User Rankings from Comparisons: Learning Permutations in High Dimensions

Ioannis Mitliagkas, Aditya Gopalan, Constantine Caramanis, Sriram Vishwanath
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
ioannis@utexas.edu, {gadit,caramanis}@mail.utexas.edu, sriram@austin.utexas.edu

***Abstract*—We consider the problem of learning users' preferential orderings for a set of items when only a limited number of pairwise comparisons of items from users is available. This problem is relevant in large collaborative recommender systems where overall rankings of users for objects need to be predicted using partial information from simple pairwise item preferences from chosen users. We consider two natural schemes of obtaining pairwise item orderings – random and active (or intelligent) sampling. Under both these schemes, assuming that the users' orderings are constrained in number, we develop efficient, low-complexity algorithms that reconstruct all the orderings with provably order-optimal sample complexities. Finally, our algorithms are shown to outperform a matrix completion based approach in terms of sample and computational requirements in numerical experiments.**

I. Introduction

Modeling and understanding user ratings based on structure is a recent but well-studied discipline. In this setting, we have n products and m users, and our goal is to determine the overall rating-matrix – which is comprised of ratings each user for each product. The main issue though, is that users only provide us with a subset (possibly random) of ratings, and we must now attempt to learn the remainder of the matrix entries. To this end, structure plays a key role, and low-rank structure is particularly useful in helping complete the overall matrix [1], [2], [3].

In many scenarios, however, the ultimate goal is to understand user *ranking*, with ratings merely being a stepping stone along the way. In other words, we are interested in determining the order in which each of the users would like these products. For example, if the n products were movies, ranking reflects each user's preference of movies using an integer ordering, with ties broken randomly. Similarly, whenever we have multiple products/brands of the same type (whether they be toasters, washers or restaurants), a rank-ordering of them proves to be an effective representation of their relative merits. Intuitively, a raw rating of 7 out of 10 in the absence of any other information is potentially useless (Is 7/10 good? or bad? or average?). In standardized tests such as SAT and LSAT, relative performance is captured using a percentile ranking which has now become the gold standard for admissions. Thus, the ranking of a product relative to its peers is valuable information for ultimate user consumption.

As mentioned before, given all ratings, rankings can be obtained by sorting the ratings for each user. However, finding user ratings first and then transforming them to rankings is indirect, and may require much more information and structure than the problem setting allows. This is due to the fact that the range of user ratings can be quite subjective. For example, given a rating scale of 0 to 10, a user can pick her top rating to be 6 and least favorable rating as 4, limiting the actual range of values significantly. As a result, the actual ranking of a product can be considerably different from what the rating indicates when taken out-of-context. In other words, two users with identical rankings of products can have a very different set of ratings. In some settings such as funding-proposal rating, coursework grading etc., we frequently observe a rating/grade *inflation*, where the range of ratings associated with the work being assessed is skewed in favor of a less-punitive scale.

The subjectivity of ratings provided by users also negatively impacts low-rank structure – the basis for the effectiveness of powerful matrix completion techniques in predicting missing ratings.

Motivating Example: Even if all m users in the system have exactly the same ranking for all products, their choices of real-valued ratings can result in a rating matrix that is full rank. Without loss of generality, we can assume the common ranking to be $[1, \dots, n]$. If each user were to generate n real numbers uniformly over $[0, 10]$ and then sort them in descending order, the resulting $m \times n$ matrix will be full rank with high probability. Intuitive justification for low-rank matrix completion techniques originates from the fact that user preferences have only a few degrees of freedom. However, with significant user subjectivity, we expect rankings to capture similarities in user preferences more effectively than ratings.

Consequently, learning the rankings of a collection of users directly is of primary interest. Indeed, as Weimer et al. [4] argue, “Rating algorithms solve the wrong problem, and one that is actually harder: The absolute value of the rating for an item is highly biased for different users, while the ranking is far less prone to this problem.”

Much of existing work on learning rankings of objects deals with learning a single, “globally appropriate” ordering using preferences from training examples, to minimize a suitable notion of loss ([5], [6]). These include the popular *learning-to-rank* approaches [7], [8] and graph-based learning techniques [9], [10], and online permutation learning algorithms and frameworks [11], [12], [13]. Related work on sorting with noise or sorting partially ordered sets can be found in [14], [15], [16].

When a collection of orderings from users is to be learnt, such methods could ideally be applied in a sequential, decoupled fashion to deduce the orderings. However, structure among user orderings, if present, can potentially be exploited to learn the orderings with savings in sample complexity. Researchers have noted that rankings in a population of users often exhibit forms of “low-dimensional” structure – to paraphrase Jagabathula and Shah [17], “Irrespective of the number of candidates in an election, the number of distinct vote rankings that prevail in the population are likely to be few, considering a small set of ‘issues’ influences ranking patterns over candidates.” This inspires the following question when jointly estimating users’ rankings of objects: How can structure among user orderings be effectively leveraged to learn orderings with significantly less effort?

In this work, we study the problem of learning a collection of permutations chosen by m users for n items using only pairwise ordering information. Pairwise sampling asks a user to compare two specified items each time, and is not only a natural choice for attempting to deduce ordering information, but also easy to implement in practical systems. We consider the learning problem under both *random* (i.e., algorithm-independent) and *active* (i.e., algorithm-dependent) pairwise sampling schemes. As a reasonable structural constraint on the space of user permutations, we assume a stochastic model in which the users pick permutations uniformly from a pool of r possible orderings.

For both the random and active sampling schemes, we design efficient, low-complexity algorithms that can reconstruct all the users’ orderings with a guaranteed number of pairwise samples, with high probability. Moreover, we establish, using information-theoretic techniques and concentration results, that the sample-complexity of our algorithms matches lower bounds on the number of pairwise samples needed by any procedure to learn permutations with high probability, when m, n, and r are large. This shows that these reconstruction algorithms are *order-optimal* – in the sense of sample complexity – for learning users’ rankings from pairwise comparisons. The superior performance of our algorithms for the task of learning user orderings is also borne out in practice in the results of numerical experiments that we report.

Organization: The remainder of the paper is organized as follows. We describe the setup for the problem of learning users’ orderings from pairwise comparisons in Section II. In Section III, we present our algorithms to infer users’ orderings, state performance guarantees and converse results for the learning problem, and discuss the implications of our results. Section IV presents numerical results for the performance of our approach compared to that of a matrix completion based technique to solve the same problem.

Notation: We let $[n]$ denote the set of all integers from 1 to n. We denote the symmetric group on $[n]$ by $\mathcal{S}_n$. A permutation $\pi \in \mathcal{S}_n$ is a bijection on $[n]$, and $\pi(i)$ represents the rank of object i. Throughout this paper, we use $N \triangleq \binom{n}{2} = n(n-1)/2$ to denote the number of distinct pairs $(i,j) \in [n] \times [n]$, $i < j$. We can also represent a permutation $\pi \in \mathcal{S}_n$ by a $n \times n$ matrix $\mathbf{P}_\pi$ such that $P_\pi(i,j) = -1$ if $\pi(i) > \pi(j)$ and $P_\pi(i,j) = +1$ otherwise. Since $\mathbf{P}_\pi$ is skew-symmetric, a more practical representation is the stacking of its upper triangular entries into a vector $\mathbf{p}_\pi \in \{-1,1\}^N$. There is a trivial bijection between the two representations, so we use them interchangeably. Throughout, the phrase “with high probability” is used to mean with probability at least $1 - cn^{-1}$ for constant $c > 0$.

II. Learning Users’ Orderings: Setup

Consider the setup where each one of m users totally orders a set of n objects; we denote the resulting permutation of user $k \in [m]$ by $\pi_k \in \mathcal{S}_n$. The goal is to recover all of these permutations with a small number of *pairwise ordering samples*, i.e. how a user relatively orders a specified pair of objects, from each user. Specifically, let $\mathbf{M} = [\mathbf{p}_{\pi_1}\ \mathbf{p}_{\pi_2}\ \ldots\ \mathbf{p}_{\pi_m}]$ be the $N \times m$ matrix of pairwise orderings for all users. The *sampling set* $\Omega \subseteq [N] \times [m]$ denotes the indices of entries of $\mathbf{M}$ we sample, $\mathbf{M}(\Omega)$ denotes the set of all samples acquired, and $s = |\Omega|$ is the number of acquired samples. Sampling can be performed either uniformly at random (*random sampling*) or arbitrarily and adaptively by the algorithm (*active sampling*). In this setup, we are interested in

- Quantifying the minimum *sample complexity* of the learning problem, i.e., the number of samples required to infer all the users' permutations with high probability, and

- Developing *efficient algorithms* that are *optimal for sample-complexity*, i.e., that successfully recover all permutations drawing the minimum number of samples required.

Model for User Permutations: Without further assumptions on the permutations π_k that all the users choose, the problem of learning all the π_k is in general decoupled. This renders unnecessary anything other than a sequential, independent approach to learn each permutation with pairwise samples. The problem of learning a collection of orderings becomes interesting when we impose structure on these orderings, since we can then hope to exploit the resulting "coupling" between user ordering behavior.

In practice, as noted in the introduction, item orderings across a population of users are likely to be much fewer than all the $n!$ permutations in $\mathcal{S}_n$. This can be attributed primarily to a small set of underlying "features" that essentially drive the users' preferences. We consider a natural structural model where each user picks her permutation uniformly at random and independently from a common pool of randomly selected permutations. Specifically, we impose a "low-dimensionality" constraint as follows:

Assumption: There exists a set of r permutations $\{\rho_1, \rho_2, \ldots, \rho_r\}$, where each ρ_j is drawn independently and uniformly at random from $\mathcal{S}_n$. Each π_k is drawn from the ρ_j independently and uniformly at random, i.e., $\mathbb{P}(\pi_k = \rho_j) = 1/r \quad \forall k \in [m],\ j \in [r]$.

We remark that in correspondence with the matrix-completion literature, the assumption above makes the ± 1 matrix of pairwise orderings across all users ($\mathbf{M} = [\mathbf{p}_{\pi_1}\ \mathbf{p}_{\pi_2}\ \ldots\ \mathbf{p}_{\pi_m}]$) *at most rank* r, and thus may be viewed as a surrogate to "low-rank" structure in our permutation-learning setup. The setup is characterized completely by the triple (n, m, r), and our algorithms and results are expressed chiefly in terms of these parameters.

III. Algorithms, Main Results and Implications

In this section, we present algorithms for recovering (and sampling when permitted) all the permutations under both the random and active sampling models. For each case, we provide rigorous analytical guarantees on the number of samples sufficient for our algorithms to exactly recover all permutations with high probability. This is followed by matching converse results, using information-theoretic source-coding techniques, that establish fundamental lower bounds on the sample-complexity required by *any* algorithm to learn the permutations with a significant probability. We discuss the implications of our results and comment on their consequences.

A. Learning with Random Pairwise Samples

Suppose that the set of samples Ω is obtained by uniform sampling with replacement from $[N] \times [m]$, i.e., the set of all (object pair, user) combinations. This models the case where, for instance, every user is asked to independently provide pairwise comparisons for a uniformly randomly chosen set of object pairs. The problem is then to use these results to deduce the users' orderings of all the objects. We introduce our first algorithm (Algorithm 1) to learn the permutations given s randomly drawn samples, and show that it recovers all the orderings with high probability given a sufficient number of random samples. In this description, we denote the sampling set by $\Omega_s \subset [N]\times[m]$ to indicate its size s and use $\Omega_{s,u} \subset [N]$ to denote the positions (object pairs) sampled from user $u \in [m]$.

In essence, Algorithm 1 uses the s pairwise samples to first separate each pair of users if there is any *discrepancy* in their sampled comparisons. A discrepancy occurs between two users u and k if their sampled orderings for a pair of objects (i,j) disagree, i.e., $(i,j) \in \Omega_{s,u} \cap \Omega_{s,k}$ and $\mathbf{M}(\Omega_{s,u}, u)_{(i,j)} \neq \mathbf{M}(\Omega_{s,k}, k)_{(i,j)}$. Having "clustered" the users' permutations thus, the algorithm proceeds to completely learn the (presumably correctly clustered) permutations by collecting all pairwise samples from users belonging to each cluster and topologically sorting the resulting Directed Acyclic Graph (DAG).

Our first result concerns the sample-complexity of Algorithm 1:

Theorem 1 (Random Sampling, Algorithm 1)**.** *Suppose $r = \theta(m^\gamma)$ for a fixed $\gamma > 0$. Algorithm 1 recovers all permutations correctly, with high probability, when the number of random samples s is at least* $\max\{(12/\gamma) m \log r, 2rN \log n\}$.

Proof sketch: The two terms given in the bound of Theorem 1 above quantify separately the sample-complexities needed to successfully complete both steps of the algorithm, i.e. *clustering* and *learning*. We first establish a concentration result for the pairwise Hamming distance between two distinct permutations drawn uniformly at random. This allows us to show that $O(\log r)$ random pairwise comparisons per permutation are sufficient to distinguish them. Alongside, for any fixed permutation, we identify a necessary and sufficient condition to exactly learn the permutation, from pairwise samples, in terms of the unique Hamiltonian path of the digraph induced by the permutation. This is used together with a coupon-collecting argument to show that the permutation-learning stage of Algorithm 1 requires $O(N \log n)$ random samples per cluster (i.e. for each

Algorithm 1

```
Input: Set of sampled positions $\Omega_s \subset [N] \times [m]$ and samples $\mathbf{M}(\Omega_s) \in \{-1,+1\}^s$.
Output: Permutations of all users $(\hat{\pi}_k)_{k=1}^m \in \mathcal{S}_n$.
Stage 1: Clustering:
  1) Set $\mathcal{C}$ to be an empty collection of clusters.
  2) Set $\mathcal{U} \leftarrow [m]$, to be the set of all unclustered users.
  3) If $\mathcal{U} = \emptyset$, go to Stage 2.
  4) Let $u \leftarrow \min_{k \in \mathcal{U}} k$, set $\mathcal{U} \leftarrow \mathcal{U} \setminus u$ and $\mathcal{L} \leftarrow \{u\}$.
  5) For every $k \in \mathcal{U}$
       • If $\mathbf{M}(\Omega_{s,u} \cap \Omega_{s,k}, u) = \mathbf{M}(\Omega_{s,u} \cap \Omega_{s,k}, k)$
         then set $\mathcal{U} \leftarrow \mathcal{U} \setminus k$ and $\mathcal{L} \leftarrow \mathcal{L} \cup \{k\}$.
  6) Set $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathcal{L}\}$ and go to Step 3.
Stage 2: Permutation Learning:
For every cluster $\mathcal{L} \in \mathcal{C}$,
  1) Let $\Omega_{\mathcal{L}} \leftarrow \bigcup_{k \in \mathcal{L}} \Omega_{s,k}$
  2) Let $G = (V, E)$ denote a directed graph, with vertex set $V = [n]$ and edge set $E = \emptyset$.
  3) For every sample position $p$ in $\Omega_{\mathcal{L}}$, drawn from user $k$ and corresponding to object pair $(i, j)$
       • if $\mathbf{M}(p, k) = -1$ then $E \leftarrow E \cup \{(i, j)\}$; else $E \leftarrow E \cup \{(j, i)\}$
  4) Set $\hat{\rho}_{\mathcal{L}} \leftarrow$ TopologicalSorting(G)
  5) Set $\hat{\pi}_k \leftarrow \hat{\rho}_{\mathcal{L}}$ for all $k \in \mathcal{L}$.
```

ρ_j) to completely infer the cluster. Putting together these estimates gives the theorem. The details of the proof are provided in the full version ([18]) of the present paper.

On the other hand, we establish a converse result on the minimum number of samples needed for successful permutation recovery. For this purpose, consider a general algorithm $\mathcal{A}$ that takes as input s pairwise random samples $\mathbf{M}(\Omega_s)$ and maps it to its output: a possibly random estimate $\hat{\mathbf{M}} \triangleq [p_{\hat{\pi}_1}\ p_{\hat{\pi}_2}\ \ldots p_{\hat{\pi}_m}]$ of all the permutations, one for each user. We denote the probability of successful reconstruction with $\mathcal{A}$ on s samples by $P_{\text{succ}} = P_{\text{succ}}(\mathcal{A}, s) = \mathbb{P}[\hat{\mathbf{M}} = \mathbf{M}]$.

Theorem 2 (Random Sampling, Lower Bound on Sample Complexity)**.** *For any algorithm $\mathcal{A}$, if $s < \max\{(m-r)\log r, rN\log n\}$, then $P_{\text{succ}} \to 0$ as $n \to \infty$.*

Proof sketch: For the sake of contradiction, assume that an algorithm can learn the orderings with fewer than the claimed number of samples. It follows that this results in an algorithm that performs the easier clustering and permutation learning (given clustering information) tasks with those samples. Hence, it suffices to prove separate converse results for these two stages. A key contribution of this work is to prove a strong converse theorem for clustering. This is carried out by first extending the information-theoretic notion of typicality to "clusterings", and using a source-coding proof technique that results in a converse theorem. For the permutation learning stage, an important step is to show that the number of random pairwise samples needed to learn a single permutation with high probability is $\Omega(N\log n)$. For this purpose, the necessary and sufficient Hamiltonian path condition – from the proof of Theorem 1 – that characterizes when a permutation is learnt can be used along with concentration estimates for the lower tail of the coupon-collector problem to prove the result. We defer details of the full proof to the full version ([18]) of this paper.

Note that Theorem 2 is a *strong converse*, i.e., it states that *any* algorithm fails to recover all the users' item orderings correctly with *overwhelming probability* when the number of random samples drawn is below a threshold.

Implications of Theorems 1 and 2:

- When the number of users m is relatively small, viz. $m = O(rN\log n) = O(rn^2\log n)$, Algorithm 1 succeeds overwhelmingly with $O(rN\log n)$ samples according to Theorem 1. At the same time, with our standing assumption that $r = \theta(m^\gamma)$ for a fixed γ, the converse Theorem 2 forces at least $\Omega(rN\log n)$ samples to be drawn for correct reconstruction. Thus, in this regime, Algorithm 1 is *order-optimal* for the sample-complexity of the problem. Further, it demands on an average $\frac{r}{m}N\log n$ random samples from each user. If $\gamma < 1$ additionally, this means that each user needs to contribute a *vanishing number* of pairwise comparisons for successful recovery. This represents a significant gain compared to decoupling the learning problem across users and reconstructing each permutation independently (whose net sample complexity is $mN\log n$).
- In general, the best reconstruction algorithm for any number of samples is information-theoretically specified to be the *Maximum-Likelihood (ML)* reconstruction algorithm, i.e., the algorithm that outputs a set of user permutations that maximizes the *a posteriori* probability of permutations given sampled observations. Solving the maximum likelihood problem requires performing a potentially hard combinatorial optimization over the space of all possible user ordering patterns – a computationally infeasible task. However, in the above regime with a relatively small number of users, it is remarkable that the efficient and simple Algorithm 1 achieves the same sample complexity as the ML algorithm for the permutation-learning problem.

B. Learning with Active Pairwise Samples

In many application scenarios, it is often desirable (and possible) to *actively* query users for comparisons of objects. Thus, the choice of samples could be more intelligent, and we can hope to accomplish the learning task with a *smaller number* of carefully chosen pairwise samples than if we took uncontrolled random pairwise samples. Here, we indeed show that this is the case, and provide a joint sampling and permutation-learning algorithm (Algorithm 2) that is both (a) order-optimal across all learning algorithms, and (b) requires fewer samples than its random-sampling counterpart.

Algorithm 2

```
Input: Pairwise representation matrix $\mathbf{M}$; Number of samples $s$ the algorithm is allowed to use.
Output: Permutations of all users $(\hat{\pi}_k)_{k=1}^m \in \mathcal{S}_n$.
Stage 1: Clustering
  1) Set $\Omega_{\mathcal{C}}$ to be a random subset of [N], of size $\min\{c \log r, s\}$
  2) Set $\mathcal{C}$ to be an empty collection of clusters.
  3) Set $\mathcal{U} \leftarrow [m]$, to be the set of all unclustered users.
  4) If $\mathcal{U} = \emptyset$, go to Stage 2.
  5) Let $u \leftarrow \min_{k \in \mathcal{U}} k$, set $\mathcal{U} \leftarrow \mathcal{U} \setminus u$ and $\mathcal{L} \leftarrow \{u\}$.
  6) For every $k \in \mathcal{U}$
       • If $\mathbf{M}(\Omega_{\mathcal{C}}, u) = \mathbf{M}(\Omega_{\mathcal{C}}, k)$ then set $\mathcal{U} \leftarrow \mathcal{U} \setminus k$
         and $\mathcal{L} \leftarrow \mathcal{L} \cup \{k\}$.
  7) Set $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathcal{L}\}$ and go to Step 3.
Stage 2: Permutation Learning through Sorting
For every cluster $\mathcal{L} \in \mathcal{C}$,
  1) Use a sorting algorithm to sort the $[n]$ objects in cluster $\mathcal{L}$ balancing the sample load across all users in the cluster.
  2) If sample budget $s$ is reached before completion, stop and declare failure.
```

Algorithm 2 follows the same basic outline of operation as Algorithm 1, i.e., working by clustering and learning each clustered permutation. The key departure here is that it is free to specify pairs of items that it wants compared by certain users, and so it uses this flexibility to cluster and learn permutations faster. Specifically, it first picks a random *common* set of $c \log r$ pairs of objects that it asks *all* r users to order, and uses these samples to cluster users' putative permutations. Once clustering is accomplished, the algorithm pretends that each cluster is a single ordering and attempts to learn the ordering using a standard sorting algorithm on n items (we use Quicksort in our implementation, Section IV) that issues pairwise queries to essentially "complete" the permutation. Following the same outline as with Algorithm 1, we first bound the sample complexity of Algorithm 2 as follows.

Theorem 3 (Active Sampling, Algorithm 2)**.** *Algorithm 2 correctly recovers all permutations, with high probability, taking $s = O(m \log r + rn \log n)$ pairwise samples.*

Proof sketch: The arguments used to prove Theorem 3 follow the same outline as those for Theorem 1, viz. estimating the number of samples sufficient to perform the two steps of clustering users and learning the clusters. Concentration properties for the Hamming distance between randomly chosen permutations are employed for the estimate of $O(\log r)$ *random common object pairs* which guarantees successful clustering for all users. This is followed by observing that actively sampling and learning a single cluster/permutation is equivalent to sorting items using pairwise comparisons, and standard tail bounds for the performance of a standard sorting algorithm such as Quicksort show that $O(n \log n)$ pairwise samples suffice to learn each cluster with high probability. Putting together these estimates gives the promised bound. The full proof details are provided in the full version ([18]) of this paper.

For the case of active sampling, we provide a matching converse theorem – in the same spirit as Theorem 2 – for the sample complexity of *any* algorithm that is *free* to draw any pairwise samples from the users. Recall from the random sampling scenario, that we denote the probability of successful reconstruction, using algorithm $\mathcal{A}$ on s samples, by $P_{\text{succ}} = P_{\text{succ}}(\mathcal{A}, s)$.

Theorem 4 (Active Sampling, Lower Bound on Sample Complexity)**.** *For any active-sampling algorithm $\mathcal{A}$, if $s < \max\{(m - r) \log r, rn \log n\}$, then $P_{succ} \to 0$ as $n \to \infty$.*

Proof sketch: The proof for this active sampling converse theorem uses the same high-level outline as that for Theorem 2. The first part – for the clustering stage – is the same information-theoretic source coding argument as in the proof of Theorem 2. For the second part, we use a converse argument for jointly learning a set of r distinct permutations, which essentially generalizes the $\Omega(n \log n)$ pairwise comparison result to sort a set of n item. We reproduce the full details of the proof in the full version ([18]) of this paper.

Implications of Theorems 3 and 4:

- Algorithm 2 achieves *perfect reconstruction* with an *order optimal* number of samples (i.e. $O(m \log r + rn \log n)$). In other words, distinguishing users on the basis of a few ($O(mlogr)$) common pairwise comparisons decouples the overall learning problem *tightly* into r independent "cluster-learning" or sorting problems.
- Compared to the sample complexity of learning with random samples (Theorems 1, 2), Algorithm 2 exhibits a saving in sample complexity of the order

of n. This can be directly attributed to the gain in "collaboratively sorting" users clustered together as the "same", in the second phase of the algorithm. Also, the sample complexity of Algorithm 2 translates into an average of $\frac{rn}{m} \log n$ samples per user – a gain of the order of r/mn over trying to reconstruct all permutations independent of each other.

Remark. *It is worth noticing that, even if the order samples are q-ary (i.e. full orderings of subsets of size q) instead of pairwise samples, for constant q, the order-wise behaviour of the sample complexities does not change.*

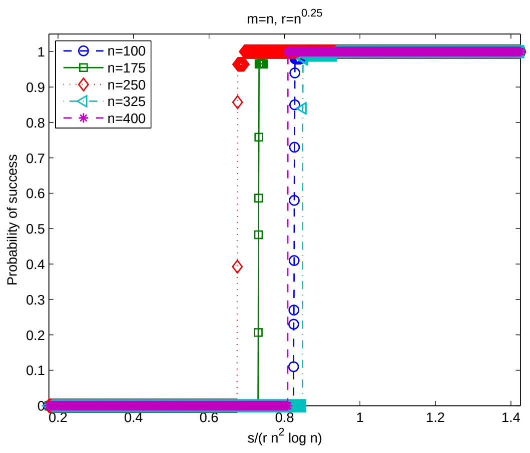
IV. Experimental Results

This section describes results of numerical experiments on synthetic data following the stochastic model for user permutations introduced in this work. For the random sampling case, Matrix Completion (MC) is an attractive choice of algorithm to hope to recover users' permutations, since (a) we essentially get to see partial entries from the ± 1 matrix $\mathbf{M}$ of pairwise representations of the permutations (note, though, that the $O(n^2)$ pairwise representation is over-complete), and (b) if the users have at most r distinct permutations among themselves, the rank of $\mathbf{M}$ is at most r. Hence, for the random sampling problem, we compare the performance of our algorithms – both in terms of sample complexity and running time – with an Augmented Lagrangian Method version of Matrix Completion (ALM-MC, relevant code from [19]). Finally, we also present an overall comparison of sample complexities across both random and active sampling cases and algorithms. This helps to put both sampling methods in perspective, and also illustrates the order-wise gains when the learning algorithm is allowed to sample pairwise orderings from users at will. All the routines run in MATLAB on a 2.4 GHz desktop computer system with 4 GB of memory.

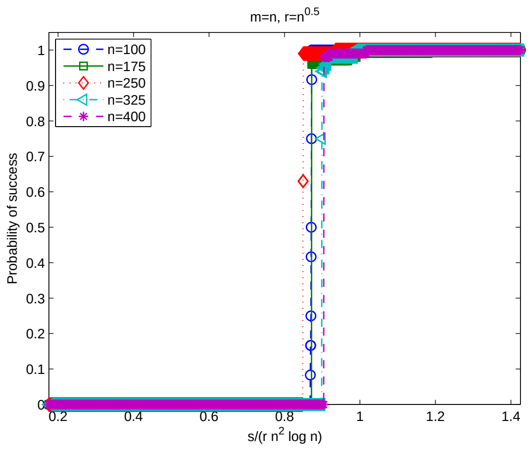
A. Random Sampling: Algorithm 1 and ALM-MC

For our random sampling experiments we set $m = n$, varying from 100 to 400, and consider two scaling regimes of r: $r = m^{0.25}$ and $r = m^{0.5}$. We generate random permutations for all m users by first picking r random permutations and then assigning users randomly to these permutations.

Figure 1 shows how the reconstruction success probability of Algorithm 1 scales with the normalized number $s/(rn^2 \log n)$ of random pairwise samples drawn from users. Each of the colored curves represents a fixed value of n, the number of items, and depicts the success probability of Algorithm 1 as the number of random samples s is varied. We observe a sharp "phase transition" effect for the probability of success at this normalized scale of samples – the "correct" normalization suggested by Theorems 1 and 2.



(a)

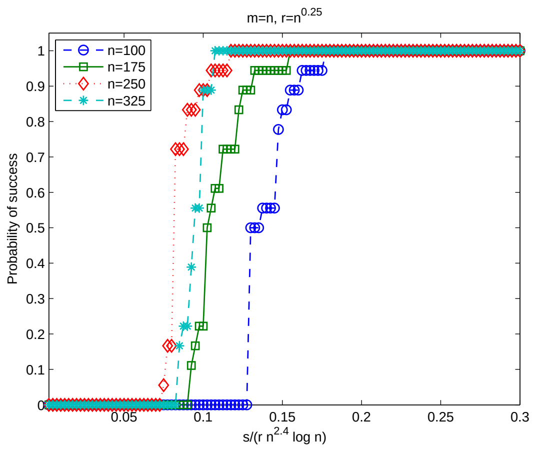


(b)

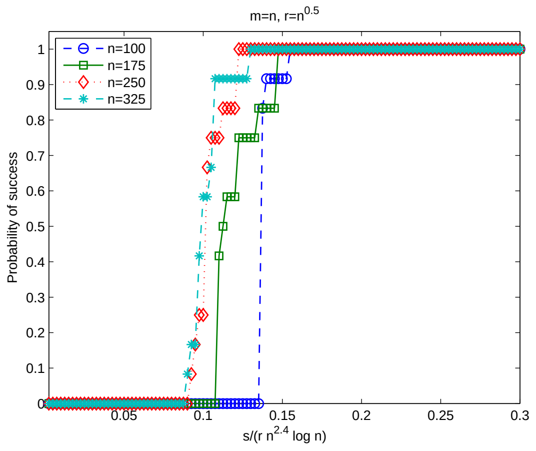
Fig. 1. Probability of success vs. number of random samples s normalized by $rn^2 \log n$ for Algorithm 1, (a) $r = m^{0.25}, m = n$,(b) $r = m^{0.5}, m = n$.

We plot the corresponding probabilities of success for ALM-MC (matrix completion on the pairwise ± 1 matrix) in Figure 2. An important point here is that due to our stochastic model for user permutations, the matrix of pairwise ordering representations of users is at most rank r, and the number of samples required to complete this matrix (and recover all orderings) is $O(rN^{1.2} \log N) = O(rn^{2.4} \log n)$ when incoherence is constant, according to Candès and Recht [2]. Thus, we use this normalizing factor for the number of samples in our plots.

Note the similar phase transition for the success probability for ALM-MC as the number of drawn samples varies. Also, since the phase transition occurs at the $n^{2.4}$ timescale instead of the n^2 timescale for Algorithm 1 (Figure 1), *ALM-MC requires order-wise $n^{0.4}$ more* *samples than Algorithm 1 to succeed*. Not only is this concordant with the lower bound on sample complexity given by Theorem 2, but it also demonstrates the order-wise superior performance of Algorithm 1 to solve the permutation-learning problem from random samples.



(a)

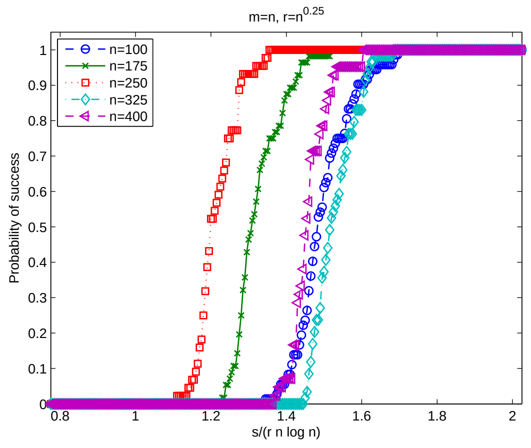


(b)

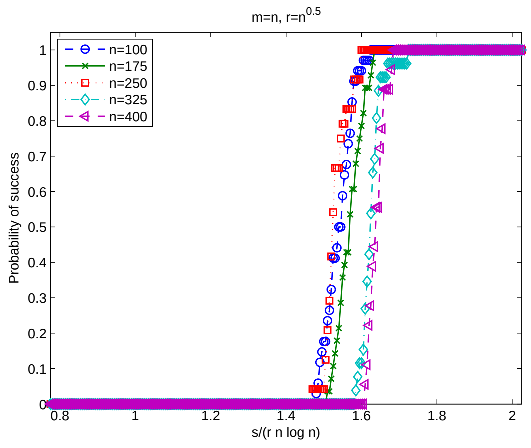
Fig. 2. Probability of success vs. number of random samples s normalized by $rn^{2.4}\log n$ for Augmented Lagrange Multiplier-based Matrix Completion (ALM-MC), (a) $r = m^{0.25}, m = n$,(b) $r = m^{0.5}, m = n$.

B. Active Sampling: Algorithm 2

We plot the success probability of Algorithm 2 which draws active samples (Figure 3), for the same regime (m, n, r) as in the random sampling case. Here, as indicated by Theorems 3 and 4, the right scale of normalization for the number of samples taken is $rn \log n$ – an n-fold improvement over reconstruction with random sampling. The phase transition for Algorithm 2's success probability is clearly visible in Figure 3.



(a)



(b)

Fig. 3. Probability of success vs. number of samples s normalized by $rn \log n$ for Algorithm 2 (active sampling), (a) $r = m^{0.25}, m = n$,(b) $r = m^{0.5}, m = n$.

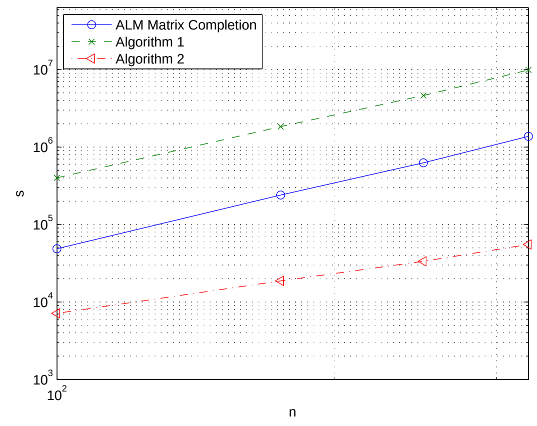
C. Overall Comparison: Sample Complexities and Running Time

To put all the algorithms considered so far in perspective, we compare and contrast their sample complexities and running times together.

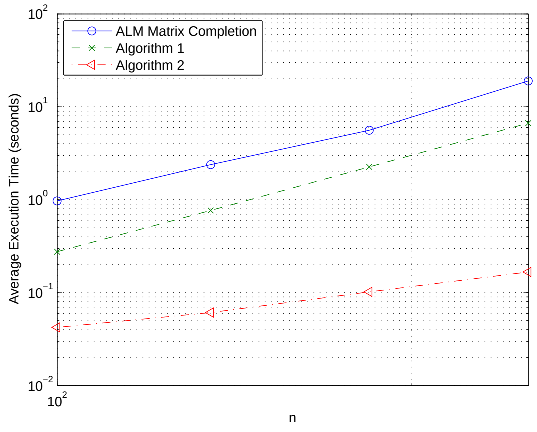
Figure 4(a) compares the number of samples at the success probability phase transition (s) against the problem size (n) for Algorithms 1 and 2 and ALM-MC. Algorithm 2 dramatically outperforms its random sampling counterparts, lending support to the active sampling model of attempting to learn user rankings. On the other hand, in the random sampling case, though ALM-MC (matrix completion) fares better than the order-optimal Algorithm 1, we suspect that this is due to overheads occurring at low problem sizes and observe that the curves for ALM-MC and Algorithm 1 are projected to cross over at larger sizes of n.

In Figure 4(b) shows the running times on a 2.4 GHz

CPU with 4 GB of memory, for our implementations of Algorithms 1 and 2, and the off-the-shelf implementation of ALM-MC. Here, *the standard ALM matrix completion algorithm is outperformed by both Algorithms 1 and 2, illustrating the gain in computational efficiency that these algorithms offer*.



(a) Experimental sample-complexities vs. n for Algorithms 1, 2 and ALM-MC, $m = n$ and $r = n^{0.5}$.



(b) Experimental execution time in seconds vs. n for Algorithms 1, 2 and ALM-MC, $m = n$ and $r = n^{0.5}$.

Fig. 4. Experimental results on sample and time complexity of all algorithms.

V. Conclusion

We considered the problem of learning a collection of users' permutations of items using just partial pairwise comparisons. Both random and active/intelligent sampling schemes were separately considered. In both cases, we developed efficient algorithms that reconstruct the permutations with a guaranteed sample complexity, and using corresponding lower bounds on sample complexity showed that these algorithms are order-optimal, additionally with an order-wise performance improvement when sampling actively. Moreover, there is a significant gain when solving the problem jointly compared to learning each permutation individually. Experiments were carried out that validated the performance benefits of the algorithms we presented, and in many cases showed their superiority over traditional matrix-completion approaches.

References

[1] R. H. Keshavan, A. Montanari, and S. Oh, "Matrix completion from a few entries," *IEEE Transactions on Information Theory*, vol. 56, pp. 2980–2998, 2010.

[2] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Found. Comput. Math.*, vol. 9, pp. 717–772, December 2009.

[3] B. Recht, "A simpler approach to matrix completion," *CoRR*, vol. abs/0910.0651, 2009.

[4] M. Weimer, A. Karatzoglou, Q. V. Le, and A. Smola, "CoFiRank, Maximum Margin Matrix Factorization for Collaborative Ranking," in *Advances in Neural Information Processing Systems*, vol. 20, 2007.

[5] B. Zelinka, "Distances between partially ordered sets," *Mathematica Bohemica*, vol. 118, no. 2, pp. 167–170, 1993.

[6] A. Haviar and B. Bystrica, "A metric on a system of ordered sets," *Mathematica Bohemica*, vol. 121, no. 2, pp. 123–131, 1996.

[7] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *J. Mach. Learn. Res.*, vol. 4, pp. 933–969, December 2003.

[8] W. W. Cohen, R. E. Schapire, and Y. Singer, "Learning to order things," *J. Artif. Intell. Res. (JAIR)*, vol. 10, pp. 243–270, 1999.

[9] S. Agarwal, "Learning to rank on graphs," *Machine Learning*, vol. 81, no. 3, pp. 333–357, 2010.

[10] S. Rajaram and S. Agarwal, "Generalization bounds for k-partite ranking," in *Proceedings of the NIPS-2005 Workshop on Learning to Rank*, 2005.

[11] D. Helmbold and M. Warmuth, "Learning permutations with exponential weights," in *Proceedings of the 20th annual conference on Learning theory*. Springer-Verlag, 2007, pp. 469–483.

[12] G. Fung, R. Rosales, and B. Krishnapuram, "Learning rankings via convex hull separation," *Advances in Neural Information Processing Systems*, vol. 18, p. 395, 2006.

[13] G. Lebanon and J. Lafferty, "Cranking: Combining rankings using conditional probability models on permutations," in *Proceedings of the Nineteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 2002, pp. 363–370.

[14] M. Ajtai, V. Feldman, A. Hassidim, and J. Nelson, "Sorting and selection with imprecise comparisons," *Automata, Languages and Programming*, pp. 37–48, 2009.

[15] U. Feige, P. Raghavan, D. Peleg, and E. Upfal, "Computing with noisy information," *SIAM J. Comput.*, vol. 23, no. 5, pp. 1001–1018, 1994.

[16] C. Daskalakis, R. Karp, E. Mossel, S. Riesenfeld, and E. Verbin, "Sorting and selection in posets," in *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2009, pp. 392–401.

[17] S. Jagabathula and D. Shah, "Inferring rankings under constrained sensing," in *Advances in Neural Information Processing Systems*, 2008, pp. 753–760.

[18] I. Mitliagkas, A. Gopalan, C. Caramanis, and S. Vishwanath, "User rankings from comparisons: Learning permutations in high dimensions," long version available at `webspace.utexas.edu/im4454/www/`.

[19] Z. Lin, M. Chen, L. Wu, and Y. Ma, "The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices," *Arxiv preprint arXiv:1009.5055*, 2010.