Minimizing Time-to-Rank: A Learning and Recommendation Approach

Haoming Li[1] , **Sujoy Sikdar**[2] , **Rohit Vaish**[2*] , **Junming Wang**[2] , **Lirong Xia**[2] and **Chaonan Ye**[3]
[1]Duke University
[2]Rensselaer Polytechnic Institute
[3]Stanford University
haoming.li@duke.edu, {sikdas, vaishr2, wangj33}@rpi.edu, xial@cs.rpi.edu, canonyeee@gmail.com

Abstract

Consider the following problem faced by an online voting platform: A user is provided with a list of alternatives, and is asked to rank them in order of preference using only drag-and-drop operations. The platform's goal is to recommend an initial ranking that minimizes the time spent by the user in arriving at her desired ranking. We develop the first optimization framework to address this problem, and make theoretical as well as practical contributions. On the practical side, our experiments on Amazon Mechanical Turk provide two interesting insights about user behavior: First, that users' ranking strategies closely resemble *selection* or *insertion* sort, and second, that the time taken for a drag-and-drop operation depends *linearly* on the number of positions moved. These insights directly motivate our theoretical model of the optimization problem. We show that computing an optimal recommendation is NP-hard, and provide exact and approximation algorithms for a variety of special cases of the problem. Experimental evaluation on MTurk shows that, compared to a random recommendation strategy, the proposed approach reduces the average time-to-rank by up to 50%.

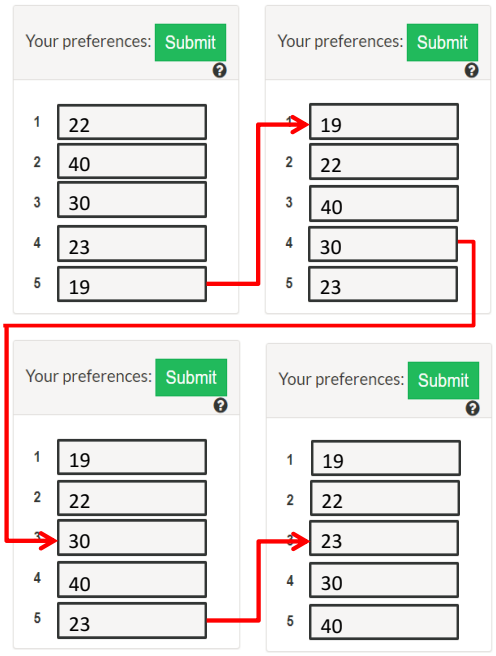

Figure 1: Sorting via drag-and-drop operations.

1 Introduction

Eliciting preferences in the form of rankings over a set of alternatives is a common task in social choice, crowdsourcing, and in daily life. For example, the organizer of a meeting might ask the participants to rank a set of time-slots based on their individual schedules. Likewise, in an election, voters might be required to rank a set of candidates in order of preference.

Over the years, computerized systems have been increasingly used in carrying out preference elicitation tasks such as the ones mentioned above. Indeed, recently there has been a proliferation of online voting platforms such as CIVS, OPRA, Pnyx, RoboVote, and Whale[4].[1] In many of these platforms, a user is presented with an arbitrarily ordered list of alternatives, and is asked to shuffle them around in-place using *drag-and-drop* operations until her desired preference ordering is achieved. Figure 1 illustrates the use of drag-and-drop operations to sort a list of numbers in descending order.

Our focus in this work is on *time-to-rank*, i.e., the time it takes for a user to arrive at her desired ranking, starting from a ranking suggested by the platform and using only drag-and-drop operations. We study this problem from the perspective of the voting platform that wants to *recommend* an optimal initial ranking to the user (i.e., one that *minimizes* time-to-rank). Time to accomplish a designated task is widely considered as a key consideration in the usability of automated systems [Bevan *et al.*, 2015; Albert and Tullis, 2013], and serves as a proxy for user effort. Indeed, 'time on task' was identified as a key factor in the usability and efficiency of computerized voting systems in a 2004 report by NIST to the U.S. Congress for the Help America Vote Act (HAVA) [Laskowski *et al.*, 2004].

*Contact Author

[1]CIVS (https://civs.cs.cornell.edu/), OPRA (opra.io), Pnyx (https://pnyx.dss.in.tum.de/), RoboVote (http://robovote.org/), Whale[4](https://whale.imag.fr/).

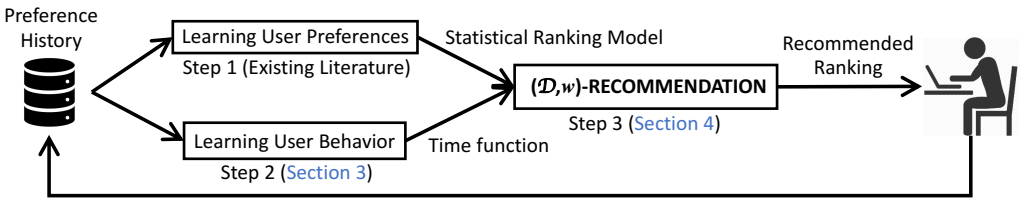

Figure 2: High level overview of our framework. Our technical contributions are highlighted in blue.

Note that the initial ranking suggested by the platform can have a significant impact on the time spent by the user on the ranking task. Indeed, if the user's preferences are known beforehand, then the platform can simply recommended it to her and she will only need to verify that the ordering is correct. In practice, however, users' preferences are often *unknown*. Furthermore, users employ a wide variety of *ranking strategies*, and based on their proficiency with the interface, users can have very different *drag-and-drop times*. All these factors make the task of predicting the time-to-rank and finding an optimal recommendation challenging and non-trivial.

We emphasize the subtle difference between our problem and that of *preference elicitation*. The latter involves repeatedly asking questions to the users (e.g., pairwise comparisons between alternatives) to gather enough information about their preferences. By contrast, our problem involves a one-shot recommendation followed by a series of drag-and-drop operations until the desired ranking is achieved. There is an extensive literature on preference elicitation [Conitzer and Sandholm, 2002; Blum *et al.*, 2004; Boutilier, 2013; Busa-Fekete *et al.*, 2014; Soufiani *et al.*, 2013; Zhao *et al.*, 2018]. Yet, somewhat surprisingly, the problem of recommending a ranking that minimizes users' time has received little attention. Our work aims to address this gap.

Our Contributions

We make contributions on three fronts:

- On the *conceptual* side, we propose the problem of minimizing time-to-rank and outline a framework for addressing it (Figure 2).
- On the *theoretical* side, we formulate the optimization problem of finding a recommendation to minimize time-to-rank (Section 4). We show that this problem is NP-hard, even under highly restricted settings (Theorem 3). We complement the intractability results by providing a number of exact (Theorem 2) and approximation algorithms (Theorems 4 to 6) for special cases of the problem.
- We use *experimental analysis* to motivate our modeling assumptions as well as to justify the effectiveness of our approach (Section 5). Our experiments on MTurk reveal two insights about user behavior (Section 5.1): (1) The ranking strategies of real-world users closely resemble *insertion/selection sort*, and (2) the drag-and-drop time of an alternative varies *linearly* with the distance moved. We find that a simple adaptive strategy (based on Borda count voting rule) can reduce time-to-rank by *up to* 50% compared to a random strategy (Section 5.2), validating the usefulness of the proposed framework.

1.1 Overview of Our Framework

Figure 2 illustrates the proposed framework which consists of three key steps. In Step 1, we *learn user preferences* from historical data by developing a statistical ranking model, typically in the form of a distribution $\mathcal{D}$ over the space of all rankings (refer to Section 2 for examples of ranking models). In Step 2, which runs in parallel to Step 1, we *learn user behavior*; in particular, we identify their *sorting strategies* (Section 3.1) as well as their *drag-and-drop times* (Section 3.2). Together, these two components define the `time` function which models the time taken by a user in transforming a given initial ranking σ into a target ranking τ, denoted by $\texttt{time}(\sigma, \tau)$. The ranking model $\mathcal{D}$ from Step 1 and the `time` function from Step 2 together define the recommendation problem in Step 3, called $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION (the parameter $\mathbf{w}$ is closely related to the `time` function; we elaborate on this below). This is the optimization problem of computing a ranking σ that minimizes the expected time-to-rank of the user, i.e., minimizes $\mathbb{E}_{\tau\sim\mathcal{D}}[\texttt{time}(\sigma, \tau)]$. The user is then recommended σ, and her preference history is updated.

The literature on learning statistical ranking models is already well-developed [Guiver and Snelson, 2009; Awasthi *et al.*, 2014; Lu and Boutilier, 2014; Zhao *et al.*, 2016; Xia, 2019]. Thus, while this is a key ingredient of our framework (Step 1), our work only focuses on Steps 2 and 3 concerning user behavior and the recommendation problem.

Recall that the `time` function defines the time taken by a user in transforming a given ranking σ into a target ranking τ. For a user who follows a fixed sorting algorithm (e.g., insertion or selection sort), the `time` function can be broken down into (1) the number of drag-and-drop operations suggested by the sorting algorithm, and, (2) the (average) time taken for each drag-and-drop operation by the user. As we will show in Lemma 1 in Section 3.1, point (1) above is *independent* of the choice of the sorting algorithm. Therefore, the `time` function can be equivalently defined in terms of the *weight function* $\mathbf{w}$, which describes the time taken by a user, denoted by $\mathbf{w}(\ell)$, in moving an alternative by ℓ positions via a drag-and-drop operation. For this reason, we use $\mathbf{w}$ in the formulation of $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION.

Applicability. Our framework is best suited for users who have already formed their preferences, so that the recommended ranking does not bias their preferences. This is a natural assumption in some applications, such as in the meeting organization example in Section 1. In general, however, it is possible that a user, who is undecided between options A and B, might prefer A over B if presented in that order by the

Distribution $\mathcal{D}$	Linear Weights			General Weights
	Hardness	Exact Algo.	Approx. Algo.	Approx. Algo.
k-mixture Plackett-Luce (k-PL)	NP-c even for $k = 4$ (Theorem 3)	Poly for $k = 1$ (Theorem 2)	PTAS (Theorem 4) 5-approx. (Theorem 5)	$\alpha\beta$-approx. (Theorem 6)
k-mixture Mallows (k-MM)	NP-c even for $k = 4$ (Theorem 3)	Poly for $k = 1$ (Theorem 2)	PTAS (Theorem 4) 5-approx. (Theorem 5)	$\alpha\beta$-approx. (Theorem 6)
Uniform (Unif)	NP-c even for $n = 4$ (Theorem 3)	Poly for $n \in \{1, 2\}$ (Theorem 2)	PTAS (Theorem 4) 5-approx. (Theorem 5)	$\alpha\beta$-approx. (Theorem 6)

Table 1: Computational complexity results for $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION. Each row corresponds to a preference model and each column corresponds to a weight function. We use the shorthands Poly, NP-c, PTAS, and $\alpha\beta$-approx. to denote polynomial-time (exact) algorithm, NP-complete, polynomial-time approximation scheme, and $\alpha\beta$-approximation algorithm (for (α, β)-close weights; see Definition 6) respectively.

recommended ranking. Investigating such biases (a.k.a. "framing effect") is an interesting direction for future work.

Additional related work. Our work is related to the literature on inferring a ground truth ordering from noisy information [Braverman and Mossel, 2008], and aggregating preferences by minimizing some notion of distance to the observed rankings such as the total Kendall's Tau distance [Procaccia and Shah, 2016]. Previous work on preference learning and learning to rank can also be integrated in our framework [Liu, 2011; Lu and Boutilier, 2014; Khetan and Oh, 2016; Agarwal, 2016; Negahban *et al.*, 2017; Zhao and Xia, 2018].

2 Preliminaries

Let $A = \{a_1, \ldots, a_m\}$ denote a set of m *alternatives*, and let $\mathcal{L}(A)$ be the set of all linear orders over A. For any $\sigma \in \mathcal{L}(A)$, $a_i \succ_\sigma a_j$ denotes that a_i is preferred over a_j under σ, and let $\sigma(k)$ denote the k^{th} most preferred alternative in σ. A set of n linear orders $\{\sigma^{(1)}, \ldots, \sigma^{(n)}\}$ is called a *preference profile*.

Definition 1 (Kendall's Tau distance; [Kendall, 1938])**.** *Given two linear orders $\sigma, \sigma' \in \mathcal{L}(A)$, the Kendall's Tau distance $\mathrm{d_{kt}}(\sigma, \sigma')$ is the number of pairwise disagreements between σ and σ'. That is, $\mathrm{d_{kt}}(\sigma, \sigma') \coloneqq \sum_{a_i, a_j \in A} \mathbf{1}[a_j \succ_{\sigma'} a_i \text{ and } a_i \succ_\sigma a_j]$, where $\mathbf{1}$ is the indicator function.*

Definition 2 (Plackett-Luce model; [Plackett, 1975; Luce, 1959])**.** *Let $\theta \coloneqq (\theta_1, \ldots, \theta_m)$ be such that $\theta_i \in (0, 1)$ for each $i \in [m]$ and $\sum_{i \in [m]} \theta_i = 1$. Let Θ denote the corresponding parameter space. The Plackett-Luce* (PL) *model parameterized by $\theta \in \Theta$ defines a distribution over the set of linear orders $\mathcal{L}(A)$ as follows: The probability of generating $\sigma \coloneqq (a_{i_1} \succ a_{i_2} \succ \ldots \succ a_{i_m})$ is given by*

$$\Pr(\sigma|\theta) = \frac{\theta_{i_1}}{\sum_{\ell=1}^{m} \theta_{i_\ell}} \cdot \frac{\theta_{i_2}}{\sum_{\ell=2}^{m} \theta_{i_\ell}} \cdot \cdots \cdot \frac{\theta_{i_{m-1}}}{\theta_{i_{m-1}} + \theta_{i_m}}.$$

More generally, a k-mixture Plackett-Luce model (k-PL) *is parameterized by $\{\gamma^{(\ell)}, \theta^{(\ell)}\}_{\ell=1}^{k}$, where $\sum_{\ell=1}^{k} \gamma^{(\ell)} = 1$, $\gamma^{(\ell)} \geq 0$ for all $\ell \in [k]$, and $\theta^{(\ell)} \in \Theta$ for all $\ell \in [k]$. The probability of generating $\sigma \in \mathcal{L}(A)$ is given by $\Pr(\sigma|\{\gamma^{(\ell)}, \theta^{(\ell)}\}_{\ell=1}^{k}) = \sum_{\ell=1}^{k} \gamma^{(\ell)} \Pr(\sigma|\theta^{(\ell)})$.*

Definition 3 (Mallows model; [Mallows, 1957])**.** *The Mallows model* (MM) *is specified by a reference ranking $\sigma^* \in \mathcal{L}(A)$ and a dispersion parameter $\phi \in (0, 1]$. The probability of generating a ranking σ is given by $\Pr(\sigma|\sigma^*, \phi) = \frac{\phi^{\mathrm{d_{kt}}(\sigma, \sigma^*)}}{Z}$, where $Z = \sum_{\sigma' \in \mathcal{L}(A)} \phi^{\mathrm{d_{kt}}(\sigma', \sigma^*)}$.*

More generally, a k-mixture Mallows model (k-MM) *is parameterized by $\{\gamma^{(\ell)}, \sigma^*_{(\ell)}, \phi_{(\ell)}\}_{\ell=1}^{k}$, where $\sum_{\ell=1}^{k} \gamma^{(\ell)} = 1$, $\gamma^{(\ell)} \geq 0$ for all $\ell \in [k]$, and $\sigma^*_{(\ell)} \in \mathcal{L}(A), \phi_{(\ell)} \in (0, 1]$ for all $\ell \in [k]$. The probability of generating $\sigma \in \mathcal{L}(A)$ is given by $\Pr(\sigma|\{\gamma^{(\ell)}, \sigma^*_{(\ell)}, \phi_{(\ell)}\}_{\ell=1}^{k}) = \sum_{\ell=1}^{k} \gamma^{(\ell)} \Pr(\sigma|\sigma^*_{(\ell)}, \phi_{(\ell)})$.*

Definition 4 (Uniform distribution)**.** *Under the uniform distribution* (Unif) *supported on a preference profile $\{\sigma^{(i)}\}_{i=1}^{n}$, the probability of generating $\sigma \in \mathcal{L}(A)$ is $\frac{1}{n}$ if $\sigma \in \{\sigma^{(i)}\}_{i=1}^{n}$ and 0 otherwise.*

3 Modeling User Behavior

In this section, we will model the time spent by the user in transforming the recommended ranking σ into the target ranking τ. Our formulation involves the *sorting strategy* of the user (Section 3.1) as well as her *drag-and-drop time* (Section 3.2).

3.1 Sorting Algorithms

A *sorting algorithm* takes as input a ranking $\sigma \in \mathcal{L}(A)$ and performs a sequence of *drag-and-drop* operations until the target ranking is achieved. At each step, an alternative is moved from its current position to another (possibly different) position and the current ranking is updated accordingly. Below we will describe two well-known examples of sorting algorithms: *selection sort* and *insertion sort*. Let $\sigma^{(k)}$ denote the *current* list at time step $k \in \{1, 2, \ldots\}$ (i.e., *before* the sorting operation at time step k takes place). Thus, $\sigma^{(1)} = \sigma$. For any $\sigma \in \mathcal{L}(A)$, define the *$k$-prefix set* of σ as $P_k(\sigma) \coloneqq \{\sigma(1), \sigma(2), \ldots, \sigma(k)\}$ (where $P_0(\sigma) \coloneqq \emptyset$) and corresponding *suffix set* as $S_k(\sigma) \coloneqq A \setminus P_k(\sigma)$.

Selection sort. Let a_i denote the most preferred alternative according to τ in the set $S_{k-1}(\sigma^{(k)})$. At step k of selection sort, the alternative a_i is promoted to a position such that the top k alternatives in the new list are ordered according to τ. Note that this step is well-defined only under the *sorted-prefix property*, i.e., at the beginning of step k of the algorithm, the alternatives in $P_{k-1}(\sigma^{(k)})$ are sorted according to τ. This property is maintained by selection sort.

Insertion sort. Let a_i denote the most preferred alternative in $S_{k-1}(\sigma^{(k)})$ according to $\sigma^{(k)}$. At step k of insertion sort, the alternative a_i is promoted to a position such that the top k alternatives in the new list are ordered according to τ. Note that this step is well-defined only under the sorted-prefix property, which is maintained by insertion sort.

Sorting algorithms. In this work, we will be concerned with sorting algorithms that involve a *combination* of insertion and selection sort. Specifically, we will use the term *sorting algorithm* to refer to a sequence of steps $s_1, s_2, \ldots$ such that each step s_k corresponds to either selection or insertion sort, i.e., $s_k \in \{\text{SEL,INS}\}$ for every k. If $s_k = \text{SEL}$, then the algorithm promotes the most preferred alternative in $S_{k-1}(\sigma^{(k)})$ (according to τ) to a position such that the top k alternatives in the new list are ordered according to τ. If $s_k = \text{INS}$, then the algorithm promotes the most preferred alternative in $S_{k-1}(\sigma^{(k)})$ (according to $\sigma^{(k)}$) to a position such that the top k alternatives in the new list are ordered according to τ. For example, in Figure 1, starting from the recommended list at the extreme left, the user performs a *selection sort* operation (promoting 19 to the top of the current list) followed by an *insertion sort* operation (promoting 30 to its correct position in the sorted prefix $\{19, 22, 40\}$) followed by either selection or insertion sort operation (promoting 23 to its correct position). We will denote a generic sorting algorithm by $\mathcal{A}$ and the class of all sorting algorithms by $\mathfrak{A}$.

Count function. Given a sorting algorithm $\mathcal{A}$, a source ranking $\sigma \in \mathcal{L}(A)$ and a target ranking $\tau \in \mathcal{L}(A)$, the *count* function $f_{\mathcal{A}}^{\sigma\to\tau} : [m-1] \to \mathbb{Z}_+ \cup \{0\}$ keeps track of the *number* of drag-and-drop operations (and the number of *positions* by which some alternative is moved in each such operation) during the execution of $\mathcal{A}$. Formally, $f_{\mathcal{A}}^{\sigma\to\tau}(\ell)$ is the number of times some alternative is 'moved up by ℓ positions' during the execution of algorithm $\mathcal{A}$ when the source and target rankings are σ and τ respectively.[2] For example, let $\mathcal{A}$ be insertion sort, $\sigma = (d, c, a, b)$, and $\tau = (a, b, c, d)$. In step 1, the user considers the alternative d and no move-up operation is required. In step 2, the user promotes c by one position (since $c \succ_\tau d$) to obtain the new list (c, d, a, b). In step 3, the user promotes a by two positions to obtain (a, c, d, b). Finally, the user promotes b by two positions to obtain the target list (a, b, c, d). Overall, the user performs one 'move up by one position' operation and two 'move up by two positions' operations. Hence, $f_{\mathcal{A}}^{\sigma\to\tau}(1) = 1$, $f_{\mathcal{A}}^{\sigma\to\tau}(2) = 2$, and $f_{\mathcal{A}}^{\sigma\to\tau}(3) = 0$. We will write #moves to denote the total number of drag-and-drop operations performed during the execution of $\mathcal{A}$, i.e., $\#\text{moves} = \sum_{\ell=1}^{m-1} f_{\mathcal{A}}^{\sigma\to\tau}(\ell)$.

Remark 1. *Notice the difference between the number of drag-and-drop operations (#moves) and the total distance covered (i.e., the number of positions by which alternatives are moved). Indeed, the above example involves three drag-and-drop operations (#moves $= 3$), but the total distance moved is $0 + 1 + 2 + 2 = 5$. The latter quantity is equal to $\mathrm{d_{kt}}(\sigma, \tau)$.*

Lemma 1. *For any two sorting algorithms $\mathcal{A}, \mathcal{A}' \in \mathfrak{A}$, any $\sigma, \tau \in \mathcal{L}(A)$, and any $\ell \in [m-1]$, $f_{\mathcal{A}}^{\sigma\to\tau}(\ell) = f_{\mathcal{A}'}^{\sigma\to\tau}(\ell)$.*

In light of Lemma 1, we will hereafter drop the subscript $\mathcal{A}$ and simply write $f^{\sigma\to\tau}$ instead of $f_{\mathcal{A}}^{\sigma\to\tau}$. All missing proofs can be found in the full version [Li *et al.*, 2019].

3.2 Drag-and-Drop Time

Weight function. The *weight* function $\mathbf{w} : [m-1] \to \mathbb{R}_{\geq 0}$ models the time taken for each drag-and-drop operation; specifically, $\mathbf{w}(\ell)$ denotes the time taken by the user in moving an alternative up by ℓ positions.[3] Of particular interest to us will be the *linear* weight function $\mathbf{w}_{\text{lin}}(\ell) = \ell$ for each $\ell \in [m-1]$ and the *affine* weight function $\mathbf{w}_{\text{aff}}(\ell) = c\ell + d$ for each $\ell \in [m-1]$ and fixed constants $c, d \in \mathbb{N}$.

Time function. Given the count function $f^{\sigma\to\tau}$ and the weight function $\mathbf{w}$, the *time* function is defined as their inner product, i.e., $\mathtt{time}_{\mathbf{w}}(\sigma, \tau) = \langle f^{\sigma\to\tau}, \mathbf{w}\rangle = \sum_{\ell=1}^{m-1} f^{\sigma\to\tau}(\ell) \cdot \mathbf{w}(\ell)$.

Theorem 1 shows that for the linear weight function $\mathbf{w}_{\text{lin}}$, time is equal to the Kendall's Tau distance, and for the affine weight function, time is equal to a weighted combination of Kendall's Tau distance and the total number of moves.

Theorem 1. *For any $\sigma, \tau \in \mathcal{L}(A)$, $\mathtt{time}_{\mathbf{w}_{\text{lin}}}(\sigma, \tau) = \mathrm{d_{kt}}(\sigma, \tau)$ and $\mathtt{time}_{\mathbf{w}_{\text{aff}}}(\sigma, \tau) = c \cdot \mathrm{d_{kt}}(\sigma, \tau) + d \cdot$ #moves.*

4 Formulation of the Recommendation Problem and Theoretical Results

We model the recommendation problem as the following computational problem: Given the *preference distribution* $\mathcal{D}$ of the user and her `time` function (which, in turn, is determined by the weight function $\mathbf{w}$), find a ranking that minimizes the expected time taken by the user to transform the recommended ranking σ into her preference τ.

Definition 5 ($(\mathcal{D}, \mathbf{w})$-RECOMMENDATION)**.** *Given a distribution $\mathcal{D}$ over $\mathcal{L}(A)$, a weight function $\mathbf{w}$, and a number $\delta \in \mathbb{Q}$, does there exist $\sigma \in \mathcal{L}(A)$ so that $\mathbb{E}_{\tau\sim\mathcal{D}}[\mathtt{time}_{\mathbf{w}}(\sigma, \tau)] \leq \delta$?*

We will focus on settings where $\mathcal{D}$ is Plackett-Luce, Mallows, or Uniform, and $\mathbf{w} \in \{\text{Linear, Affine, General}\}$. Note that if the quantity $\mathbb{E}_{\tau\sim\mathcal{D}}[\mathtt{time}_{\mathbf{w}}(\sigma, \tau)]$ can be computed in polynomial time for a given distribution $\mathcal{D}$ and weight function $\mathbf{w}$, then $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION is in NP.

Our computational results for $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION are summarized in Table 1. We show that this problem is NP-hard, even when the weight function is linear (Theorem 3). On the algorithmic side, we provide a polynomial-time approximation scheme (PTAS) and a 5-approximation algorithm for the linear weight function (Theorems 4 and 5), and an approximation scheme for non-linear weights (Theorem 6).

Theorem 2 (Exact Algorithms)**.** $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION *is solvable in polynomial time when $\mathbf{w}$ is linear and $\mathcal{D}$ is*

[2] Notice that we do not keep track of *which* alternative is moved by ℓ positions. We believe it is reasonable to assume that moving a_1 up by ℓ positions takes the same time as it does for a_2. Also, we do not need to define the count function for *move down* operations as neither selection sort nor insertion sort will ever make such a move.

[3] Here, 'time taken' includes the time spent in *thinking* about which alternative to move as well as actually *carrying out* the move.

either (a) k-mixture Plackett-Luce (k-PL) with $k = 1$, (b) k-mixture Mallows model (k-MM) with $k = 1$, or (c) a uniform distribution with support size $n \leq 2$.

Theorem 3 (Hardness results)**.** $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION *is* NP-complete *even when* $\mathbf{w}$ *is linear and* $\mathcal{D}$ *is either (a) k-mixture Plackett-Luce model (k-PL) for $k = 4$, (b) k-mixture Mallows model (k-MM) for $k = 4$, or (c) a uniform distribution over $n = 4$ linear orders.*

Theorem 4 (PTAS)**.** $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION *admits a polynomial time approximation scheme* (PTAS) *when* $\mathbf{w}$ *is linear and* $\mathcal{D}$ *is either (a) k-mixture Plackett-Luce model (k-PL) for $k \in \mathbb{N}$, (b) k-mixture Mallows model (k-MM) for $k \in \mathbb{N}$, or (c) a uniform distribution* (Unif).

The PTAS in Theorem 4 is quite complicated and is primarily of theoretical interest (indeed, for any fixed $\varepsilon > 0$, the running time of the algorithm is $m^{2^{\tilde{\mathcal{O}}(1/\varepsilon)}}$, making it difficult to be applied in experiments). A simpler and more practical algorithm (although with a worse approximation) is based on the well-known Borda count voting rule (Theorem 5).

Theorem 5 (5-approximation)**.** $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION *admits a polynomial time 5-approximation algorithm when* $\mathbf{w}$ *is linear and* $\mathcal{D}$ *is either (a) k-mixture Plackett-Luce model (k-PL) for $k \in \mathbb{N}$, (b) k-mixture Mallows model (k-MM) for $k \in \mathbb{N}$, or (c) a uniform distribution* (Unif).

Our next result (Theorem 6) provides an approximation guarantee for $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION that applies to *non-linear* weight functions, as long as they are "close" to the linear weight function in the following sense:

Definition 6 (Closeness-of-weights)**.** *A weight function* $\mathbf{w}$ *is said to be (α, β)-close to another weight function* $\mathbf{w}'$ *if there exist $\alpha, \beta \geq 1$ such that for every $\ell \in [m-1]$, we have*

$$\mathbf{w}'(\ell)/\beta \leq \mathbf{w}(\ell) \leq \alpha \mathbf{w}'(\ell).$$

For *any* (possibly non-linear) weight function $\mathbf{w}$ that is (α, β) close to the linear weight function $\mathbf{w}_{\text{lin}}$, Theorem 6 provides an $\alpha\beta$-approximation scheme for $(\mathcal{D}, \mathbf{w})$-RECOMMENDATION.

Theorem 6 (Approximation for general weights)**.** *Given any $\varepsilon > 0$ and any weight function* $\mathbf{w}$ *that is (α, β)-close to the linear weight function* $\mathbf{w}_{\text{lin}}$*, there exists an algorithm that runs in time $m^{2^{\tilde{\mathcal{O}}(1/\varepsilon)}}$ and returns a linear order σ such that*

$$\mathbb{E}_{\tau \sim \mathcal{D}}[\texttt{time}_{\mathbf{w}}(\sigma, \tau)] \leq \alpha\beta(1+\varepsilon)\mathbb{E}_{\tau \sim \mathcal{D}}[\texttt{time}_{\mathbf{w}}(\sigma^*, \tau)],$$

where $\sigma^* \in \arg\min_{\sigma' \in \mathcal{L}(A)} \mathbb{E}_{\tau \sim \mathcal{D}}[\texttt{time}_{\mathbf{w}}(\sigma', \tau)]$.

Remark 2. *Notice that the* PTAS *of Theorem 4 is applicable for any* affine *weight function $\mathbf{w}_{\text{aff}} = c \cdot \mathbf{w}_{\text{lin}} + d$ for some fixed constants $c, d \in \mathbb{N}$. As a result, the approximation guarantee of Theorem 6 also extends to any weight function that is (α, β)-close to* some *affine weight function.*

5 Experimental Results

We perform two sets of experiments on Amazon Mechanical Turk (MTurk). The first set of experiments (Section 5.1) is aimed at identifying the *sorting strategies* of the users as well as a model of their *drag-and-drop behavior*. The observations from these experiments directly motivate the formulation of our theoretical model, which we have already presented in Section 4. The second set of experiments (Section 5.2) is aimed at *evaluating* the practical usefulness of our approach.

In both sets of experiments, the crowdworkers were asked to sort in increasing order randomly generated lists of numbers between 0 and 100. Sections 5.1 and 5.2 provide details about the length of the lists and how they are generated.

In each experiment, the task length was advertised as 10 minutes, and the payment offered was $\$0.25$ per task. The crowdworkers were provided a user interface (see Figure 1) that allows for drag-and-drop operations. To ensure data quality, we removed those workers from the data who failed to successfully order the integers more than 80% of the time, or did not complete all the polls. We also removed the workers with high variance in their sorting time; in particular, those with coefficient of variation above the 80^{th} percentile. The reported results are for the workers whose data was retained.

5.1 Identifying User Behavior

To identify user behavior, we performed two experiments: (a) RANK10, where each crowdworker participated in 20 polls, each consisting of a list of 10 integers (between 0 and 100) generated uniformly at random, and (b) RANK5, which is a similar task with 30 polls and lists of length 5. In each poll, we recorded the time taken by a crowdworker to move an alternative (via drag-and-drop operation) and the number of positions by which the alternative was moved. After the initial pruning (as described above), we retained 9840 polls submitted by 492 workers in the RANK10 experiment, and 10320 polls submitted by 344 workers retained in the RANK5 experiment. Table 2 summarizes the aggregate statistics. Our observations are discussed below.

Sorting behavior. Our hypothesis regarding the ranking behavior of human crowdworkers was that they use (some combination of) natural sorting algorithms such as selection sort or insertion sort (Section 3.1). To test our hypothesis, we examined the fraction of the drag-and-drop operations that *coincided* with an iteration of selection/insertion sort. (Given a ranking σ, a drag-and-drop operation on σ coincides with selection/insertion sort if the order of alternatives resulting from the drag-and-drop operation exactly matches the order of alternatives when one iteration of either selection or insertion sort is applied on σ.) We found that, on average, $\frac{2.21}{2.91} = 76\%$ of all drag-and-drop operations in RANK5 (and $\frac{5.09}{7.69} = 66.2\%$ in the RANK10) coincided with selection/insertion sort.

Drag-and-drop behavior. To identify the drag-and-drop behavior of the users, we plot the time-to-rank as a function of the total number of positions by which the alternatives are moved in each poll (Figure 3). Recall from Remark 1 that for an ideal user who uses only insertion/selection sort, the latter quantity is equal to $\mathrm{d}_{\text{kt}}(\sigma, \tau)$. Our hypothesis was that the sorting time varies *linearly* with the total number of drag-and-drop operations (#moves) and the Kendall's Tau distance ($\mathrm{d}_{\text{kt}}(\sigma, \tau)$). To verify this, we used linear regression with time-to-rank (or sorting time) as the target variable and measured the mean squared error (MSE) using 5-fold cross-validation

	RANK10			RANK5		
	Mean	Median	Std. Dev.	Mean	Median	Std. Dev.
Sorting time	24.41	22.65	9.12	7.75	6.99	3.54
Total number of drag-and-drop operations	7.69	8	1.8	2.91	3	1.13
Total number of positions moved during drag-and-drop operations	22.59	23	5.59	5.05	5	2.01
Number of operations coinciding with selection/insertion sort	5.09	6	2.28	2.21	2	1.06
Kendall's Tau distance between the initial and final rankings	22.55	22	5.6	5.04	5	2.01

Table 2: Summary of the user statistics recorded in the experiments in Section 5.1.

Dataset	Avg. MSE (in seconds^2)	$\sqrt{\text{Avg. MSE}}$ (in seconds)	Avg. Sorting Time (in seconds)	Number of users based on their best-fit model		
				Only d_{kt}	Only #moves	Both d_{kt} and #moves
RANK10	42.98	6.56	24.41	217	199	76
RANK5	7.74	2.78	7.75	138	180	26

Table 3: Average 5-fold cross-validation MSE over all workers using the best model for each worker, and the number of users for which each of the models was identified to be the best. # moves is the number of times alternatives are moved using selection or insertion sort.

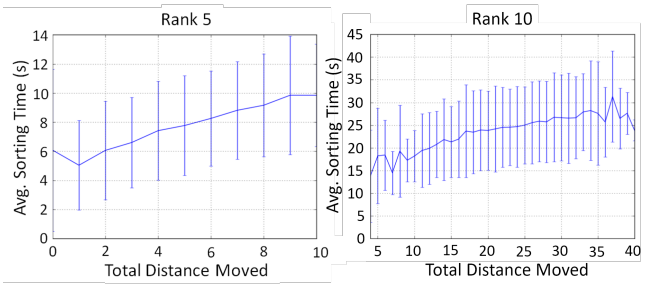

Figure 3: Relationship between the number of positions moved and the total sorting time for RANK5 (left) and RANK10 (right).

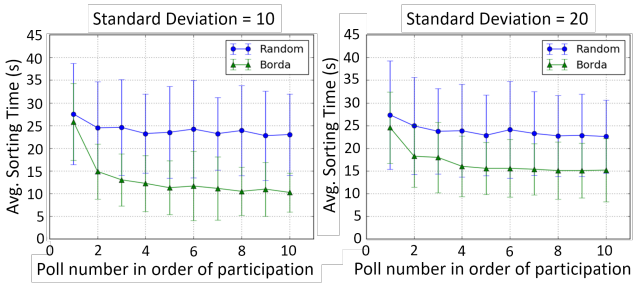

Figure 4: Relationship between sorting time and the number of polls completed by the users for std dev=10 (left) and std dev=20 (right).

for three different choices of independent variables: (1) Only d_{kt}, (2) only #moves, and (3) both d_{kt} and #moves. For each user, we picked the model with the smallest MSE (see Table 3 for the resulting distribution of the number of users). We found that the predicted drag-and-drop times (using the best-fit model for each user) are, on average, within $\frac{6.56}{24.41} = 26.8\%$ of the observed times for RANK10 and within $\frac{2.78}{7.75} = 35.8\%$ for RANK5.

5.2 Evaluating the Proposed Framework

To evaluate the usefulness of our framework, we compared a random recommendation strategy with one that forms an increasingly accurate estimate of users' preferences with time. Specifically, we first fix the ground truth ranking of 10 alternatives consisting of randomly generated integers between 0 and 100. Each crowdworker then participates in two sets of 10 polls each. In one set of polls, the crowdworkers are provided with initial rankings generated by adding independent Gaussian noise to the ground truth (to simulate a *random* recommendation strategy), and their sorting times are recorded.

In the second set of polls, the recommended set of alternatives is the same as under the random strategy but ordered order to a *Borda* ranking. Specifically, the ordering in the k^{th} iteration is determined by the Borda ranking aggregated from the previous $k-1$ iterations.

Figure 4 shows the average sorting time of the crowdworkers as a function of the index of the polls under two different noise settings: std. dev. = 10 and std. dev. = 20.

We observe that Borda recommendation strategy (in green) provides a significant reduction in the sorting time of the users compared to the random strategy (in blue). Indeed, the sorting time of the users is reduced by *up to* 50%, thus validating the practical usefulness of our framework. Note that the reduction in sorting time is *not* due to increasing familiarity with the interface. This is because the average sorting time for the random strategy remains almost constant throughout the duration of the poll.

6 Future Work

Our work opens up a number of directions for future research. First, it would be interesting to analyze the complexity of the recommendation problem for other distance measures, e.g., Ulam distance. Second, it would be interesting to analyze the effect of cognitive biases such as the *framing effect* [Tversky and Kahneman, 1981] and *list position bias* [Lerman and Hogg, 2014] on the recommendation problem. Progress in this direction can, in turn, have implications on the *fairness* of recommendation algorithms.

Acknowledgments

We are grateful to IJCAI-19 reviewers for their helpful comments. This work is supported by NSF #1453542 and ONR #N00014-17-1-2621.

References

[Agarwal, 2016] Shivani Agarwal. On Ranking and Choice Models. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 4050–4053, 2016.

[Albert and Tullis, 2013] William Albert and Thomas Tullis. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Newnes, 2013.

[Awasthi *et al.*, 2014] Pranjal Awasthi, Avrim Blum, Or Sheffet, and Aravindan Vijayaraghavan. Learning Mixtures of Ranking Models. In *Proceedings of Advances in Neural Information Processing Systems*, pages 2609–2617, 2014.

[Bevan *et al.*, 2015] Nigel Bevan, James Carter, and Susan Harker. ISO 9241-11 revised: What have we Learnt about Usability Since 1998? In *International Conference on Human-Computer Interaction*, pages 143–151, 2015.

[Blum *et al.*, 2004] Avrim Blum, Jeffrey Jackson, Tuomas Sandholm, and Martin Zinkevich. Preference Elicitation and Query Learning. *Journal of Machine Learning Research*, 5:649–667, 2004.

[Boutilier, 2013] Craig Boutilier. Computational Decision Support: Regret-Based Models for Optimization and Preference Elicitation. In P. H. Crowley and T. R. Zentall, editors, *Comparative Decision Making: Analysis and Support Across Disciplines and Applications*. Oxford University Press, 2013.

[Braverman and Mossel, 2008] Mark Braverman and Elchanan Mossel. Noisy Sorting Without Resampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 268–276, 2008.

[Busa-Fekete *et al.*, 2014] Róbert Busa-Fekete, Eyke Hüllermeier, and Balázs Szörényi. Preference-Based Rank Elicitation using Statistical Models: The Case of Mallows. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*, pages II:1071–1079, 2014.

[Conitzer and Sandholm, 2002] Vincent Conitzer and Tuomas Sandholm. Vote Elicitation: Complexity and Strategy-Proofness. In *Eighteenth National Conference on Artificial Intelligence*, pages 392–397, 2002.

[Guiver and Snelson, 2009] John Guiver and Edward Snelson. Bayesian Inference for Plackett-Luce Ranking Models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML-09, pages 377–384, 2009.

[Kendall, 1938] Maurice G Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.

[Khetan and Oh, 2016] Ashish Khetan and Sewoong Oh. Data-Driven Rank Breaking for Efficient Rank Aggregation. *Journal of Machine Learning Research*, 17(193):1–54, 2016.

[Laskowski *et al.*, 2004] Sharon J Laskowski, Marguerite Autry, John Cugini, and William Killam. Improving the Usability and Accessibility of Voting Systems and Products. *NIST Special Publication*, 500:256, 2004.

[Lerman and Hogg, 2014] Kristina Lerman and Tad Hogg. Leveraging Position Bias to Improve Peer Recommendation. *PloS one*, 9(6):e98914, 2014.

[Li *et al.*, 2019] Haoming Li, Sujoy Sikdar, Rohit Vaish, Junming Wang, Lirong Xia, and Chaonan Ye. Minimizing Time-to-Rank: A Learning and Recommendation Approach. *arXiv preprint arXiv:1905.11984*, 2019.

[Liu, 2011] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2011.

[Lu and Boutilier, 2014] Tyler Lu and Craig Boutilier. Effective Sampling and Learning for Mallows Models with Pairwise-Preference Data. *Journal of Machine Learning Research*, 15:3963–4009, 2014.

[Luce, 1959] Robert Duncan Luce. *Individual Choice Behavior: A Theoretical Analysis*. Wiley, 1959.

[Mallows, 1957] Colin L. Mallows. Non-Null Ranking Model. *Biometrika*, 44(1/2):114–130, 1957.

[Negahban *et al.*, 2017] Sahand Negahban, Sewoong Oh, and Devavrat Shah. Rank Centrality: Ranking from Pairwise Comparisons. *Operations Research*, 65(1):266–287, 2017.

[Plackett, 1975] Robin L. Plackett. The Analysis of Permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202, 1975.

[Procaccia and Shah, 2016] Ariel D Procaccia and Nisarg Shah. Optimal Aggregation of Uncertain Preferences. In *Thirtieth AAAI Conference on Artificial Intelligence*, pages 608–614, 2016.

[Soufiani *et al.*, 2013] Hossein Azari Soufiani, David C Parkes, and Lirong Xia. Preference Elicitation For General Random Utility Models. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 596–605, 2013.

[Tversky and Kahneman, 1981] Amos Tversky and Daniel Kahneman. The Framing of Decisions and the Psychology of Choice. *Science*, 211(4481):453–458, 1981.

[Xia, 2019] Lirong Xia. *Learning and Decision-Making from Rank Data*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019.

[Zhao and Xia, 2018] Zhibing Zhao and Lirong Xia. Composite Marginal Likelihood Methods for Random Utility Models. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5922–5931, 2018.

[Zhao *et al.*, 2016] Zhibing Zhao, Peter Piech, and Lirong Xia. Learning Mixtures of Plackett-Luce Models. In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*, 2016.

[Zhao *et al.*, 2018] Zhibing Zhao, Haoming Li, Junming Wang, Jeffrey Kephart, Nicholas Mattei, Hui Su, and Lirong Xia. A Cost-Effective Framework for Preference Elicitation and Aggregation. In *Proceedings of the 2018 Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.