Collaborate With Strangers to Find Own Preferences*

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

March 15, 2006

Abstract

We consider a model with n players and m objects. Each player has a "preference vector" of length m, that models his grades for all objects. The grades are initially unknown to the players. A player can learn his grade for an object by probing that object, but performing a probe incurs cost. The goal of a player is to learn his preference vector 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 a sequential and a parallel algorithm to solve the problem with logarithmic cost overhead.

*An extended abstract of this work appeared in the 17th Ann. ACM Symp. on Parallelism in Algorithms and Architecture, Las Vegas, Nevada, July 2005.

†Dept. of Computer Science, Johns Hopkins University. Email: `baruch@cs.jhu.edu`. Supported by NSF grant ANIR-0240551 and NSF grant CCR-0311795.

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. Email: `azar@tau.ac.il`. Research supported in part by the German-Israeli Foundation and by the Israel Science Foundation.

§Kruislaan 413 P.O. Box 94079, CWI, 1090 GB Amsterdam, The Netherlands. Email: `lotker@cwi.nl`.

¶School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel. Email: `boaz@eng.tau.ac.il`. Research supported in part by Israel Ministry of Science and Technology and by the Israel Science Foundation.

‖Intel Strategic CAD Lab, 77 Reed Road, Hudson, MA 01749. Email: `tuttle@acm.org`.

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.[1] The goal is for each user to find his complete preference vector. 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. To allow a non-trivial solution to exist, we assume that some preference vectors are popular, namely many players have them. Such players are said to belong to the same *type*. Ideally, players of the same type should share the work and the results among them; however, it is not known who belongs to which type when the algorithm starts.

Symmetric operation. One important requirement imposed on any reasonable solution is that it must be symmetric, i.e., with high probability, all players execute more-or-less the same amount of work. If this restriction is lifted, it is easy to come up with simple asymmetric "committee-style"

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

solutions to the problem (see Sec. 5.1). However, such algorithms, while possibly suitable for centralized settings, seem unrealistic in the distributed model, for the following reasons (see [3, 1] for more discussion of the issue):

- 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.
- 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, each object must be probed at least once by some type-T player. Therefore, a lower bound of $\Omega(m)$ on the total cost to the players of type T is unavoidable. 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 $k \le m$ possible types, then $\Omega(k)$ 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 $\Omega(n)$ in the worst case. 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), which 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 $0 < \alpha \le 1$ such that there are at least αn players with the preference vector we would like to output.

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

We also present a sequential algorithm which 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 any given type is at most $O\left(\frac{1}{\alpha}(m \log n + n)\right)$, and these probes are approximately evenly distributed among the players. Unfortunately, the running time of the latter algorithm may be high.

Detailed comparison with previous results is given in Sec. 5.

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, $\mathcal{D}$. 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.[2] 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 $0 \leq \alpha \leq 1$, 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 probe an object i, thus revealing the value v_i^j. 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.

By the end of an execution of an algorithm, each player outputs a vector in $\{0,1\}^m$. An algorithm is said to be *correct with high probability for players of type* T if with probability $1 - n^{-\Omega(c)}$, the output vector of each player of type T is exactly the player's preference vector, where c is the confidence parameter of the algorithm. Probability is taken over the tape of random bits.

We assume that the identities of the players and objects are unique.

[2] In fact, the grades can be drawn from any set; we assume the binary value model for simplicity of presentation only.

Algorithm S_1:

For each object i in turn do:

1. Let the current number of votes by qualified players for object i be q_i.
2. Let $\theta_i \leftarrow \frac{\alpha}{2} \cdot q_i - 8c \ln n$. *($\theta_i$ 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 new 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 "disqualified."

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

3 Sequential Algorithm

In this section we present a simple algorithm called S_1 with low cost, but high running time.

3.1 Description of Algorithm S_1

We fix our attention on a single type T of popularity $\alpha > 0$. Call the players of type T *T-typical*. 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) when $\alpha = \Omega(1/\log n)$ and $m = o(n/\log n)$.

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 α on the fraction of players with the same type. Using α, we compute, in Step 2, a threshold θ such that it is extremely unlikely that less than θ T-typical players probed the object so far. Thus, any value with less than θ votes is unlikely to be the true value for type T players. Probing proceeds until there is only one value that passes the threshold. However, it may be the case, when $\alpha < 1/2$, that the high probability test will never end with a single value for the object: there may always be more than one value which pass the current threshold. In this case all players will end up probing the object. Thus, the threshold strategy alone, for $\alpha \geq 1/2$, may lead the T-typical players to the trivial cost of αnm probes in the worst case. To avoid that, we use the following simple additional idea. Each player locally marks each other player as either "qualified" or "disqualified." The interpretation of the marks is that a player j' is marked "disqualified" 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.

3.2 Analysis of S_1

The important invariant maintained by the algorithm is that all typical players have a consistent view of which players are qualified and which are not.

Lemma 3.1 *For any object i, with probability at least $1-i\cdot n^{-c}$, all T-typical players execute* CALL(i, v) *with the same value v. Furthermore, they all have the same set of qualified players.*

Proof: By induction on the iterations of the main loop of the algorithm. For the basis of the induction we have that all T-typical players have the same set of players as qualified when the algorithm starts, since all players are qualified for anyone. For the induction step, consider the point when an object is called. This happens either in Step 3 or in Step 4b. We proceed by case analysis. If the object is called in Step 4b, all players of type T have the same local value for the object by definition of type. If the object was called in Step 3, then 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. (Note that by the specification of the algorithm, even players that probed the object will call the value according to the billboard, possibly—albeit with low probability—in disagreement with his own vote!) The agreement on the set of qualified players follows in both cases. ∎

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

Lemma 3.2 *If a T-typical player j executes* CALL(i, v), *then* $\mathbf{P}\left[v_i^j \neq v\right] < n^{-c}$.

Proof: If player j executes CALL(i, v) at Step 4b, then j actually probed i, and $v_i^j = v$ with certainty. If j executes 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 tail bound for hypergeometric distributions as follows (see, e.g., [13]). 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 the expected number of votes for v_i^j is at least αq_i; if $v \neq v_i^j$ then v_i^j received less than θ_i votes, which happens with probability

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

The first inequality above follows from Step 3; the second inequality follows from the Chernoff-like tail inequality of [13] , since the expected number of votes for v_i^j is at least $\alpha q_i > 2\theta_i$; the last inequality follows from the fact that when CALL(i, v) is executed, θ_i is non-negative (because the number of votes for the single value with most votes must be positive), and hence, by definition of θ_i, we have $\alpha q_i > 8c \ln n$ at this point. ∎

Corollary 3.3 *With probability at least* $1 - \frac{mn}{n^c}$*, all* T*-typical player output the correct values for all objects.*

Proof: Follows from Lemma 3.1 and the union bound (applied over all αn T-typical players and all m objects). ∎

We now bound the cost of S_1. Let us call a vote to an object i "qualified" if it is due to a player considered qualified at the time of calling object i (note a qualified vote may be cast by a player who later becomes disqualified). To bound the cost, in the following lemma we count the number of qualified votes.

Lemma 3.4 *With probability at least* $1 - \frac{mn}{n^c}$*, the total number of qualified votes throughout the execution cast by players of type* T *is at most* $O\left(\frac{\ln n}{\alpha} m + \frac{1-\alpha}{\alpha} n\right)$*.*

Proof: By Corollary 3.3 and Lemma 3.1, with probability at least $1 - \frac{mn}{n^c}$, all T-typical players decide correctly, and in the same way for objects, and therefore have identical sets of qualified players. We henceforth condition on this event. In this case, for each object i, let q'_i denote the total number of qualified votes on i when $\text{CALL}(i, v)$ is executed. Since T-typical players are never disqualified by other T-typical players (in the high-probability case we consider), the total number of probes executed by T-typical players is at most $\sum_{i=1}^{m} q'_i$. We bound this sum by counting the number of *disqualified players*. On one hand, a player may be disqualified at most once. By Corollary 3.3, with high probability only non-typical players are disqualified and hence the total number of disqualifications by a typical player, throughout the execution, is at most $(1-\alpha)n$. On the other hand, note that whenever $\text{CALL}(i, v)$ is executed, at least $\frac{\alpha}{2} \cdot (q'_i - 1) - 8c \ln n$ 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 $\frac{\alpha}{2} \cdot (q'_i - 1) - 8c \ln n$ players who voted for a value different than v. It follows that

$$\begin{aligned} (1-\alpha)n &\geq \sum_{i=1}^{m} \left(\frac{\alpha}{2}(q'_i - 1) - 8c \ln n \right) , \qquad \text{or, rearranging,} \\ \sum_{i=1}^{m} q'_i &\leq \frac{2}{\alpha}((1-\alpha)n + 8cm \ln n) + m , \end{aligned}$$

and the lemma follows. ∎

We summarize with the following theorem.

Theorem 3.5 *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* $O(\frac{1}{\alpha}(m \log n + n))$ *probes.*

3.3 Notes

Randomization. The only way Algorithm S_1 utilizes randomization is by letting the players access the objects in random order.

Multiple values. We note that the algorithm works for objects with arbitrary values. The assumption we use is that the preference vectors of all players belonging to the same type are *identical*. The grades (entries of the vectors) can be from any space, so long as we can test for equality.

Multiple types interaction. When using Algorithm S_1, all players, of all types, take part. However, each type has a different view of the current state of the protocol, including state components such as the set of qualified players, the current item to probe etc. These different views may lead to different schedules: it may be the case that while players of type T have already determined the value of an object i and moved to the next object $i+1$, players of another type (with a different α) still have more than one value for i which passes their threshold, and thus they will keep probing object i. We note that this divergence between types can only help the algorithm, because the only coordination crucial to the correctness is that the players of the *same* type remain synchronized, and that the θ threshold is still a lower bound on the frequency of the typical players in their current object.

In particular, the algorithm proceeds by specifying a globally known random order (schedule) by which players probe objects. Since all players in a given type maintain an identical view of who is qualified, all we need to do is to add the natural rule that disqualifies (in the eyes of the players of type T) anyone who does not conform to the schedule as computed by a type T player. For example, it may be the case that all type T player expect the next probe to be performed by player 17 to object 4, but player 17 probes object 5. The only interpretation of such an event is (w.h.p.) that player 17 is not a type T player, and therefore it will be disqualified by type T players, who will continue to probe object 4. Again, since we assume that the schedule and the probes are available to all players, the coordination between players of the same type will not be violated. The net effect of this divergence is to have parallel threads running the algorithm, where all players of each type all run the same thread.

Time Complexity. Since in Algorithm S_1 only one player probes in each round, the time complexity for a type is never less than $\Omega(m \log n)$. It is easy to speed up the algorithm somewhat. We do not elaborate any further on this idea, since in the following section we present an algorithm with nearly the best possible parallel time.

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.

4.1 Description of Algorithm $\mathcal{D}$

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

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

1. If $\min(|P|, |O|) < \frac{16c \ln n}{\alpha}$ 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. *(Use random partitions of P and O)*
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 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; let v denote the probe result.
   (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.*

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 player set P is randomly partitioned into two equal-size disjoint subsets $P = P' \cup P''$. Similarly, we assume that the objects set O is partitioned into two subsets $O = O' \cup O''$. We assume, for convenience, that n and m are powers of 2.

4.2 Analysis of Algorithm $\mathcal{D}$

As usual, we focus our attention on a certain type T with popularity α.

Lemma 4.1 *Let P be the set of players in any invocation of $\mathcal{D}(P, O)$. Then with probability at least $1 - n^{-c}$, there are at least $\alpha|P|/2$ players of type T 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 T-typical players in P is $\alpha|P|$. Let t be the number of T-typical players in P. Using the tail bounds for hypergeometric distributions [13], we obtain

$$\mathbf{P}\left[t < \frac{\alpha|P|}{2}\right] \; < \; e^{-\alpha|P|/8} \; \leq \; n^{-c} \; ,$$

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

Corollary 4.2 *With probability at least $1 - n^{-c+1}$, in all invocations $\mathcal{D}(P, O)$ of the algorithm, P contains at least $\alpha|P/2$ T-typical players.*

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

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.*

Proof: Consider the partition of the players into the smallest subsets $\{P_i\}$, used in the bottom level of the recursion. By Lemma 4.1, we have that the probability that each P_i will contain less than $\alpha|P_i|/2$ T-typical players is at most $1 - n^{-c}$. Since there are $O(n/\log n)$ such subsets, it follows from the union bound, that the probability that all these smallest subsets contain at least than $\alpha|P_i|/2$ T-typical players each is at least $1 - n^{-c+1}$. Consider now a player set P' in an arbitrary invocation of the algorithm. By the algorithm, P' is the union of some of the smallest subsets, and therefore, with probability at least $1 - n^{-c+1}$, P' contains at least $\alpha|P'|/2$ T-typical players. ∎

Lemma 4.3 *Under Algorithm $\mathcal{D}$, with probability at least $1 - n^{-c+1}$ all values filled in by T-typical players are correct.*

Proof: By Corollary 4.2, with probability at least $1 - n^{-c+1}$, each player set P in any invocation of the algorithm contains at least $\alpha|P|/2$ T-typical players. Conditioned on this property, we prove the lemma by induction on the levels of recursion. The base case is depth 0, where the error probability is 0 since in Step 1 values are trivially correct because they are filled based on direct probes. For the inductive step, consider an invocation with players $P = P' \cup P''$ and objects $O = O' \cup O''$. let us focus on the players in P'. By assumption, there are at least $\alpha|P''|/2$ T-typical players in P'', and by the induction hypothesis, their output for O'' is correct (and identical). Therefore, in Step 4, their common vector will be chosen into V. Steps 5 and 6 remove from V vectors which disagree with the player doing the probing, and thus they do not remove the vector corresponding to T-typical players. It follows that the vector adopted by a T-typical player from P' for the objects in O'' is correct; since by induction its vector for O' was also correct, we are done with the inductive step. ∎

Lemma 4.4 *The total number of probes for a single player in the execution of the algorithm is* $O\left(\frac{\log n}{\alpha}\left\lceil\frac{m}{n}\right\rceil\right)$.

Proof: Consider Step 1 first, and let P_1 and O_1 be sets of players and objects, respectively, when this step is executed. Step 1 is taken only once by a player, and its cost is precisely $|O_1|$ probes. If Step 1 is taken because $|P_1| < \frac{16c\ln n}{\alpha}$, then since the player set and the object set are halved in each recursive call, we conclude that when Step 1 is taken, $|O_1| < \frac{m}{n} \cdot \frac{16c \ln n}{\alpha}$. Otherwise, Step 1 is taken because $|O_1| < \frac{16c \ln n}{\alpha}$. In any case, Step 1 incurs at most $O(\lceil \frac{m}{n} \rceil \frac{\log n}{\alpha})$ probes.

The only other probes made by the algorithm are made in Step 6a, where the vectors in V are eliminated. In any given invocation of the algorithm, each player makes at most $|V|$ probes at Step 6a. Since by Step 4 each member of V represents at least an $\alpha/2$ fraction of the players, we have that $|V| \le \frac{2}{\alpha}$. Since each probe in Step 6 removes at least one vector from V, we have that each player makes at most $O(1/\alpha)$ probes in each non-bottom level of the recursion. 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 4.5 *Suppose that there are at least αn players from a certain type T. Then under Algorithm $\mathcal{D}$, with probability at least $1 - n^{-\Omega(1)}$ all players of type T will all output the correct vector, after executing $O((m+n)\log n)$ probes in $O\left(\frac{\log n}{\alpha} \lceil \frac{m}{n} \rceil\right)$ rounds.*

Proof: Correctness follows from Lemma 4.3, and the time complexity follows from Lemma 4.4. For the overall cost to the T-typical players, note that if $m \ge n$ then the total number of probes is $O(\alpha n \cdot \frac{m}{n} \cdot \frac{\log n}{\alpha}) = O(m \log n) = O((m+n)\log n)$. And if $m < n$, then the total number of probes is $O(\alpha n \cdot \frac{\log n}{\alpha}) = O(n \log n) = O((m+n)\log n)$. ∎

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 $O(\log n)$.

4.3 Notes

Randomization. The only way algorithm $\mathcal{D}$ uses randomization is in selecting a random partition of the objects and players in each recursive invocation.

Multiple values. Like Algorithm S_1, Algorithm $\mathcal{D}$ works for any set of values. The important restriction is that all players of the same type have exactly the same preference vector.

Multiple types interaction. The algorithm is described from the point of view of a single type T. Similarly to Algorithm S_1, when we run algorithm $\mathcal{D}$ with many types, each types has its own α. This may cause the executions of different types to diverge: since the time invested in each invocation of $\mathcal{D}$ is proportional to $1/\alpha$, it is not difficult to see that types with larger α will finish faster. This means that when players with large α start a recursive call of level ℓ, say, the players with smaller α may lag behind, which leaves some unfilled rows. However, these missing rows cannot be candidate vectors in V (in Step 4), and therefore the players with large α values need not wait for players with small α values. Thus the time complexity stated in Theorem 4.5 holds for each type (or, more precisely, for each α) independently.

5 Related work

Most prior research on recommendation systems focused on a *passive* model: 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. In this work we study the *active* algorithms, which propose to the users which objects to probe, and process their feedback to achieve the desired result.

Let us first review passive algorithms. Passive algorithm usually work by heuristically identifying clusters of users [9] (or products [10]) 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 [11]. Some of these systems enjoy industrial success, but they are known to perform poorly when prior data is less than plentiful [12], and they are quite vulnerable even to mild attacks [7, 8].

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 passive algorithms 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.* introduced in [2] the idea of using SVD to reconstruct the unknown preference vectors. 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 algorithms require only a bound on the individual type size: what happens with other types is 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.

The *active* model was introduced in [3] (called "competitive recommendation systems" there). In that paper 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, [3] uses SVD and works only if type orthogonality holds; it is vulnerable to attacks by dishonest users. Subsequently, it was shown in [1] 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).

Learning relations. Another related problem is the learning model of Goldman *et al.* [4], where the algorithm works for all inputs. The setting in [4] 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 must *predict* 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. Several variants of the basic model are possible, depending on who decides which (i, j) entry is the next one to predict. In [4], they consider, among others, 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 [4] 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* [3, 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.
- 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.

Acknowledgment

We thank Avrim Blum for pointing the existence of [4] to us and the anonymous reviewer for clearing some bugs from this 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] P. Drineas, I. Kerenidis, and P. Raghavan. Competitive recommendation systems. In *Proc. 34th ACM Symp. on Theory of Computing (STOC)*, pages 82–90, 2002.

[4] 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.

[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. 39th 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 International 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] 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 (CSCW)*, pages 175–186. ACM Press, 1994.

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

[11] 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.

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

[13] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM J. Disc. Math*, 8(2):223–250, 1995. Preliminary version in SODA 1993.