Tell Me Who I Am: An Interactive Recommendation System

Noga Alon[*] Tel Aviv U. and IAS

Baruch Awerbuch[†] Johns Hopkins U.

Yossi Azar[‡] Tel Aviv U.

Boaz Patt-Shamir[§] Tel Aviv U.

January 9, 2008

Abstract

We consider a model of recommendation systems, where each member from a given set of *players* has a binary preference to each element in a given set of *objects*: intuitively, each player either likes or dislikes each object. However, the players do not know their preferences. To find his preference of an object, a player may *probe* it, but each probe incurs unit cost. The goal of the players is to learn their complete preference vector (approximately) while incurring minimal cost. This is possible if many players have similar preference vectors: such a set of players with similar "taste" may split the cost of probing all objects among them, and share the results of their probes by posting them on a public *billboard*. The problem is that players do not know a priori whose taste is close to theirs. In this paper we present a distributed randomized peer-to-peer algorithm in which each player outputs a vector which is close to the best possible approximation of the player's real preference vector after a polylogarithmic number of rounds. The algorithm works under adversarial preferences. Previous algorithms either made severely limiting assumptions on the structure of the preference vectors, or had polynomial overhead.

[*] Schools of Mathematics and Computer Science, Tel Aviv University, Tel Aviv 69978, Israel and IAS, Princeton, NJ 08540, USA. Research supported in part by the Israel Science Foundation and by the Von Neumann Fund. Email: nogaa@tau.ac.il.

[†] Dept. of Computer Science, Johns Hopkins University. Supported by NSF grants ANIR-0240551, CCF-0515080 and CCR-0311795. Email: baruch@cs.jhu.edu.

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

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

1 Introduction

Recommendation systems come up on a daily basis in many human activities, such as buying a book, renting a movie, choosing a restaurant, looking for a service provider, etc. Abstractly, one can think of users and objects, and the task of a recommendation system is to predict which of the objects each user would like. To do that, recommendation algorithms use reports of past experience of the user in question and reports of others. The key difficulty in making a recommendation is the diversity of opinions regarding objects. There are many possible sources for such diversity, e.g., differing personal tastes of humans, different users may not have the same objects accessible to them, some users may be dishonest, etc. Even when no inherent diversity appears to exist, various time-variable factors (such as noise, weather, mood) may create diversity as a side effect. The main challenge in recommendation systems is how to overcome diversity: Intuitively, a set of users with similar preferences should be able to collaborate (perhaps implicitly) by sharing the load of exploring the object space on one hand, and benefit from sharing the results of their experience on the other hand. The difficulty is that users do not know a priori which of the other users share their opinions.

Intuitively, it appears that arbitrary diversity is unmanageable, and one has to make strong assumptions in order come up with algorithmic results for recommendation systems. Indeed, most existing approaches restrict diversity somehow, e.g., by assuming that most users fall in one of a few "well-separated types," or by assuming a simple stochastic generative model for user preferences. Such assumptions are hard to justify, but superficially, they seem unavoidable. In this paper we present novel algorithmic tools that show that effective (near optimal) collaboration by cooperative agents is possible even with unrestricted diversity. Our guarantees are relative rather than absolute: the quality of a recommendation to a user depends on the number of users with similar opinions. In other words, without relying on any assumption about user preferences, our algorithm provides high-quality recommendations to users whose preferences are close to the preferences of many other users, while esoteric users will receive lower-quality recommendations.

Let us first briefly describe the model we study. User preferences are represented by a matrix where rows represent users, and columns represent objects. An entry (i, j) represents the opinion of user i about object j. It is assumed that user opinions are fixed (i.e., do not change over time), but are unknown to the users at the beginning. In this paper we consider the goal of reconstructing the preference matrix. To facilitate information sharing, it is assumed that the system maintains a shared *billboard* where all past experience is reported (similar, say, to eBay's feedback records, where users post the results of their transactions). We distinguish between two variants of recommendation systems: interactive and non-interactive, which differ in the way data is assumed to be obtained from the users. More specifically, these models are as follows.

Interactive recommendation system. In this model, the basic step of an algorithm is to reveal a grade (user preference) of a user for an object, where the user and the object are chosen by the algorithm. Revealing a grade models the action of a user testing a product, called *probing.* Each probe is assumed to incur cost, and the goal is to minimize the worst-case cost to a user [4, 6]. This is the model we shall consider in this paper. To illustrate this model, consider advertisement placement. Probing takes place each time the advertiser provides a user with an advertisement for some product: if the user clicks on this advertisement, the appropriate matrix entry is set to 1, and if the user does not click, it is set to 0. In any case, the corresponding matrix entry is revealed. Many scenarios, for example tracking a dynamic environment by unreliable sensors, or estimating message latencies in a peer-to-peer network [1] fall under this "interactive" framework.

Non-interactive recommendation system. This model received much attention, see, e.g., [10, 11]. Intuitively, while the main question algorithms for the interactive model have to answer is *what to sample*, the main question algorithms for non-interactive model answer is *how to interpret* given results of prior sampling. Some assumptions must be made about the known samples in the non-interactive model (on top of the assumptions about the preferences). Typically, it is assumed that the matrix is generated by a low-entropy random process, and that the given probe results are generated by some probability distribution which reflects the users' preferences.

1.1 Problem Statement and Results

In this paper we focus on the interactive model, and present a solution with polylogarithmic cost to interactive recommendation systems that finds all preferences with precision comparable to the best possible for the given probing budget, while making no assumptions on user preferences. Our execution model is synchronous, i.e., time progresses in global steps called *rounds*, where in each round each player makes a probe. In this model, we use "time" and "cost" interchangeably.

Statement of the problem. There are n *players* and m *objects*; each player has an unknown 0/1 grade for each object. The algorithm proceeds in parallel rounds: in each round, each player reads the shared billboard, probes one object, and writes the result of the probe on the billboard. The task of the algorithm is for each player to output a vector as close as possible to that player's original preference vector (under the Hamming distance metric). Formally, we have the following input-output relation.

Definition 1.1 (Problem *Find_Preferences***)** *Let* $\text{dist}(x, y)$ *denote the Hamming distance between* x *and* y*.*

- ***Input:*** *A set* $\mathcal{P}$ *of* n players*, and a vector* $v(p) \in \{0,1\}^m$ *for each player* p*.*
- ***Output:*** *Each player* p *outputs an estimate* $w(p) \in \{0,1\}^m$*.*
- ***Goal:*** *Minimize* $\text{dist}(w(p), v(p))$ *for each player* p*.*

We note that entries of the input vector $v(p)$ can be revealed only by player p (using probes), but the output vector $w(p)$ is posted on the billboard and is readable by all players.

Evaluating the algorithms. Intuitively, our goal is as follows. Call a set of players with similar preferences a virtual community (for some measure of similarity). We would like to show that even in the worst case, players who are members of a sufficiently large virtual community need only a small number of samples to reconstruct (approximately) all their preferences. Obviously, the larger is the community we are considering, the more leverage we get from other members of that community, and thus preference reconstruction for this community is faster. On the other hand, the larger is the community, the larger are internal disagreements between members of the community, and the larger is the error. Our approach is that the probing budget defines the size of the community. For example, a linear probing budget of m means that the player can "go it alone"and probe all objects, while constant (or poly-logarithmic) probing budget means that the player must leverage probes of a large community. Naturally, it is best to pick the tightest (smallest radius) community of the required size. This could have been easily accomplished if each player had access to an oracle that produces a list of players in decreasing similarity of taste, but building such an oracle is, essentially, the task we wish to accomplish. In fact, our algorithm can continuously reconstruct all such sub-communities in parallel, refining clusterings on-the-fly, as time goes on and probing budget is increasing.

What is the best possible output for a given probing budget, i.e., number of probing rounds? Consider first an ideal situation: If all players had *identical* preference vectors, then by dividing the workload equally, all of them can output perfect results in $O(m/n)$ time units. More generally, if the disparity (i.e., Hamming distance) between all preference vectors were bounded by some known value D, then it can be shown that the players can output a vector with $O(D)$ errors within $O(m/n)$ time, and then reach full accuracy after time of m. The latter case corresponds to the following scenario. There exists a subset of players P^* of cardinality n^* and internal disparity d^*. Imagine that these players are perfectly coordinated (in particular, each of them knows the identities of all members in the set), and their common goal is to find their preference vectors as efficiently as possible. The best one could hope for in general is that they can reach disparity of $O(d^*)$ within $\tau = m/n^*$ rounds. This consideration leads us to define the following concept. Given a time bound τ, and a set of players P^*, we define the *set stretch* of P^* as the ratio between the maximal current number of errors of a player in P^* and the diameter of the set of preference vectors of P^* (using the Hamming distance measure).

More formally, given vectors of equal length, let $\mathrm{dist}(x, y)$ denote the number of coordinates in which they differ (i.e., the Hamming distance between them). Recall that for a player p, $v(p)$ denotes his input vector and $w(p)$ denotes his output vector. Let $w^t(p)$ denote the output of player p at time t. Now, for an arbitrary subset $P^* \subset \mathcal{P}$ and a given time step t, define

$$\begin{array}{rcll} D(P^*) & \stackrel{\text{def}}{=} & \max\{\mathrm{dist}(v(p), v(q)) \mid p, q \in P^*\} & \qquad \textit{diameter} \\ \Delta^t(P^*) & \stackrel{\text{def}}{=} & \max\{\mathrm{dist}(w^t(p), v(p)) \mid p \in P^*\} & \qquad \textit{discrepancy} \\ \rho^t(P^*) & \stackrel{\text{def}}{=} & \frac{\Delta^t(P^*)}{D(P^*)} & \qquad \textit{stretch} \end{array}$$

Using the definition of stretch, we state our result.

Theorem 1.1 (Main result) *Suppose that $m = \Omega(n)$. Let P^* be any set of players with $|P^*| = \Omega(n)$. Then there exists a distributed algorithm such that with probability $1 - n^{-\Omega(1)}$, after $t = (\log n)^{O(1)}$ rounds the output of each player in P^* has constant stretch, i.e., $\rho^t(P^*) = O(1)$.*

A more precise statement is given later (Theorem 5.4). Intuitively, we show that users can predict their preferences for objects they never tried, with confidence that grows with the number of probes executed. The absolute quality of the results depends on how esoteric are the preferences of the user, but constant stretch can be attained in polylogarithmic time. We note that previous results either made strong assumptions about the input, or forced polynomial cost on all or some users.

Our techniques and paper organization. We present our solution in increasing order of complexity. Algorithm Zero_Radius, described in Section 3.2, solves the problem for the special case of communities of users with identical preferences. This algorithm and its proof are quite simple; it is a modification of work published in [3]. In Section 4 we present Algorithm Small_Radius, which uses Zero_Radius as a subroutine. Algorithm Small_Radius works for any distribution of the preference vectors, but the probing cost in Small_Radius is polynomial in the diameter of the collaborating set. This property makes Small_Radius suitable for the case where many users have close preferences. We note that straightforward recursion does not work, because (similarly to metric embedding problems) the error grows exponentially with the depth of the recursion. The crux to the efficiency of this algorithm is a non-trivial combinatorial result that we prove in Lemma 4.1. In Section 5, we present Algorithm Large_Radius, which uses both Small_Radius and Zero_Radius as subroutines, and brings down probing cost to poly-logarithmic in the diameter of the collaborating set. It reduces general instances to the zero- and low-diameter case, by first partitioning the object set and then clustering the subsets. All the above algorithms assume that the size and diameter of the collaborating set are known; this assumption is removed in Section 6. Some related work is surveyed in Section 2, and the high-level algorithm with some basic building blocks are described in Section 3.

2 Related work

Interactive model. Our paper essentially generalizes and improves much of the vast amount of the existing work on multi-agent learning and interactive collaborative filtering. The first algorithmic theory approach is due to Drineas et al. [6], who defined the model we use in this paper. The goal in [6] is to provide a single good recommendation to each user, but the algorithm in fact reproduces the complete preferences of most users. The basic idea is to adapt the SVD technique to the competitive model; this adaptation comes at the price of assuming further restrictions on the preference vectors. Specifically, in addition to assuming the existence of a big gap between two consecutive singular values of the preference matrix (which is inherent to the SVD technique), the algorithm of [6] requires that the preference vectors of users belonging to different types are nearly orthogonal, and that the allowed noise is tiny: each preference vector is obtained by its corresponding canonical vector plus a random noise, which is a vector of m independent random variables with 0 mean and $O(1/(m+n))$ variance. Recently, it has been shown that the problem of finding a good object for each user can be solved by very simple combinatorial algorithms without any restriction on the preference vectors [4]: for any set P of users with a common object they all like, only $O(m + n \log |P|)$ probes are required overall until all users in P will find a good object (w.h.p.). The result closest to our work is [3], where algorithms are given for the case where many users have identical preference vectors (see Section 3.2). We note that when the preference matrix is arbitrary, the case where user preferences may be concentrated in sets of positive diameter is much harder than dealing with sets of diameter 0.

Non-interactive model. The effectiveness of provable algorithms in the non-interactive model relies on the (usually, implicit) assumption that most user preference vectors can be approximated by a low-rank matrix; basically, this assumption means that there are a few (say, constant) "canonical" preference vectors such that most user preference vectors are linear combinations of the canonical vectors. Under this assumption, algebraic or clustering techniques can reconstruct most preference vectors with relatively few errors, based on the scarce available data.

Specifically, there are systems that use principal component analysis [7] or singular value decomposition (SVD) [13]. Papadimitriou et al. [12], and Azar et al. [5] rigorously prove conditions under which SVD is effective. It turns out that SVD works well when there exists a very significant gap between the kth and the $(k+1)$st largest singular values, where k is the number of canonical vectors in the underlying generative model.

Other generative processes that were considered in the passive model include simple Markov chain models [10, 11], where users randomly select their "type," and each type is a probability distribution over the objects.

Worst-case (non-stochastic) input is considered in the works by Goldman et al. [8], [9] where, the algorithm is requested to learn a relation represented as a 0/1 matrix. In a basic step, the algorithm must *predict* the value of an entry in the matrix; then that entry is revealed, and the algorithm is charged one unit if the prediction was wrong. By contrast, our model we require that prediction becomes perfect after small number number of errors.

Our model charges one unit every time a grade is revealed; moreover, a prediction algorithm gets to know the true answer regardless of whether the prediction is correct, while in our model, most estimates are never exposed. Assuming random sampling pattern and this (much weaker) performance measure, the algorithms in [8, 9] still suffer from *polynomial* overhead (which might be best possible under the circumstances) even in the simple "noise-free" case where all the players in a large (constant fraction) community are identical.

3 The High-Level Algorithm and Basic Building Blocks

Simplifying assumptions and notation. Throughout the description of the algorithm, we shall consider a set P^* of players with "similar taste." Formally, we assume that there are two parameters $\frac{\log n}{n} \le \alpha \le 1$ and $D \ge 0$, such that $|P^*| \ge \alpha n$ (i.e., $|P^*| \ge \log n$), and $D(P^*) \le D$. P^* is called an (α, D)-*typical set*, and its members are (α, D)-*typical players*, or just *typical*, when α and D are clear from the context. In general, there may be multiple, overlapping typical sets of typical players.

For the most part, we describe an algorithm that works with known α and D. This assumption is lifted in Section 6. To simplify the description, we also assume, without loss of generality, that $m = \Theta(n)$ (if $m < n$ we can add dummy objects, and when $m > n$ we can let each real player simulate $\lceil m/n \rceil$ players of the algorithm.) If the assumption that $\frac{\log n}{n} \leq \alpha$ does not hold then the player is better off by just probing all objects on his own.

The main algorithm. Our solution consists of three algorithms, depending on value of D, as specified in Fig. 1. Algorithms Zero_Radius, Small_Radius and Large_Radius are specified in Sections 3.2, 4 and 5, resp.

(1) If $D = 0$ apply procedure Zero_Radius with all players and all objects, using known α.

(2) If $D = O(\log n))$ apply procedure Small_Radius with all players and all objects, using known α.

(3) Otherwise (i.e., $D \geq \Omega(\log n)$), apply procedure Large_Radius, using known α and D.

Figure 1: Main algorithm for known α and D (see Section 6).

3.1 The *Choose_Closest* problem: Algorithm Select

One of the basic building blocks we use is a little algorithm solving a problem which can be formulated as follows.

Definition 3.1 (Problem *Choose_Closest***)**

- ***Input:*** *a set V of k vectors and a player p with preference vector $v(p)$.*
- ***Output:*** *a vector $w^* \in V$ such that* $\mathrm{dist}(w^*, v(p)) \leq \mathrm{dist}(w, v(p))$ *for all $w \in V$.*

The algorithm we describe below requires an additional input parameter:

- ***Additional input:*** *A* distance bound $D \geq 0$ *such that for some $w \in V$,* $\mathrm{dist}(w, v(p)) \leq D$.

Given D, this task can easily be implemented by player p at the cost of probing $k(2D+1)$ coordinates; we present a slightly more efficient algorithm in Figure 2. The algorithm uses the following notation.

Notation 3.2 *For given vectors $v, u \in \{0, 1, \star\}^m$, $\tilde{d}(u, v)$ denotes the number of differing coordinates in which both u and v have entries that are not $\star$. $\tilde{d}_I(v, u) \overset{\text{def}}{=} \tilde{d}(v|_I, u|_I)$ is the restriction of $\tilde{d}$ to the coordinate set I.*

The algorithm uses an abstract `Probe` action that, when invoked by a player $p \in P$ on an object $o \in O$, returns the value of o for p. Its properties are summarized in the following theorem.

Theorem 3.1 *If V contains a vector at distance at most D from $v(p)$, then Procedure* Select *outputs the lexicographically first vector in V among the vectors closest to $v(p)$. Moreover, the total number of times* `Probe` *is invoked is (unconditionally) never more than $k(D+1)$.*

```
(1) Repeat
    (1a) Let $X(V)$ be set of coordinates on which some two vectors in $V$ differ.
    (1b) Execute Probe on the first coordinate in $X(V)$ that has not been probed yet.
    (1c) Remove from $V$ any vector with more than $D$ disagreements with $v(p)$.
    Until all coordinates in $X(V)$ are probed or $X(V)$ is empty.
(2) Let $Y$ be the set of coordinates probed by $p$ throughout the algorithm. Find the set of vectors
    $U \subseteq V$ closest to $v(p)$ on $Y$, i.e.,
        $U = \left\{ v \in V \mid \forall u \in V : \tilde{d}_Y(v, v(p)) \leq \tilde{d}_Y(u, v(p)) \right\}$ .
    Output the lexicographically first vector in $U$.
```

Figure 2: Algorithm Select using distance bound D, executed by player p.

Proof: Any vector removed from V in Step 1c is at distance more than D from $v(p)$. Among the vectors remaining in V in Step 2, all distinguishing coordinates were probed, so their distances from $v(p)$ are precisely computed, up to a common additive term. Therefore, the output made in Step 2 is trivially correct by assumption that the closest vector is at distance at most D. To bound the total number of probes in Select, consider the total number of disagreements between $v(p)$ and all vectors of the input set V. By definition of X, each probe exposes at least one such disagreement. Since no vector remains in V after finding $D+1$ coordinates on which it disagrees with $v(p)$, we get that the total number of probes is at most $k(D+1)$. ▮

Remark: To ensure that the result of Select is completely defined by its input, we require that Select disregards probes done before its execution.

3.2 Exact types solution: Algorithm Zero_Radius

Below we present, for completeness, an algorithm for the special case of $D = 0$, i.e., the case where typical players completely agree on all coordinates. This task is carried out by Algorithm Zero_Radius. The algorithm, presented in Figure 3, is a slight generalization of an algorithm given in [3]. In the variant we use here, the algorithm uses the abstract `Probe` action. Another slight generalization is that the set of allowed values for an object is not necessarily binary. The implementation of `Probe` depends on the context: for simple objects we use the primitive probing action as an implementation; for compound objects (defined within the context of algorithm Large_Radius in Section 5), `Probe` is implemented by Select with an appropriate distance bound.

For this algorithm, and using Theorem 3.1, we have the following result (see [3] for full details).

Theorem 3.2 *Suppose that there are at least αn players with identical value vectors, and that they all run Algorithm* Zero_Radius*. Then with probability $1 - n^{-\Omega(1)}$ all of them output the correct vector. The algorithm terminates unconditionally after $O(\frac{\log n}{\alpha} \lceil \frac{m}{n} \rceil)$ invocations of procedure* `Probe`.

(1) If $\min(|P|,|O|) < \frac{8c\ln n}{\alpha}$, player p invokes `Probe` for all objects in O, outputs their values, and returns. ($c > 0$ is a constant controlling the probability of success.)

(2) (Otherwise) Partition randomly $P = P' \cup P''$ and $O = O' \cup O''$ (each player is assigned with probability $1/2$ to P' and probability $1/2$ to P'', and similarly with the objects). The partition is known to all players in P. Let P' be the half that contains p, and let P'' be the other half.

(3) Recursively execute Zero_Radius(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 $\alpha/2$ fraction of the players in P''. Compute Select on V with distance bound 0. Output the result vector for all objects in O'' and return. (If $V = \emptyset$, Zero_Radius fails.)

Figure 3: Algorithm Zero_Radius executed by player p. P is the set of players and O is the set of objects.

Proof Sketch: (adapted from [3].) Let P_T denote the set of players with identical preference vectors. Using the Chernoff bound (see, e.g., [2], Appendix A) and the union bound, it is straightforward to show that in each invocation of Zero_Radius with $|P| \geq \frac{8c\ln n}{\alpha}$, we have, with probability at least $1 - n^{-\Omega(1)}$, that $|P \cap P_T| \geq \alpha|P|/2$. Therefore, barring events of very low probability, the output is proved correct by induction on the level of recursion and by correctness of Algorithm Select.

To analyze the cost, we note that probes are done in Step 1 (the basis of the recursion) and Step 4 (recursive step). Step 1 is executed at most once by each player, and its cost is $O\left(\lceil\frac{m}{n}\rceil\frac{\log n}{\alpha}\right)$ invocations of `Probe` per player: this is true because the recursive halving maintains $|O| \approx |P| \cdot \frac{m}{n}$, so if $n < m$ we have that the recursion stops when $|P| = O(\frac{\log n}{\alpha})$, in which case $|O| = O(\frac{m}{n} \cdot \frac{\log n}{\alpha})$; and when $n \geq m$, each the recursion stops when $|O| = O(\frac{\log n}{\alpha})$, and therefore each player probes $O(\frac{\log n}{\alpha})$ objects. Consider now other recursive calls of Zero_Radius. Each such invocation entails a call to Select with $O(1/\alpha)$ candidate vectors and distance bound 0, for a total cost of $O(1/\alpha)$ invocations of `Probe` by Theorem 3.1. Since the depth of the recursion is upper bounded by $O(\log n)$, we have that the total number of invocations of `Probe` done in Step 4 by each player, throughout the execution of the algorithm, is upper bounded by $O\left(\frac{\log n}{\alpha}\right)$. ▮

4 Algorithm Small_Radius

In this section we describe Algorithm Small_Radius. We assume that α and D are given, and the goal is that all these players will output a vector which differs from their input vector by at most $O(D)$. The running time of the algorithm is polynomial in D, and hence it is suitable only for small D values (in the main algorithm, Algorithm Small_Radius is invoked with $D = O(\log n)$).

The algorithm proceeds by repeating the following process K times (we always set $K = O(\log n)$): The object set O is partitioned into $s = O(D^{3/2})$ random parts denoted O_i, and all players run

```
(1) For each $t \in \{1, \dots, K\}$ do:
    (1a) Partition $O$ randomly into $s = 100D^{3/2}$ disjoint subsets: $O = O_1 \cup O_2 \cup \cdots \cup O_s$.
    (1b) For each $i \in \{1, \dots, s\}$: all players apply procedure Zero_Radius to the objects of $O_i$ using
         parameter $\alpha/5$; let $U_i$ be the set of vectors s.t. each is output by at least $\alpha n/5$ players.
    (1c) Each player $p$ applies procedure Select to $U_i$ with distance bound $D$, obtaining vector
         $u_i(p)$ for each $i \in \{1, \dots, s\}$. Let $u^t(p)$ denote the concatenation of $u_i(p)$ over all $i$.
(2) Each player $p$ applies procedure Select with distance bound $5D$ to the vectors $u^1(p), \dots, u^K(p)$
    computed in Step 1c and outputs the result $w(p)$.
```

Figure 4: Algorithm Small_Radius. α and D are given, K is a confidence parameter.

Algorithm Zero_Radius on each O_i object set (Step 1b). However, Algorithm Zero_Radius is guaranteed to succeed only if there are sufficiently many players that fully agree. To this end, we show that with constant probability, a random partition of O will have, in all O_i parts simultaneously, many (but not all) typical players *fully agreeing*. Therefore, one of the K independent executions of the exact algorithm will succeed in all parts with probability at least $1 - 2^{-\Omega(K)}$. However, in each part there may be many typical players whose preferences are not shared by many others *exactly*, and may therefore have arbitrary results in that part, because Theorem 3.2 does not apply in that case. To solve this problem, in Step 1c we force each player to adopt, for each i, the closest of the popular vectors in O_i. Then, in Step 2, each player chooses the closest result among the vectors produced in the K iterations. Since the typical players differ on the O objects, they will not all choose the same vector in Step 1c; however, we prove that all their chosen vectors lie within $O(D)$ distance from each other.

4.1 Analysis of Small_Radius

We now state the properties Algorithm Small_Radius. There are a few points which are not obvious. First, in Step 1b we use Algorithm Zero_Radius which is guaranteed to work only if there are at least $\alpha n/5$ players who completely agree on all objects. It turns out that for $s = O(D^{3/2})$, there is a constant probability that all instances of Zero_Radius are successful in any given iteration. The following lemma proves this crucial fact in more general terms.

Lemma 4.1 *Let V be a set of M binary vectors on a set O of coordinates, and suppose that $\mathrm{dist}(v, v') \le d$ for any $v, v' \in V$. Let $O = O_1 \cup O_2 \cup \cdots \cup O_s$ be a random partition of O into s pairwise disjoint sets, where each coordinate $j \in O$ is chosen, randomly and independently, to lie in a uniformly chosen O_i. Call the partition successful if for every $i \in \{1, \dots, s\}$ there is a set $U_i \subset V$ of size $|U_i| \ge M/5$, and such that $u|_{O_i} = u'|_{O_i}$ for all $u, u' \in U_i$. Then, the probability that the partition is not successful is at most $\frac{10^3 \cdot 5^5}{6!} \frac{d^3}{s^2}$. In particular, if $s \ge 100d^{3/2}$ then this probability is smaller than $1/2$.*

Proof: Let X be the random variable whose value is the number of ordered 6-tuples $(i, v_1, v_2, v_3, v_4, v_5)$, where $1 \leq i \leq s$, $v_1, \ldots, v_5 \in V$, and

$$\text{for each } 1 \leq j < k \leq 5, \text{ the vectors } v_j, v_k \text{ differ on } O_i. \tag{1}$$

For a fixed i, $1 \leq i \leq s$, and for fixed distinct vectors $v_1, \ldots, v_5 \in V$, the probability that the tuple $(i, v_1, v_2, ..., v_5)$ satisfies (1) can be bounded as follows. Note, first, that there are at most $\binom{5}{2}d = 10d$ coordinates in which some pair of the vectors v_j differ. In order to satisfy (1), O_i has to contain at least 3 such coordinates (as, by the pigeonhole principle, at least two of the vectors will agree on each pair of coordinates). Therefore, the required probability is at most

$$\binom{10d}{3}\frac{1}{s^3} < \frac{10^3 d^3}{6s^3} .$$

By linearity of expectation, the expected value of X satisfies

$$E(X) \leq sM^5 \cdot \frac{10^3 d^3}{6s^3} = \frac{10^3 M^5 d^3}{6s^2}.$$

On the other hand, if there exists some i such that no set of $M/5$ of the vectors completely agree on O_i, then the number of ordered 5-tuples of vectors $v_1, \ldots, v_5$ so that each pair of them differs on O_i is at least

$$M \cdot \frac{4M}{5} \cdot \frac{3M}{5} \cdot \frac{2M}{5} \cdot \frac{M}{5} = \frac{4!\ M^5}{5^4}.$$

It follows that if the partition is not successful, then the value of the random variable X is at least $\frac{4!\ M^5}{5^4}$, and hence, by Markov's Inequality, the probability this happens does not exceed

$$E(X)/\frac{4!\ M^5}{5^4} \leq \frac{10^3 \cdot 5^5}{6!}\frac{d^3}{s^2}. \qquad \blacksquare$$

To deal with our case, let us first introduce the following standard notation.

Notation 4.1 *Given a vector v and a subset $S \subset \mathcal{O}$ of coordinates, let $v|_S$ denote the projection of v on S. Similarly, let* $\mathrm{dist}|_S$ *denote the Hamming distance applied to vectors projected on S.*

Lemma 4.1, applied to our setting with $M = \alpha n$, implies the following immediate corollary.

Corollary 4.2 *For $s = \Theta(D^{3/2})$, the following holds with probability at least $1 - 2^{-\Omega(K)}$ after the execution of Step 1 of Algorithm* Small_Radius*: there exists an iteration $t_0 \in \{1, \ldots, K\}$ in which for each $i \in \{1, \ldots, s\}$ there exists a set of players $G_i \subseteq P^*$ satisfying $|G_i| \geq \alpha n/5$ and $v(p)|_{O_i} = v(p')|_{O_i}$ for any $p, p' \in G_i$.*

By the correctness condition of Algorithm Zero_Radius, Corollary 4.2 implies that after Step 1b of Algorithm Small_Radius, w.h.p., there exists an iteration t_0 such that for each $i \in \{1, \ldots, s\}$ there exists a vector $u_i^{t_0}$ which is identical to the vector of all players in a set $G_i \subseteq P^*$ with $|G_i| \geq \alpha n/5$. However, the G_is may be different for each i. Moreover, note that it is possible to have more than one such G_i for any given part O_i. In Step 1c, the algorithm "stitches" a vector u^t for O from the u_i^t components for O_i. We can now prove that in a successful iteration, *any* vector produced in Step 1c by a typical player is close to the vectors produced by *all* typical players.

Lemma 4.3 *Consider a partition $O_1, \ldots, O_s$ of O. Suppose that for each $i \in \{1, \ldots, s\}$ there exists a vector u_i and a set $G_i \subseteq P^*$ with $|G_i| \geq \alpha n/5$ such that $v(p)|_{O_i} = u_i$ for any $p \in G_i$. Let u be any vector satisfying $u|_{O_i} = u_i$ for all $i \in \{1, \ldots, s\}$. Then $\mathrm{dist}_O(u, v(p)) \leq 5D$ for any player $p \in P^*$.*

Proof: Fix a value vector v^* of a player in P^*. We count the sum of the distances from the vectors of the players in P^* to v^* in two different ways. First, by the precondition of Algorithm SMALL_RADIUS,

$$\sum_{p \in P^*} \mathrm{dist}_O(v(p), v^*) \leq |P^*| \cdot D \ . \tag{2}$$

On the other hand,

$$\begin{aligned}\sum_{p \in P^*} \mathrm{dist}_O(v(p), v^*) &= \sum_{p \in P^*} \sum_{i=1}^{s} \mathrm{dist}_{O_i}(v(p), v^*) \\ &\geq \sum_{i=1}^{s} \sum_{p \in G_i} \mathrm{dist}_{O_i}(u_i, v^*) \\ &\geq \sum_{i=1}^{s} \frac{|P^*|}{5} \cdot \mathrm{dist}_{O_i}(u_i, v^*) \\ &= \frac{|P^*|}{5} \cdot \mathrm{dist}_O(u, v^*) \ . \end{aligned} \tag{3}$$

Eqs. (2,3) together imply $\mathrm{dist}_O(u, v^*) \leq 5D$. ▮

We summarize the properties of Algorithm SMALL_RADIUS as follows:

Theorem 4.4 *Suppose that there exists a set P^* of at least αn players such that $\mathrm{dist}(v(p), v(p')) \leq D$ for any $p, p' \in P^*$. Let $w(p)$ be the output vector of player $p \in P^*$ after running Algorithm* SMALL_RADIUS. *Then with probability at least $1 - 2^{-\Omega(K)}$, $\mathrm{dist}_O(v(p), w(p)) \leq 5D$ for every $p \in P^*$. Furthermore, the total number of probing rounds is $O\left(K \frac{m}{\alpha n} D^{3/2}(D + \log n)\right)$.*

Proof: By Corollary 4.2, with probability at least $1 - 2^{-\Omega(K)}$ at least one of the iterations satisfies the premise of Lemma 4.3. Using Theorem 3.1 (correctness of SELECT), the correctness claim follows. Regarding complexity, consider a single iteration of Step 1. In such an iteration, procedure ZERO_RADIUS is invoked $s = O(D^{3/2})$ times, each time with all n users and m/s objects. By Theorem 3.2, the cost of a single invocation of ZERO_RADIUS in Step 1b is $O(\lceil \frac{m/s}{n} \rceil \cdot \frac{\log n}{\alpha})$. It follows that the total cost incurred by Step 1b in a single iteration is

$$O\left(s \cdot \left\lceil \frac{m/s}{n} \right\rceil \cdot \frac{\log n}{\alpha}\right) = O\left((\frac{m}{n} + D^{3/2}) \cdot \frac{\log n}{\alpha}\right) \ .$$

In addition, in Step 1c, each iteration contains s applications of SELECT, each time with a bound D and at most $O(1/\alpha)$ candidates, totaling $O(D^{5/2}/\alpha)$ probes in each iteration. Therefore the total cost of Step 1 throughout the execution of SMALL_RADIUS is upper bounded by

$$O\left(K \cdot \left(\frac{m}{n} + D^{3/2}) \cdot \frac{\log n}{\alpha} + (D^{5/2}/\alpha)\right)\right) \leq O\left(K \frac{m}{\alpha n} D^{3/2}(\log n + D)\right) \ .$$

Since Step 2 entails only $O(KD)$ probes, the overall complexity is dominated by Step 1. ▮

(1) Randomly partition the objects into $cD/\log n$ disjoint subsets O_ℓ for $1 \le \ell \le cD/\log n$, where c is an appropriate constant. The partition is done by assigning each object independently and uniformly to one of the object subsets.
Assign randomly the players to $cD/\log n$ subsets P_ℓ for $1 \le \ell \le cD/\log n$. Each player is assigned to $\left\lceil \frac{D}{\alpha n} \right\rceil$ subsets.

(2) For each $\ell \in \{1, \ldots, cD/\log n\}$, the players of P_ℓ apply procedure Small_Radius to objects O_ℓ with frequency parameter $\alpha/2$ and confidence parameter $K = O(\log n)$.
Let $v_\ell(p)$ denote the output of a player $p \in P_\ell$ on O_ℓ.

(3) All players apply procedure Coalesce to each of the sets of vectors $\{v_\ell(p) \mid p \in P_\ell\}$ produced in Step 2.
The result of this step is, for each O_ℓ, a set B_ℓ of at most $O(1/\alpha)$ vectors of $\{0, 1, \star\}^{(m \log n)/(cD)}$.

(4) Apply procedure Zero_Radius with all players, where each "object" for the algorithm is a set O_ℓ of primitive objects (see Step 1), with possible values from the B_ℓ vectors (computed in Step 3). Flatten the resulting vector of vectors to obtain the final output.

Figure 5: Algorithm Large_Radius for known α, D.

5 Algorithm Large_Radius

In this section we assume that α and D are known. Algorithm Large_Radius, presented in Figure 5, deals with the case of $D > \log n$, and it uses, as subroutines, Algorithms Zero_Radius and Small_Radius.

The main idea in Algorithm Large_Radius is to transform the input instance so that we can apply Algorithm Zero_Radius to the new instance. To do that, we need to somehow aggregate objects in a way that all players of P^* will agree on the result of each aggregate, and to implement probing an aggregate efficiently. Specifically, we do it as follows.

The algorithm starts (in Step 1) by randomly chopping the object set into small parts denoted O_ℓ and the player set into corresponding parts denoted P_ℓ. The number of parts is such that w.h.p., the distance between any two (α, D)-typical players on the objects of O_ℓ is bounded by $O(\log n)$. In Step 2, each player set P_ℓ applies procedure Small_Radius to the object set O_ℓ. When all invocations of procedure Small_Radius return, each player in P_ℓ has a complete output vector for O_ℓ, and, w.h.p., the output vectors of any two (α, D)-typical players from P_ℓ differ in only $O(\log n)$ coordinates of O_ℓ. Relying on this property, in Step 3 we run a basic clustering algorithm called Coalesce on the results for O_ℓ. The outcome of Algorithm Coalesce, for each object set O_ℓ, is a collection B_ℓ of only $O(1/\alpha)$ possible value vectors ("candidates"), such that for each ℓ, there is exactly one candidate which is the closest to *all* typical players on O_ℓ. This comes at the price of possibly blowing up the radius of each candidate from 0 (a simple binary vector) to $O(\log n/\alpha)$, represented by a vector with "don't care" entries (denoted $\star$). This key property allows us to apply Algorithm Zero_Radius in Step 4 by all

```
(1) $A \leftarrow \emptyset$.
(2) While $V \neq \emptyset$ do
    (2a) Remove from $V$ all vectors $v$ with $|\mathrm{ball}(v, D) \cap V| < \alpha n$.
    (2b) Let $v$ be the lexicographically first vector $v \in V$.
    (2c) $A \leftarrow A \cup \{v\}$; $V \leftarrow V \setminus \mathrm{ball}(v, D)$.
(3) Let $B \leftarrow A$.
(4) While there are two distinct vectors $v, v' \in B$ with $\tilde{d}(v, v') \leq 5D$ do:
    (4a) Define a vector $v^*$ by $v, v'$ as follows: If $v$ and $v'$ have the same value for an object $j$, let
         the value of $v^*$ for $j$ be their common value. If $v$ and $v'$ disagree on $j$, let the value of $v^*$
         for $j$ be $\star$.
    (4b) $B \leftarrow B \setminus \{v, v'\} \cup \{v^*\}$.
(5) Output $B$.
```

Figure 6: Algorithm Coalesce.

players, where the "objects" are actually complete O_ℓ sets, and the possible values for each such object are the B_ℓ vectors computed in Step 4. The abstract `Probe`(O_ℓ) action used in Zero_Radius in this case is implemented by applying Select to the vector set B_ℓ, with distance bound $O(\log n)$. When the algorithm ends, any two typical players will have the same output vector, which may include up to $O(\frac{D}{\alpha})$ "don't care" entries (if the output must be binary, we may set $\star$ entries arbitrarily).

We first present Algorithm Coalesce (Step 3).

5.1 Algorithm Coalesce

The problem we solve here is the following. It is a pure clustering problem: all input vectors are readable by all users.

- **Input:** A multiset V of n vectors, each in $\{0,1\}^m$; a distance parameter D; a frequency parameter α.
- **Output:** A set U of at most $1/\alpha$ vectors from $\{0,1,\star\}^m$.

The requirement is that if there exists a subset $V_T \subseteq V$ of size at least αn satisfying $\mathrm{dist}(v, v') \leq D$ for all $v, v' \in V_T$, then there exists a unique vector $v^* \in U$ such that (1) v^* is the closest in U to any vector in V_T, and (2) the number of $\star$ coordinates in v^* is small (specifically at most $5D/\alpha$).

Note that this problem does not involve probing at all and hence in our case, all players have the same input. Pseudo code of the algorithm to solve this problem is presented in Figure 6. It uses the notation $\mathrm{ball}(v, D) \overset{\text{def}}{=} \{u \mid \tilde{d}(v, u) \leq D\}$ to denote the ball in the distance metric $\tilde{d}$ which ignores coordinates with $\star$ entries (see Notation 3.2).

To analyze Algorithm Coalesce, we use use the following concept. For each vector removed from B in Step 4b (denoted v, v' in Figure 6), there is a unique vector that is added to B (denoted v^* there). Extending this relation transitively in the natural way, we define for each vector $v \in A$ a vector $\mathrm{rep}(v)$ that appears in the final output set. Using this concept, we have the following lemmas.

Lemma 5.1 *For any input vector $v \in V$ and any $u \in A$, $\tilde{d}(v, \mathrm{rep}(u)) \le \mathrm{dist}(v, u)$.*

Proof: By definition, the $\tilde{d}$ measure ignores $\star$ entries. The lemma follows from the observation that u and $\mathrm{rep}(u)$ agree on all coordinates except the $\star$ coordinates in $\mathrm{rep}(u)$. ▮

Lemma 5.2 *For any $v \in V_T$ there exists a vector u in the output set such that $\tilde{d}(v, u) \le 2D$.*

Proof: Observe first that there must be a vector $v_1 \in A$ such that $\mathrm{ball}(v_1, D) \cap \mathrm{ball}(v, D) \neq \emptyset$: otherwise, the vector v would have been added to A in Step 2 since by assumption, $|\mathrm{ball}(v, D)| \ge \alpha n$. For that vector v_1 we have $\mathrm{dist}(v_1, v) \le 2D$ by the triangle inequality. Therefore, by Lemma 5.1, $\tilde{d}(v, \mathrm{rep}(v_1)) \le 2D$. ▮

We summarize with the following statement.

Theorem 5.3 *The output of Algorithm* Coalesce *contains at most $1/\alpha$ vectors. There is exactly one vector v^* in the output set which is closest to all vectors of V_T, and $\tilde{d}(v^*, v) \le 2D$ for any vector $v \in V_T$. Moreover, the number of $\star$ entries in v^* is at most $5D/\alpha$.*

Proof: Regarding the size of the output set, note that by Step 2, each vector in A represents a disjoint set of size at least αn vectors from a set whose total size is n, and hence B starts at Step 3 with size at most $1/\alpha$; the claim about the size follows, since Step 4 may only reduce the size of B. The distance claim follows from Lemma 5.2. To see uniqueness, suppose that there were vectors $u, u' \in B$ and $v, v' \in V_T$ such that u is the closest to v and u' is the closest to v'. Then by the triangle inequality $\tilde{d}(u, u') \le \tilde{d}(u, v) + \tilde{d}(v, v') + \tilde{d}(v', u') \le 5D$. But by the stopping condition of the while loop of Step 4, $\tilde{d}(u, u') \le 5D$ iff $u = u'$. Finally, regarding the number of $\star$ entries in v^*, note that Step 4 is performed at most $|A| \le 1/\alpha$ times, and each iteration adds at most $5D$ $\star$ entries. ▮

Note that the output of Algorithm Coalesce is deterministic (the order in which vectors are merged in Step 4 is immaterial). Since there is no probing and all players have the same input, all players will have the same output.

5.2 Analysis of Large_Radius

We summarize the properties of the main algorithm (Fig. 1) in the following theorem.

Theorem 5.4 *Suppose that the algorithm is given $0 < \alpha \le 1$ and $D \ge 0$ such that there exists a set of players $P^* \subseteq P$ with $|P^*| \ge \alpha n$ satisfying $\mathrm{dist}(v(p), v(p')) \le D$ for any $p, p' \in P^*$. Then w.h.p., the output vector $w(p)$ of each player $p \in P^*$ satisfies $\mathrm{dist}(w(p), v(p)) = O(D/\alpha)$. The number of probes performed by each player throughout the execution of the algorithm is $O\left(\lceil \frac{m}{n} \rceil \cdot \frac{\log^{7/2} n}{\alpha^2}\right)$.*

To prove Theorem 5.4 we first prove the following immediate properties of the random partitions of Step 1.

Lemma 5.5 *With probability at least $1 - n^{-\Omega(1)}$, the following properties hold for each $1 \le \ell \le cD/\log n$:*

- $|O_\ell| = \Theta(\frac{m \log n}{D})$.
- $|P_\ell| = \Omega(\frac{\log n}{\alpha})$.
- $|P_\ell \cap P^*| = \Theta(\alpha |P_\ell|)$.
- *For any two typical players $p, p' \in P_\ell \cap P^*$, $\mathrm{dist}_{O_\ell}(v(p), v(p')) = O(\log n)$.*

Proof: By the Chernoff bound. For the partition of objects, note that the expected size of O_ℓ is $\frac{m \log n}{cD} = \Omega(\log n)$ since $D \le m$ always; for the partition of players, the expected number of players in P_ℓ is $\frac{n}{cD/\log n} \cdot \left\lceil \frac{D}{\alpha n} \right\rceil = \Omega(\frac{\log n}{\alpha})$. The expected number of typical players in P_ℓ is $\Omega(\log n)$, and the expected number of objects in O_ℓ on which any two typical players differ is at most $\frac{D}{cD/\log n} = O(\log n)$. ▮

Proof of Theorem 5.4: Let $1 \le \ell \le cD/\log n$. By Lemma 5.5, with probability at least $1 - n^{-\Omega(1)}$, there are at least $\Omega(\alpha |P_\ell|)$ players from P^* in P_ℓ, and the distance between any two of them on O_ℓ is at most $\lambda \overset{\mathrm{def}}{=} \min(D, O(\log n))$. It therefore follows from Theorem 4.4 that with probability at least $1 - n^{-\Omega(1)}$, after Step 2 is done, $\mathrm{dist}_{O_\ell}(v_\ell(p), v(p)) \le \lambda$ for any $p \in P^* \cap P_\ell$. Next we note that by Theorem 5.3, after executing Step 3, there exists exactly one vector v_ℓ among all $O(1/\alpha)$ vectors of B_ℓ which is the closest to any player $p \in P^*$, and furthermore, that $\tilde{d}_{O_\ell}(v_\ell, v(p)) \le O(\log n)$. This means that the preconditions for Theorem 3.2 hold, and hence, with probability at least $1 - n^{-\Omega(1)}$, all players in P^* will output the vector composed of the v_ℓ components.

Regarding complexity, note that Steps 1 and 3 do not involve any probing. Consider Step 2. Let us consider Algorithm SMALL_RADIUS in context: denote by n' and m' the number of players and objects (respectively) in the invocation of SMALL_RADIUS. Algorithm SMALL_RADIUS is invoked with $n' = O(\log n/\alpha)$ players and $m' = O(m \log n/D)$ objects in Step 2 of the main algorithm. (We have $\frac{n'}{m'} \ge \frac{n}{m}$). Also, the confidence parameter is $K = O(\log n)$, and distance bound is $D = O(\log n)$. It follows from Theorem 4.4 that after Step 2 of the main algorithm, the vector $w(p)$ adopted by a player $p \in P_\ell$ satisfies $\mathrm{dist}_{O_\ell}(v(p), w(p)) \le O(\log n)$, and that the total number of probing rounds is at most $O\left(\frac{\log n}{\alpha} \log^{3/2} n \log n\right) = O\left(\frac{\log^{7/2} n}{\alpha}\right)$. Next, consider Step 4. Algorithm ZERO_RADIUS is invoked with n players and $D/\log n < n$ objects. Since each logical probe of this invocation consists of $O(1/\alpha)$ primitive probes, we conclude from Theorem 3.2 that the total number of probes per player in this step is $O(\frac{\log n}{\alpha^2})$. ▮

6 Coping with unknown distance bound D

Our main algorithm (Figure 1) so far required knowing D for a given α. We now describe how to extend it to the case of unknown α and D. First, note that for any given α and a player p, there exists a minimal $D = D_p(\alpha)$ such that at least an α fraction of the players are within distance D from p. So suppose for now that α is given and D is not known. In this case we run $O(\log n)$ independent versions of the main algorithm (sequentially or in parallel): in the ith version, it is run with $D_i = 2^i$. We also run another version with $D = 0$. From all $O(\log n)$ resulting output vectors $w(p)$, we select (using procedure *Choose_Closest* described in Section 6.1) the vector that appears to be closest to its input vector $v(p)$ and output it.

The search procedure increases the running time of the algorithm by an $O(\log n)$ factor, and decreases the quality of the output by a constant factor, as compared to the algorithm that assumes known α and D.

Next we discuss how to choose α. The idea is as follows. Let the running time τ be given. From τ, we can compute the smallest value α_τ the algorithm can "afford" for the parameter α without breaking the given time bound. Finally, using repeated doubling (and paying a constant factor increase in the running time), we can lift of the requirement that the running time is given. More specifically, we run the algorithm in phases, where in phase j, we run the algorithm with $\tau = 2^j$ for all possible $O(\log m)$ values of D, and then choose, using RSelect, the best result produced in this phase and output it. This way we obtain an "anytime algorithm," i.e., an algorithm whose output quality at any time t is close to the best possible in t time units.

6.1 Solving *Choose_Closest* without a distance bound

```
1. For any pair of distinct vectors $v, v' \in V$ in turn do:
    (a) Let $X$ be the set of coordinates on which non-$\star$ values for $v$ and $v'$ differ.
    (b) Probe randomly $c \log n$ coordinates from $X$ (if $|X| < c \log n$, probe all coordinates in $X$).
    (c) Declare $v'$ "loser" if it disagrees with $v(p)$ on 2/3 or more of the probed coordinates; declare $v$
        "loser" if it disagrees with $v(p)$ on 2/3 or more of the probed coordinates; otherwise none is declared
        loser.
2. Output any vector with 0 losses.
```

Figure 7: Algorithm RSelect for the *Choose_Closest* problem.

We give an alternative algorithm for solving *Choose_Closest*, which we call below RSelect. RSelect solves the same problem as Select, with a few important differences outlined below.

In Select, a bound D on the distance of $v(p)$ to the set is given as input, and the number of probes is linear in D. In RSelect, no such bound is given, and the number of probes per input vector is $O(\log n)$, irrespective of the distance between the vectors. On the other hand, Select is deterministic and guaranteed to produce the closest vector, while RSelect is randomized, and is only guaranteed w.h.p. to be close to the closest vector.

Pseudo-code for the Algorithm RSelect is given in Figure 7, and its properties are summarized in the theorem below.

Theorem 6.1 *Let V be a set of equal-length vectors over $\{0, 1, \star\}$, and fix a player p. Let $D = \min\left\{\tilde{d}(v(p), v) \mid v \in V\right\}$. With probability at least $1 - n^{-\Omega(1)}$, Algorithm* RSelect *outputs a vector u such that $\tilde{d}(u, v(p)) = O(D)$. The number of probes executed by* RSelect *is $O(|V|^2 \log n)$.*

Proof: The complexity bound is obvious. For correctness, let u_0 be the vector in V which is closest to $v(p)$. By the Chernoff bound, the probability that u_0 loses against any other vector is $1 - n^{-\Omega(1)}$. Therefore there is at least one vector with 0 losses (w.h.p.). Also, if $\tilde{d}(u', v) \geq cD$ for some $u' \in V$, then the probability that u' is declared a loser against u_0 is also $1 - n^{-\Omega(1)}$. Hence only a vector whose distance from $v(p)$ is at most $O(D)$ may have 0 losses. The result follows. ▮

References

[1] I. Abraham, Y. Bartal, T.-H. Chan, K. Dhamdhere, A. Gupta, J. Kleinberg, O. Neiman, and A. Slivkins. Metric embeddings with relaxed guarantees. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, 2005.

[2] N. Alon and J. H. Spencer. *The Probabilistic Method.* Wiley, second edition, 2000.

[3] B. Awerbuch, Y. Azar, Z. Lotker, B. Patt-Shamir, and M. Tuttle. Collaborate with strangers to find own preferences. In *Proc. 17th ACM Symp. on Parallelism in Algorithms and Architectures*, pages 263–269, 2005.

[4] 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.

[5] 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.

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

[7] K. Goldberg, T. Roeder, D. Gupta, , and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval Journal*, 4(2):133–151, July 2001.

[8] 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.

[9] S. A. Goldman and M. K. Warmuth. Learning binary relations using weighted majority voting. *Machine Learning*, 20(3):245–271, 1995.

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

[11] 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.

[12] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proc. 17th ACM Symp. on Principles of Database Systems (PODS)*, pages 159–168. ACM Press, 1998.

[13] 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.