errors committed by the algorithm. For this model they consider a few types of schedules, including the case where the algorithm chooses which entry to guess next, which is similar to the on-line model we consider. However, there are a few important differences between the learning model and ours. First, since the true values become known while the algorithm is running, the task is easier than the one we consider, where almost all values remain unexposed even after the algorithm has terminated. Formally, in our model, revealing any true grade incurs cost, and hence the cost of the algorithm of [5] in our model is linear, i.e., the worst possible. Second, the learning algorithms are fundamentally centralized in the sense that some players (possibly all) will probe all objects similarly to the committee-based algorithms (see below).

5.1 Asymmetric algorithms

One simplistic approach to solve the recommendation problem is to use *committees* [4, 1]. In this case, a few players are essentially charged with doing the work for all others. Consider the following algorithm for our problem:

(1) Choose $K = O\left(\frac{\log n}{\alpha}\right)$ random players.

(2) Let each player probe all objects.

(3) Let all other players find the committee member with whom they agree.

Step 3 can be implemented in K probes by each non-committee player (as done in Steps 5–6 of Algorithm $\mathcal{D}$ in Section 4). The correctness of the algorithm follows from the observation that with high probability, the committee contains at least one member from each type whose frequency is at least α. However, as mentioned in the introduction, committee-based algorithms are simply unacceptable in a distributed setting, due to their inherent unfairness, and to their vulnerability to malicious attacks.

6 Conclusion and Open Problems

In this paper we showed that in a highly simplified model, there exist very simple solutions to the important problem of low-cost preference reconstruction. Our results allow us to hope that the considerable gaps between the abstract model and real life can be bridged by algorithmic solutions. In particular, we believe that the following problems are important to solve.

- The algorithms in this paper are synchronous, i.e., players are assumed to proceed in lockstep fashion. Finding an asynchronous algorithm would be a significant progress in making the algorithm practical.

- Another important aspect is that the algorithms in this paper assume that all players of the same type have identical preference vector. It would be extremely useful to extend the algorithms to the case where types are not sharply defined, i.e., to allow some tolerance for deviation (some partial results for the sequential algorithm are mentioned in Section 3).

- The algorithms use random player and product identities. It seems interesting to try to develop an algorithm that lifts this assumption.

- The algorithms rely on knowing a lower bound on α, the popularity of the type. We would like to find an algorithm that adapts to any α.

Taste and honesty. One (post modern) interpretation for taste is *honesty*. In this model, each object has a single value ("truth"), independent of the player who probes it. This model is applicable if the grades of objects reflect some objectively defined predicate. In this case a player who announces a value which is not the true value of the object is simply lying. With this interpretation, our algorithms can be used to ensure that, with high probability, *each and every lie will eventually be uncovered*. This certainty may lead to the following intriguing consequence (assuming that the penalty for getting caught lying is larger than the gain): no rational player will ever lie, and the overhead of enforcing honesty will be minimized, since the algorithm will be invoked only in the rare cases of irrationality. Analyzing this scenario is beyond the scope of the current paper.

References

[1] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. Tuttle. Improved recommendation systems. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1174–1183, January 2005.

[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 *IEEE Symposium 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 (STOC)*, pages 82–90, 2002.

[5] S. A. Goldman, R. L. Rivest, and R. E. Schapire. Learning binary relations and total orders. *SIAM J. Computing*, 22(5):1006–1034, October 1993.

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

[7] 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.

[8] S. K. Lam and J. Riedl. Shilling recommender systems for fun and profit. In *13th conference on World Wide Web*, pages 393–402. ACM Press, 2004.

[9] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[10] M. P. OMahony, N. J. Hurley, and G. C. M. Silvestre. Utility-based neighbourhood formation for efficient and robust collaborative filtering. In *5th ACM conference on Electronic Commerce*, pages 260–261, 2004.

[11] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM Press, 1994.

[12] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *10th international conference on World Wide Web*, pages 285–295. ACM Press, 2001.

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

[14] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *25th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '02)*, pages 253–260, 2002.